

Implémentation temps réel des algorithmes de tri dans les applications de transports intelligents en se basant sur l'outil de synthèse haut niveau HLS

Yomna Ben Jmaa Chtourou

▶ **To cite this version:**

Yomna Ben Jmaa Chtourou. Implémentation temps réel des algorithmes de tri dans les applications de transports intelligents en se basant sur l'outil de synthèse haut niveau HLS. Systèmes embarqués. Université de Valenciennes et du Hainaut-Cambresis; École nationale d'ingénieurs de Sfax (Tunisie), 2019. Français. NNT : 2019VALE0013 . tel-02638412

HAL Id: tel-02638412
https://theses.hal.science/tel-02638412

Submitted on 28 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Polytechnique
HAUTS-DE-FRANCE

Thèse de doctorat

Pour obtenir le grade de Docteur de

l'UNIVERSITE POLYTECHNIQUE HAUTS-DE-FRANCE

et de L'Ecole Nationale des Ingénieurs de Sfax

Disciplines :
Ingénierie des Systèmes Informatiques (ENIS)/
Informatique (UPHF)

Présentée et soutenue par Yomna BEN JMAA CHTOUROU

Le 27/04/2019, à l'ENIS

Ecole doctorale :
Sciences Pour l'Ingénieur (ED SPI 072)
Sciences et Technologies

Equipe de recherche, Laboratoire :
Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH - UMR 8201)
Laboratoire de recherche en Développement et contrôle d'applications distribués (ReDCAD)

Implémentation temps réel des algorithmes de tri dans les applications de transports intelligents en se basant sur l'outil de synthèse haut niveau HLS

JURY

Président du jury

- Abderrazek Jemai (Professeur à l'INSAT)

Rapporteurs

- Carlos Valderrama (Professeur à l'UMONS).
- Abderrazek Jemai (Professeur à l'INSAT).

Examinateurs

- Mouna Baklouti (Maître de conférence à l'ENIS)

Co-directeurs de thèse :

- Rabie Ben Atitallah (Maître de Conférence HDR à l'UPHF)
- Mohamed Abid (Professeur à l'ENIS)

Co-encadrant :

- David Duvivier (Professeur à l'UPHF)

Dédicace

Je dédie ce travail...

À tous celles et ceux qui m'ont permis de rendre le rêve d'être docteur possible

A mes très chers parents Mohamed et Mounira

Sources de mes joies, secrets de ma force, Vous serez toujours mon modèle.

Papa, dans ta détermination, ta force et ton honnêteté.

Maman dans ta bonté, ta patience et ton dévouement.

Merci pour tous vos sacrifices pour que vos enfants grandissent et prospèrent. Merci d'être tout simplement mes parents. C'est à vous que je dois cette réussite. Je suis fière de vous l'offrir.

À mon cher époux Naim

Aucun mot ne saurait t'exprimer mon profond attachement et ma reconnaissance pour l'amour, la tendresse et la gentillesse dont tu m'as toujours entouré. Cher mari j'aimerai bien que tu trouves dans ce travail l'expression de mes sentiments de reconnaissance les plus sincères car grâce à ton aide et à ta patience avec moi que ce travail a pu voir le jour.

À ma petite princesse charmante Kmayra

C'est à toi mon adorable ange, ma joie, mon petit trésor que maman dédie ce travail. Sans toi, je ne connaîtrais pas le plaisir d'être maman.

À mes beaux-parents Najib et Najet

Pour leurs respects, leurs soutiens et leurs encouragements. J'espère que mon travail sera le témoignage de mon respect et de mes sentiments les plus sincères

À la mémoire de mon cher directeur Maher BEN JEMAA

Aucune dédicace ne saurait exprimer mes respects, ma reconnaissance et mon profond amour à mon cher professeur et père scientifique M. Maher qui a dirigé cette thèse.

Qu'il se repose en paix et j'espère qu'il puisse trouver dans ce travail le fruit de toutes ses peines et de tous ses efforts.

En ce jour, j'espère réaliser l'un de ses rêves.

À ma chère soeur Yosra et à la mémoire de son mari Maher
Pour avoir partagé mes enthousiasmes et mes déceptions. En reconnaissances de tous les sacrifices pour me permettre d'atteindre cette étape de ma vie.
Avec un amour particulier pour Maysen et Yessine.
À mon cher frére Ahmed leur conjoint Amira
Pour leurs amour, aide et pour avoir donné du goût et du sens à notre vie de famille. Un amour particulier pour Rourou, Ranouma
À mes beau frére Nabil leur conjoint Amina
À toute la famille BEN JMAA et la famille CHTOUROU
À tous mes amis qui suivaient mon parcours de près avec bienveillance surtout Rania, Amina, Salma, Ibtissem, Lamia, Hana, Manel, Zeineb, Rim, Nabila, Nesrine, Wafa.
À toutes les personnes qui ont servi pour ma formation, mon éducation et mon enseignement.

Yomna Ben Jmaa

Remerciements

C'est avec un grand plaisir que je réserve cette page en signe de gratitude et de reconnaissance à tous ceux qui ont assisté ce travail. Cette thèse s'inscrit dans le cadre d'une cotutelle entre laboratoire de recherche en développement et contrôle des applications distribués (ReDCAD) de l'ENIS à Sfax, Tunisie ainsi que le Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH UMR CNRS 8201) de l'Université Polytechnique Hauts-de-France (UPHF), France.

C'est avec une grande tristesse que je saisis cette occasion pour remercier mon directeur de thèse décédé, Professeur Maher Ben Jemaa, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion. Le malheur de l'avoir perdu ne doit pas faire oublier le bonheur de l'avoir connu par son dévouement et ses qualités humaines et scientifiques exemplaires envers ses collègues et envers moi en particulier. Que Dieu le tout puissant lui accorde son infinie miséricorde et l'accueille dans son éternel Paradis.

Aprés le décés du Professeur Maher Ben Jemaa, c'est au Professeur M. Mohamed Abid que revient le mérite d'avoir suivi de très près mon travail et de le mener à bien jusqu'à sa réalisation totale.

Je tiens à remercier mes directeurs de thèse, M. Rabie Ben Atitallah, Maître de Conférences habilité à l'Université Polytechnique Hauts-de-France et M. Mohamed Abid, Professeur à l'ENIS, pour la confiance qu'ils m'ont accordée en acceptant la direction scientifique de mes travaux. Je leur suis reconnaissante de m'avoir fait bénéficier tout au long de ce travail de leur grande compétence et de leurs conseils que je n'oublierai jamais.

Mes vifs remerciements s'adressent également à mon co-encadrant de thèse M. David Duvivier, Professeur à l'UPHF, pour la qualité de son encadrement, ses conseils et ses remarques toujours judicieux dont il a su me faire bénéficier, ainsi que pour ses aides, ses compétences et sa disponibilités afin d'améliorer grandement la qualité de mes travaux et de ce mémoire. Il m'a aidée, avec ses précieuses remarques à l'accomplissement et l'aboutissement de ce travail.

Je tiens aussi à exprimer mes sincères respects à Monsieur Abderrazek JEMAI,

Professeur à INSAT qui, malgré ses nombreuses occupations, a accepté de présider le jury de cette thèse.

Je remercie vivement M. Abderrazak JEMAI Professeur à l'Institut Nationale des sciences appliquées et de technologie, Tunisie et M. Carlos Valderrama, Professeur à l'Université de Mons, Belgique pour l'intérêt qu'ils ont apporté à mes travaux en acceptant d'évaluer et de rapporter ce mémoire, ainsi que pour leurs participations au jury.

Je tiens à remercier Madame Mouna BAKLOUTI, Maître de conférence à l'ENIS , pour l'honneur qu'il me fait en acceptant de juger ce travail en tant qu'examinateur. Merci pour l'intérêt et la considération que vous avez porté sur ce travail.
Je tiens à remercier, aussi, tous les membres de l'équipe ReDCAD-Sfax et de l'équipe LAMIH pour leurs encouragements, leurs échanges et leurs aides.

Je remercie tous mes ami(e)s de près ou de loin à la réalisation de ce travail.

Enfin, je remercie sincèrement ma famille, et plus précisément mes parents, mes beaux parents, mon mari et ma fille pour leurs patiences, leurs encouragements et leurs soutiens. Ils m'ont toujours encouragée à atteindre mes objectifs et mes ambitions malgré tous les problèmes rencontrés.

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude

Table des matières

1 Introduction générale **1**

1.1 Objectifs et contributions 3

1.2 Plan de thèse 4

2 Étude générale sur l'utilisation des algorithmes de tri dans les systèmes de transports intelligents **5**

2.1 Introduction 6

2.2 Systèmes de transport intelligents (STI) 6

2.2.1 Historique des STI 6

2.2.2 Contexte des STI 7

2.2.3 Systèmes avioniques 7

2.3 État de l'art 14

2.4 Vue globale sur les plateformes d'exécution et les algorithmes de tri 19

2.4.1 Plateformes d'exécution 19

2.4.2 Algorithmes de tri 21

2.5 Étude de la complexité des algorithmes de tri 28

2.5.1 Permutation 29

2.5.2 Méthode de Lehmer 30

2.5.3 Temps d'exécution et écart type 31

2.5.4 Dispersion statistique 42

2.6 Conclusion 43

3 Implémentation logicielle des algorithmes de tri sur la plateforme Zynq Zedboard **44**

3.1 Introduction 45

3.2 Architecture du SoPC et méthodologie de conception 45

3.2.1 Généralités 45
3.2.2 Architecture du SoPC 46
3.3 Présentation de la carte Xilinx Zedboard 49
3.3.1 La partie programmable FPGA : Famille Xilinx Zynq 7000 49
3.3.2 Système microprocesseur (PS) 53
3.3.3 Outils de synthèse et d'analyse de conceptions HDL 54
3.4 Implémentation logicielle des algorithmes de tri 55
3.4.1 Environnements de développement 55
3.4.2 Étude de la complexité des algorithmes de tri 57
3.5 Conclusion 68

4 Conception et synthèse haut niveau des algorithmes de tri 70
4.1 Introduction 71
4.2 Évolution des méthodologies de conception haut niveau 71
4.2.1 Avantages du HLS 72
4.2.2 Les problèmes du HLS 73
4.3 Génération de l'IP matériel par HLS 74
4.3.1 Synthèse et optimisation en RTL 75
4.3.2 Développement des accélérateurs matériels avec Vivadeo HLS . . . 78
4.3.3 Intégration des IPs dans un environnement SW/HW avec HLS . . . 82
4.4 Evaluation des performances des algorithmes de tri dans un environnement SW/HW 83
4.5 Conclusion 93

5 Conclusion générale 94
5.1 Synthèse 94
5.2 Perspectives 95
5.2.1 Perspectives à court et à moyen terme 95
5.2.2 Perspectives à long terme 96

A Annexe 97
A.1 Vue globale sur les représentations graphiques avec Boite à moustaches . . 97
A.2 Résultats expérimentaux 99
A.2.1 Exécution sur le processeur CPU 99
A.2.2 Exécution sur le processeur ARM de FPGA 111

Bibliographie 117

Liste des figures

1.1 Évolution chronologique de la densité d'intégration 2
1.2 Co-design : Conception conjointe logicielle/matérielle 3

2.1 Chemin le plus court en appliquant l'algorithme A* 11
2.2 Algorithme BubbleSort 22
2.3 Algorithme InsertionSort 22
2.4 Algorithme SelectionSort 23
2.5 Algorithme HeapSort 24
2.6 Algorithme ShellSort 25
2.7 Algorithme QuickSort 26
2.8 Algorithme MergeSort 27
2.9 Algorithme TimSort 29
2.10 Temps d'exécution sur le processeur 32
2.11 Temps d'exécution des algorithmes HeapSort, QuickSort, MergeSort, ShellSort et TimSort sur le processeur 32
2.12 Ecart type des temps d'exécution pour les algorithmes de tri en l'exécutant sur le CPU 33
2.13 Nombres aberrantes 35
2.14 Boxplots de l'algorithme MergeSort avec N>64 36
2.15 Médiane et Moyenne de l'algorithme MergeSort avec N>64 38
2.16 Boxplots de l'algorithme InsertionSort avec N<=64 39
2.17 La Médiane et la moyenne de l'algorithme InsertionSort avec N<=64 . . . 41
2.18 MergeSort 42
2.19 InsertionSort 42

3.1 Diagramme d'évolution des technologies de conception sur FPGA 46
3.2 Flot de conception SW/HW 48

3.3 Vue globale de Vivado HLS 48
3.4 Schéma synoptique de la carte Xilinx Zedboard 50
3.5 Architecture de l'FPGA Zynq 7000 51
3.6 Architecture de l'FPGA Zynq 7020 51
3.7 Architecture du *Slice* 52
3.8 Architecture du bloc DSP 52
3.9 Architecture du bloc RAM 52
3.10 La partie PS du Zynq 702 53
3.11 Communication entre PS et PL avec le bus AXI4-Lite 54
3.12 Communication entre PS et PL avec le bus AXI4-Stream 54
3.13 Les fonctions définies dans la bibliothèque *"Fat File System"* 57
3.14 Module de la consommation d'énergie 59
3.15 Temps d'exécution des algorithmes de tri exécutés sur le processeur 59
3.16 Zoom des cinq algorithmes de tri 60
3.17 Interface de sortie de la consommation de puissance 61
3.18 Boxplots de l'algorithme TimSort sur le processeur ARM 63
3.19 Boxplots de l'algorithme ShellSort sur le processeur ARM 66
3.20 ShellSort 68
3.21 TimSort 68

4.1 Flot de conception du HLS 72
4.2 Les étapes de HLS 72
4.3 Fenêtre de choix et de configuration de directives 75
4.4 Flot de conception du Vivado HLS 76
4.5 Exemple de boucle optimisée avec le directive UNROLL 76
4.6 Exemple de boucle optimisée avec le directive MERGE 77
4.7 Exemple de boucle optimisée en PIPELINE 77
4.8 Exemple de fonction optimisée avec le directive INLINE 78
4.9 Exemple de fonction optimisée avec le directive UNROLL avec factor=2 . . 79
4.10 Démarche de l'algorithme MergeSort 79
4.11 Implémentation Hardware optimisée de l'algorithme MergeSort 80
4.12 Architecture Hardware des algorithmes de tri 81
4.13 Bloc DMA 81
4.14 Communication entre la carte SD et FPGA 82
4.15 Exemple de design SW/HW conçu avec Vivadeo 83

4.16 Comparaison entre les algorithmes de tri en termes de temps d'exécution Si N<64 avec des données codées sur 32 bits 86
4.17 Comparaison entre les algorithmes de tri en termes de temps d'exécution avec des données codées sur 64 bits Si N<64 86
4.18 Comparaison entre les algorithmes de tri en termes de temps d'exécution avec des données codées sur 32 bits 88
4.19 Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 32 bits 89
4.20 Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 64 bits 89
4.21 Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 64 bits 90
4.22 Comparaison de temps d'exécution entre TimSort, HeapSort, ShellSort et MergeSort pour N>=64 et avec des données codées sur 32 bits 91

A.1 Informations nécessaires pour caractériser un Boxplots 98
A.2 Boxplots de l'algorithme MergeSort sur CPU 100
A.3 Boxplots de l'algorithme TimSort sur CPU 101
A.4 Variation temporelle de l'algorithme TimSort sur CPU 103
A.5 Boxplots de l'algorithme ShellSort sur CPU 105
A.6 Variation temporelle de l'algorithme ShellSort sur CPU 107
A.7 Boxplots de l'algorithme HeapSort sur CPU 108
A.8 Variation temporelle de l'algorithme HeapSort sur CPU 110
A.9 Boxplots de l'algorithme MergeSort sur le processeur ARM 111
A.10 Boxplots de l'algorithme HeapSort sur le processeur ARM 114

Liste des tableaux

2.1 Complexité des algorithmes de tri 14
2.2 Temps d'exécution et écart type des algorithmes de tri sur le processeur Intel 31
2.3 Coefficient de variation des algorithmes de tri sur le processeur 33
2.4 Pourcentages d'outliers pour l'algorithme MergeSort 35
2.5 Critères de performances pour l'algorithme MergeSort (N>64) 37
2.6 Critères de performances pour l'algorithme InsertionSort (N<=64) 40
2.7 Pourcentage d'outliers pour InsertionSort 41

3.1 Temps d'exécution des algorithmes de tri sur ARM Cortex A9 58
3.2 Écart type pour les algorithmes de tri N<=64 61
3.3 Ecart type pour les meilleurs algorithmes de tri N>64 61
3.5 Coefficient de variation pour les algorithmes de tri N<=64 62
3.6 Coefficient de variation pour les meilleurs algorithmes de tri N>64 62
3.4 Consommation d'énergie des algorithmes de tri 62
3.7 Critères de performance pour l'algorithme TimSort N>64 64
3.8 Pourcentage d'outliers pour TimSort sur ARM 65
3.9 Boxplots de l'algorithme ShellSort sur le processeur ARM 67
3.10 Pourcentage d'outliers pour ShellSort sur ARM 68

4.1 Temps d'exécution en microseconde des algorithmes de tri avec des données codées sur 4 et 8 octets pour N<=64 85
4.2 Temps d'exécution en microseconde des algorithmes de tri avec des données codées sur 4 et 8 octets pour N>64 87
4.3 Écart type des algorithmes de tri (N<64) 90
4.4 Écart type pour les meilleures algorithmes de tri avec N >=64 90
4.5 Ressources d'utilisation pour les algorithmes de tri 92

A.1 Critères de performance pour l'algorithme TimSort sur CPU 102
A.2 Pourcentage d'outliers pour l'algorithme TimSort sur CPU 103
A.3 Critères de performances pour l'algorithme ShellSort sur CPU 106
A.4 Pourcentages d'outliers pour l'algorithme ShellSort sur CPU 107
A.5 Critères de performances de l'algorithme HeapSort sur CPU 109
A.6 Pourcentage d'outliers pour HeapSort sur CPU 110
A.7 Critères de performances de l'algorithme MergeSort sur ARM 112
A.8 Pourcentage d'outliers pour MergeSort sur ARM 113
A.9 Critères de performances de l'algorithme HeapSort sur ARM 115
A.10 Pourcentage d'outliers pour HeapSort sur ARM 116

Glossaire

ITS :	Intelligent Transportation Systems
SoC :	System on Chip
FPGA :	Field-Programmable Gate Array
SoPC :	System On Programmable Chip
VHDL/Verilog :	VHSIC Hardware Description Language
HLS :	High Level Synthesis
RTL :	Register-Transfer-Level
OP :	optimal parameter
CPU :	Central Processsing Unit
GPU :	Graphics Processing Unit
ASIC :	Application Specific Integrated Circuit
IP :	Intellectual Property
BSP :	Board Support Package
PS :	Processing System
PL :	Programmable Logic
CLB :	Configurable Logic Block
LUT :	LookUp Table
FIFO :	First in First out
MAC :	Multiply and ACcumulate
DSP :	Digital Signal Processor
AMBA :	Advanced Microcontroller Bus Architecture
AXI4 :	Advanced eXtensible Interface
DMA :	Direct Memory Access
MCDM :	Multiple Criteria Decision Making
AHP :	Analytical hierarchy Process
ELECTRE :	ELimination and Choice Expressing Reality
DSS :	Decision Support System

Chapitre 1

Introduction générale

De nos jours, les systèmes électroniques embarqués sont de plus en plus utilisés dans la majorité des secteurs d'activités. Nous citons quelques domaines d'application : transport, domaine grand public, médicale, télécommunication, industriels et bureautique. Différents objets intègrent des systèmes électroniques embarqués tels que les systèmes de transport intelligent, le téléphone portable, les télécommunications, le multimédia, le médical, la domotique. Pour des raisons de fiabilité et de rentabilité, ces systèmes sont essentiellement conçus pour effectuer des tâches complexes et respecter les contraintes de temps, de consommation d'énergie et de coût de fabrication : À titre d'exemple, nous citons les algorithmes de planification de plan de vol pour trouver le chemin le plus court ou proposer des trajectoires d'évitement en temps réel. Ces algorithmes intègrent des algorithmes de tri.

Les systèmes de transport intelligents désignent des applications des nouvelles technologies de l'information et de la communication. Ces systèmes se présentent dans plusieurs domaines d'activités :
– Dans l'optimisation de l'utilisation des infrastructures de transport ;
– Dans l'amélioration de sécurité ou de la sureté de fonctionnement ;
– Ainsi que dans le développement des services.
Les technologies utilisées dans les systèmes de transport intelligents varient, allant de systèmes de gestion basiques comme Systèmes de navigation, Évitement de collisions et suivi d'obstacles. Ces systèmes doivent offrir un calcul haute performance, une adaptabilité selon le paramètre de l'environnement, fiabilité du système, etc.

Les applications de transport intelligents nécessite des systèmes décisionnels qui sont capable d'analyser l'environnement et l'état du système , de comparer plusieurs solutions possibles et de prendre des décisions. Ces systèmes ont besoin des algorithmes de tri. L'objectif principal de ce travail est de proposer une implémentation matérielle efficace et optimisée des algorithmes de tri issu des systèmes avioniques sur FPGA en se basant sur l'outil de synthèse haut niveau HLS pour améliorer les performances en termes de temps d'exécution, d'écart type et des ressources utilisées.

Avec l'évolution rapide de la densité d'intégration sur puce, il est devenu possible d'intégrer une architecture mixte à base de processeurs et d'un ensemble de composants matériels spécifiques (ASIC, FPGA) sur une seule puce nommée système sur puce ou SoC (System on Chip)comme montre la Figure 1.1. Le concept du SoC a été emprunté du développement de systèmes complexes sur technologie ASIC (Application-Specific In-

tegrated Circuit). Cependant, il a été étendu pour le développement sur cible FPGA (Field-Programmable Gate Array). On parlera alors de SoPC (System On Programmable Chip). [1] [1].

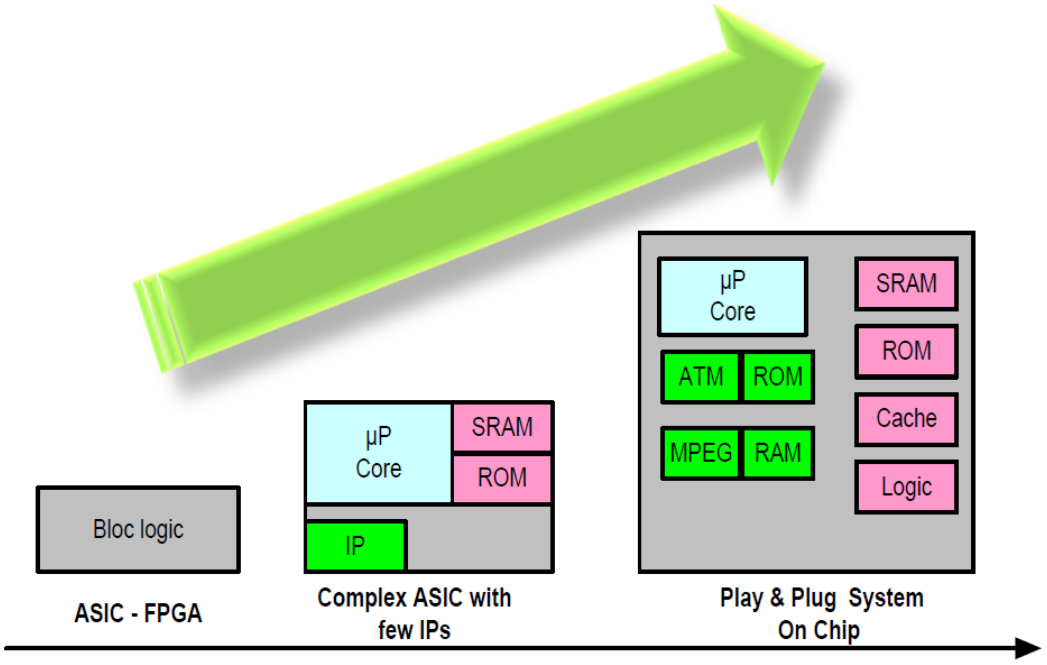


FIGURE 1.1 – Évolution chronologique de la densité d'intégration

Le développement de SOPC nécessite à la fois des éléments de conception matérielle et logicielle. Le logiciel est généralement écrit (C ou C ++) et compilé à l'aide d'un outil autonome (environnement de développement intégré d'Altera pour Nios II ou Embedded de Xilinx). La partie matérielle est généralement implémentée à l'aide d'outils spécifiques fournis dans la suite CAO du fabricant FPGA, qui peut également être utilisée pour la conception FPGA traditionnelle (Quartus II d'Altera et ISE de Xilinx). La complexité de la conception des systèmes sur puces ne cesse de croitre pour plusieurs raisons. Nous citons par exemple :

- L'augmentation du niveau d'intégration des transistors sur la même puce,
- Le déploiement des architectures hétérogènes,
- L'augmentation de la complexité des applications,
- L'importance du nombre de contraintes qu'il faut respecter (Contraintes non fonctionnelles : temps de mise sur le marché, coût du système, fiabilité, etc et les contraintes fonctionnelles : puissance de calcul, consommation, reconfiguration, etc.

La solution efficace pour résoudre ce problème complexe est d'utiliser la conception conjointe logicielle/matérielle (Codesign). La figure 1.2 montre les étapes de flot de conception SW/HW sur FPGA. Tout d'abord, la spécification de la conception permet de définir l'architecture adéquat. Cet algorithme est ensuite rédigé en langage de programmation matérielle HDL. On peut par la suite simuler le comportement de ce circuit afin de corriger les erreurs s'ils existent. Cette étape est suivie d'une phase de synthèse et de placement et routage. En effet, chaque phase nécessite une vérification du bon fonctionnement de l'architecture globale. Le fichier binaire généré est finalement chargé dans la carte FPGA cible. Parallèlement, on lance le développement de la partie logicielle qui nécessite une

1. System On Programmable Chip

rédaction du code en langage de programmation adéquate avec l'outil de développement. Cet outil compile et puis génère le fichier exécutable du code source qui sera finalement exécuter par le processeur de l'FPGA.

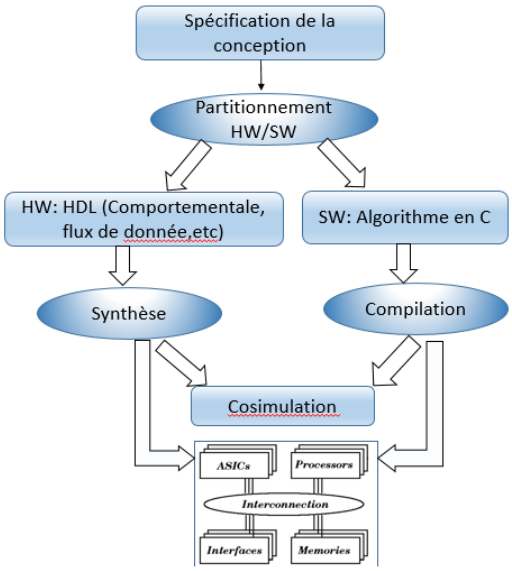


FIGURE 1.2 – Co-design : Conception conjointe logicielle/matérielle

La complexité croissante des applications embarquées sur puce ont conduit les méthodologies de conception à des niveaux d'abstraction plus élevés [2, 3] et par conséquent, les outils de synthèse de haut niveau HLS *(High Level Synthesis)* ont évolué pour augmenter la productivité des conceptions basées sur FPGA. En effet, les premières générations de HLS répondaient aux attentes des concepteurs de matériel ; différentes raisons ont encouragé les chercheurs à les améliorer pour produire des conceptions matérielles plus efficaces. Nous pouvons citer parmi ces raisons : l'énorme croissance de la capacité de silicium, les conceptions récentes ont tendance à utiliser des accélérateurs et des systèmes sur puce (SoCs) hétérogènes, réduisant la contrainte de time-to-market, améliorant la productivité de la conception en réutilisant les conceptions comportementales au lieu des conceptions RTL (Register-Transfer-Level), en séparant un algorithme de l'architecture pour permettre une vaste exploration des alternatives d'implémentation. Vivado HLS [4] fourni par Xilinx [2] est un exemple d'outil permettant de créer un accélérateur matériel à partir d'un langage de programmation de haut niveau (C, C ++ ou SystemC). Par conséquent, HLS réduit les efforts de capture et de débogage de la conception HDL tout en permettant une flexibilité dans l'implémentation matérielle finale afin de répondre aux contraintes de conception.

1.1 Objectifs et contributions

Dans ce contexte s'inscrit ce sujet de thèse qui s'intéresse essentiellement à la proposition d'une méthodologie de conception conjointe logicielle/matérielle (SW/HW) [5, 6] de haut niveau afin d'implémenter les différents algorithmes de tri sur une plateforme Xilinx à base de SoC. En fait, ce travail entre dans le cadre d'une cotutelle entre le Laboratoire

2. https ://www.xilinx.com

de recherche en développement et contrôle des applications Distribués (ReDCAD) en Tunisie et le Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH, UMR CNRS 8201) en France.

Le premier et le principal objectif de cette thèse consiste à proposer une implémentation hardware efficace des algorithmes de tri (BubbleSort, InsertionSort, SelectionSort, QuickSort, HeapSort, MergeSort, ShellSort, TimSort) en utilisant plusieurs étapes d'optimisation de HLS dans la partie logicielle et hardware en intégrant plusieurs permutations comme entrée qui sont générées à l'aide de la méthode de Lehmer. Cet objectif considére des données codées sur 32 bits et 64 bits.

Le deuxième objectif met l'accent sur l'étude des performances de la méthode de conception haut niveau en terme de surface FPGA, temps d'exécution et écart type afin d'estimer le meilleur choix de l'algorithme pour l'utilisation dans le domaine avionique.

1.2 Plan de thèse

Ce sujet de thèse est résumé en 3 grands chapitres qui sont détaillés comme suit :

Dans le premier chapitre, nous présentons le contexte de nos travaux qui aborde sur les applications avioniques (planification de plans de vol, aide à la décision multicritère et les algorithmes de tri) dans les systèmes de transport intelligents. Nous commençons tout d'abord par une étude de l'état de l'art sur l'optimisation et l'implémentation des différents algorithmes sur la plateforme FPGA. Ensuite, nous détaillons de même les nouveautés apportées aux algorithmes de tri ainsi que les degrés de complexité de ces différents algorithmes. Nous concluons ce chapitre par une étude de complexité et de stabilité sur CPU afin de choisir le meilleur algorithme pour l'utiliser dans le domaine avionique.

Dans le deuxième chapitre, nous présentons tout d'abord la plateforme Zynq ZC 702. Ensuite, nous détaillons l'implémentation logicielle des algorithmes de tri sur le processeur ARM Cortex A9. Une étude de performance des algorithmes de tri est réalisée en utilisant le mode *Standalone*[3] de l'environnement de développement en termes de temps d'exécution et d'écart type.

Finalement, le troisième chapitre met l'accent sur l'utilisation de l'outil de synthèse haut niveau HLS. Dans ce contexte, nous détaillons la synthèse de haut niveau des algorithmes de tri. Par la suite, nous calculons les mesures de performance nécessaires en termes de temps d'exécution et de l'écart type aussi les ressources matérielles utilisées pour ces différents algorithmes en utilisant une méthode de conception SW/HW. Nous concluons ce chapitre par une étude comparative entre les différentes implémentations matérielles efficaces des algorithmes de tri en utilisant la méthodologie de conception haut niveau pour choisir le meilleur algorithme.

3. Pas de système d'exploitation

Chapitre 2

Étude générale sur l'utilisation des algorithmes de tri dans les systèmes de transports intelligents

Sommaire

2.1 Introduction 6
2.2 Systèmes de transport intelligents (STI) 6
2.2.1 Historique des STI 6
2.2.2 Contexte des STI 7
2.2.3 Systèmes avioniques 7
2.3 État de l'art 14
2.4 Vue globale sur les plateformes d'exécution et les algorithmes de tri 19
2.4.1 Plateformes d'exécution 19
2.4.2 Algorithmes de tri 21
2.5 Étude de la complexité des algorithmes de tri 28
2.5.1 Permutation 29
2.5.2 Méthode de Lehmer 30
2.5.3 Temps d'exécution et écart type 31
2.5.4 Dispersion statistique 42
2.6 Conclusion 43

2.1 Introduction

De nos jours, les systèmes de transports intelligents [7, 8, 9, 10] jouent un rôle significatif dans la réduction des risques d'accidents, embouteillages et la pollution de l'air. Ils sont employés dans différents domaines d'applications. Nous citons par exemple avionique, ferroviaire et automobile. Dans chaque système, plusieurs applications utilisent les algorithmes de tri dans différentes étapes de programmation comme les systèmes d'aide à la décision, planification de plan de vol et l'ordonnancement des tâches, etc. La performance des algorithmes de tri n'est pas seulement affectée par leurs complexités, mais aussi par les plateformes d'exécution ciblées. Un ensemble de plateformes différentes peut être utilisé tel que CPU (single ou multi cœur), GPU [11], FPGA ou des architectures hétérogènes [12].

Dans ce chapitre, nous nous intéressons à étudier les systèmes de transport intelligents. Nous commençons tout d'abord par citer quelques exemples de ces systèmes : planification de plan de vol, systèmes d'aide à la décision, etc. Ensuite, nous achevons ce chapitre par une description des solutions proposées au niveau de l'état de l'art pour l'implémentation des algorithmes de tri sur une plateforme embarquée. Par la suite, nous consacrons essentiellement cette partie pour l'étude des algorithmes de tri sur différentes plateformes d'utilisation. Finalement, nous détaillons les résultats de la validation des algorithmes de tri sur un CPU.

2.2 Systèmes de transport intelligents (STI)

Dans cette section, nous présenterons dans un premier lieu un aperçu général sur les systèmes de transports intelligents(STI). Par la suite, nous détaillons les différents algorithmes de tri qui sont utilisés dans les fonctions d'aide à la décision multicritère aéronautique et dans la planification de plan de vol pour piloter des aéronefs dans des situations d'urgence avec des contraintes de temps réel pour rendre la situation actuelle du vol sécurisée [13].

2.2.1 Historique des STI

À l'aide de l'évolution technologique, durant les années 60-70, les chercheurs ont commencé le développement des STI dans le domaine informatique. Les systèmes d'aide à la conduite est l'objectif commun des projets lancés au cours de ces années. Pour cela, les systèmes interactifs embarqués sont nécessaires pour cette technologie. Nous citons à titre d'exemple quelques projets qui ont participé au développement des STIs :

- Projet ALI (Autofahrer-Leit-und-Information System) en Allemagne [14] :Ce projet a lancé dans les années 1970. Le système permet aux autorités d'obtenir des données sur les véhicules sur les routes et même de communiquer avec eux.
- Projet CACS (Comprehensive Automobile trafic Control System) au japon [15, 7] : Ce projet a pour objectif de développer un système et un moyen efficaces pour contrôler de manière complète le trafic automobile de la ville. Techniquement, ce projet vise à développer un système complet de traitement de l'information pour le trafic automobile en combinant un dispositif électronique, une communication sol-sol, un contrôle informatique et des technologies de réseau. Ce système est utilisé en coordination avec

Ces deux projets sont des systèmes de guidage dynamique basés sur des conditions de trafic réelles. Mais ils se sont heurtés à un problème majeur : les infrastructures informatiques de l'époque n'étaient pas assez performantes pour fournir un service convenable et fluide. Avec l'évolution des technologies des microprocesseurs, une nouvelle génération de projets a été proposée dans les années 80. Nous citons par exemple projet PROMETHEUS (Programme pour un trafic européen avec une efficacité maximale en 1986) et IVHS (America - Société de transport routier intelligent en Amérique en 1988).ces deux programmes permettent une optimisation de la sécurité et une diminution de la pollution, etc.

2.2.2 Contexte des STI

Les STIs comprennent une vaste gamme de technologies et applications y compris l'électronique, le traitement de l'information, le trafic [16, 17], les communications vers les systèmes classiques de transport et les commandes. Ces systèmes permettent une amélioration significative des performances des systèmes de transport et du trafic en termes de la sûreté et de la sécurité de transport [18, 19], de l'efficacité et de la commodité du réseau de transport de surface globale. Depuis, les technologies des ordinateurs, des communications et des capteurs se sont considérablement améliorées, les technologies STIs sont apparues dans les secteurs des autoroutes et des transports en commun du monde entier. Les avantages directs du déploiement des STIs peuvent être analysés en fonction de différents ensembles de facteurs, appelés catégories de bénéfices des STIs. Dans la littérature, les avantages des STIs sont classés dans les catégories suivantes :
– Sécurité,
– Efficacité de flux,
– Productivité et réduction des coûts,
– Avantages pour l'environnement.
Ces systèmes sont employés dans différents domaines d'applications. Nous citons par exemple les systèmes avioniques, ferroviaires et automobiles.

2.2.3 Systèmes avioniques

Les systèmes avioniques sont constitués d'un ensemble d'équipements, composés eux-mêmes de matériel et de logiciel. Plus précisément, l'avionique est l'ensemble des capteurs, actionneurs, calculateurs, logiciels et bus réseaux, qui interagit ensemble pour accomplir différentes fonctionnalités d'un avion [20, 21, 22]. Ces fonctionnalités sont soumises à des contraintes temps réel et de criticité tout en maintenant des normes élevées de sécurité et de fiabilité. La communication entre ces équipements est assuré par des interfaces physiques/logiques. Pour répondre aux besoins de l'évolution des systèmes avioniques, il y a essentiellement plusieurs architectures utilisées [23]
– Architecture centralisé : Les équipements passent uniquement par un seul calculateur pour exécuter leurs fonctions. Le dysfonctionnement ou la perte de ce calculateur peut entrainer des conséquences critiques dans l'exécution de certaines tâches. Pour résoudre ce problème, l'architecture fédérée est proposée comme solution.
– Architecture fédérée : les équipements embarqués sont répartis dans l'avion. Pour cela, chaque fonction avionique possède ses propres ressources matérielles et sa plateforme d'exécution. Cette architecture permet de réduire la longueur du câblage entre les équipement et le poids de l'avion. Dans cette architecture, le contrôle des équipements est

centralisé autour de plusieurs calculateurs plus puissants que celui de l'architecture centralisée.
– Architecture répartie : La seule différence entre cette architecture et l'architecture fédérée est qu'il n'existe plus de contrôle centralisé des communications. Les équipements interagissent ensemble pour réaliser leurs missions. Un avantage de cette architecture est la facilitée de maintenance.
– Avionique Modulaire Intégrée (Integrated Modular Avionics <IMA>) [24, 25] : basée sur un environnement partitionné où plusieurs fonctions avioniques sont exécutées par une plateforme d'exécution partagée. Elle est la plus récente architecture utilisée dans les avions pour répondre aux besoins de l'avionique.

Les systèmes avioniques sont organisés en sous-systèmes pour des domaines fonctionnels tels que le contrôle de vol, le contrôle moteur, la navigation, la communication, etc. Les technologies avioniques offrent aux pilotes une flexibilité, une assistance et une précision tout en garantissant un niveau de sécurité optimal. Elles s'appliquent à plusieurs utilisations :
– Avions commerciaux : information de bord, aide à la maintenance, assistance à l'atterrissage, régulation des moteurs ;
– Hélicoptères civils : contrôle de vol, chaîne de pilotage, pilotage automatique, observation ;
– Avions commerciaux et hélicoptères civils : navigation inertielle, traitement et transmission de données, commandes de vol, pilotage automatique, enregistreurs de données de vol, système de suivi de situation avion.

Néanmoins, les systèmes avioniques sont généralement des systèmes réactifs durs en temps réel avec différents niveaux de criticité. Les caractéristiques des systèmes avioniques sont les suivantes :
– Systèmes liés à la sécurité ;
– Systèmes avioniques temps réel ;
– Systèmes réactifs ;
– Tolérance aux pannes, évitement des pannes, élimination des défauts, détection des pannes [26].

Comme nous l'avons vu en discutant des caractéristiques des systèmes critiques en temps réel durs (c'est-à-dire de la plupart des systèmes avioniques), l'architecture du système a un impact majeur sur :
– Les capacités de tolérance en temps réel et de panne,
– L'applicabilité des techniques d'évitement des défauts et d'élimination des défauts

Planification de plan de vol

La planification de trajectoires [27, 28] optimales est un domaine de recherche très riche et dynamique dans de nombreux domaines d'application tels que la robotique [29], l'espace ou l'aviation, l'intelligence artificielle ou la théorie de contrôle [28]. Selon les besoins, les problèmes sont de nature différente et les techniques utilisées pour les résoudre sont différentes. Ici, nous nous intéressons à trouver le chemin optimal le plus court dans un espace bidimensionnel [30] et tridimensionnel [13]. Plusieurs méthodes, comme l'algorithme de Dijkstra [31, 32] ou l'algorithme A* [33], discrétisent le domaine et travaillent sur le graphe généré pour trouver le chemin optimal. Pour les algorithmes Dijikstra et A* [34], la première étape essentielle est de modéliser l'environnent avant de chercher le chemin optimal tandis que, pour certains autres algorithmes tels que Potential Fields [35] et RRTs

(Rappidly Exploring Random Trees) [36], la modélisation de l'environnement n'est pas nécessaire. Dix ans après l'apparition de l'algorithme Dijkstra, Hart et Nilson ont publié en 1969 une amélioration importante. Il s'agit de l'algorithme A* qui est utilisé pour de nombreuses applications et demeure l'un des algorithmes de planification de chemin les plus connus, applicable à l'espace de configuration métrique ou topologique [37]. Cet algorithme utilise une combinaison de recherche heuristique telle que BFS (Best First Search) et de recherche basée sur le plus court chemin tel que l'algorithme de Dijkstra. Cette heuristique est utilisée pour chaque nœud pour trouver le meilleur nœd qu'il faille visité. Il est optimiste et déterministe grâce à l'utilisation de l'heuristique. Dans le domaine des algorithmes de recherche heuristique, l'algorithme A* est largement appliqué. Il permet de calculer le chemin le plus court dans un graphe entre un point de départ et un point d'arrivée donnés. Il utilise une fonction d'évaluation pour trier les nœds [38]. La sélection de la fonction d'évaluation est le problème central de l'algorithme A*, elle détermine le succès de l'algorithme. La fonction d'évaluation de l'algorithme A* est définie comme suit :

$$f(n) = g(n) + h(n) \tag{2.1}$$

où $g(n)$ est le coût de chemin minimum entre le nœud de départ et le nœud actuel n via la séquence de cellules sélectionnées (*i.e.* chemin déjà parcouru), $h(n)$ est le coût le moins élevé entre le nœud actuel n et le nœud cible (*i.e.* distance heuristique minorant pour le chemin restant à parcourir entre le nœud actuel et le nœud cible). Chaque nœud adjacent du nœud réel atteint est évalué par la valeur de la fonction $f(n)$. Le nœud associé à la valeur la plus basse de $f(n)$ est choisi comme le suivant dans la séquence. Les avantages de cet algorithme est que les distances utilisées comme critère peuvent être adoptées, modifiées ou qu'une autre distance peut être ajoutée. l'algorithme 2.2.3 est un algorithme simple à utiliser et ne nécessite pas de pré-traitement, aussi ne consomme que peu de mémoire. Aussi, il est relativement rapide. A* utilise deux listes différentes qui contiennent des nœuds d'une grille ou d'un graphe :

- la liste ouverte contient les nœuds à étudier.
- La seconde liste, appelée liste fermée, contient tous les n
  oeuds qui, à un moment où à un autre, ont été considérés comme faisant partie du chemin solution. Avant de passer dans la liste fermée, un n
  oeud doit d'abord passer dans la liste ouverte, en effet, il doit d'abord être étudié avant d'être inséré dans le chemin minimum ou abandonné.

Nous expliquons l'algorithme 2.2.3 par un exemple concret sur base d'un graphe G de 9 nœuds (voir Figure 2.1). Pour chaque nœud, on calcule les valeurs suivantes de l'heuristique h : $h(a) = 5$, $h(b) = 4$, $h(c) = 4$, $h(d) = 3$, $h(e) = 7$, $h(f) = 4$ et $h(g) = 2$. Le but de cet exemple est d'appliquer l'algorithme A* entre le point S0 et le point D pour trouver le plus court chemin comme montre la Figure 2.1. À partir de ce graphe, nous calculons tout d'abord la valeur des fonctions $g(n)$ et $h(n)$ à chaque sommet. Par la suite, nous trouvons le chemin le plus court entre S0 et D.

Aide à la décision multicritère

L'aide à la décision multicritère [39, 40, 41] est une branche bien connue dans le domaine de la prise de décision. C'est une branche d'étude majeur de la recherche opérationnelle qui aide le décideur à résoudre les problèmes lorsque plusieurs critères contradictoires sont impliqués et doivent être évalués. Cette branche de modèles est très souvent appelée

Algorithm 1 Algorithme A*

$S0$: Nœud de depart
D : Nœud de destination
$h(x)$: heuristique estimant le coût de x vers D
$l(x, y)$:poids entre x et y ;$P(x)$: Parent
F : Ensemble des nœuds étudiés
O : Ensemble des nœuds voisins
$g(x)$: Fonction de coût de trajet de S0 à x ;
$f(x)$: Fonction de coût total de S vers D en passant par x
Initialisation : $O < -s0$; $F < -$; $g(s) < -0$; $f(s) < -h(s)$;
while $O \neq \varnothing$ **do**
  Supprimer x de O
  Ajouter x dans F ;
  if $xeqD$ **then**
    Solution trouvée
  else
    for y successeur de x **do**
      if $y \notin F \cup Oorg(y) > g(x) + l(x, y)$ **then**
        $g(y) < -g(x) + l(x, y)$
        $f(y) < -g(y) + h(y)$
        $p(y) < -x$
        Ajouter y dans O ;
      end if
    end for
  end if
end while

MCDM. Elle est divisée en deux catégories :

– MODM : Multiple objective decision making : prise de décision multi-objectif. MODM est généralement utilisé pour la conception ou la planification, avec le nombre d'alternatives est infini (l'espace de décision est continu). Le but de MODM est de concevoir l'alternative optimale en tenant compte des diverses interactions au sein des contraintes données. En atteignant des niveaux satisfaisants de plusieurs objectifs, la meilleure alternative est obtenue.
– MADM : Multiple attribute decision making : MADM est principalement utilisé pour les problèmes de sélection et il est toujours associé à un nombre limité d'alternatives et à un classement de préférence (l'espace de décision est discret). La proposition d'un nombre fini d'alternatives est évalué en respectant les différents attributs pondérés et le classement de préférence obtenu qui décrit les performances de chaque alternative. [42, 43]

Elle permet de choisir la meilleure solution ou la solution optimale parmi tout un ensemble de solution. Il existe différentes classifications des problèmes et méthodes de MCDM. Une distinction majeure entre les problèmes de MCDM repose sur le fait que les solutions sont définies explicitement ou implicitement.

– Les problèmes d'évaluation multicritère : ces problèmes consistent en un nombre limité d'alternatives, explicitement connues au début du processus de solution. Chaque alternative est représentée par sa performance selon plusieurs critères. Le problème peut être défini comme la recherche de la meilleure alternative pour un décideur (DM) ou la

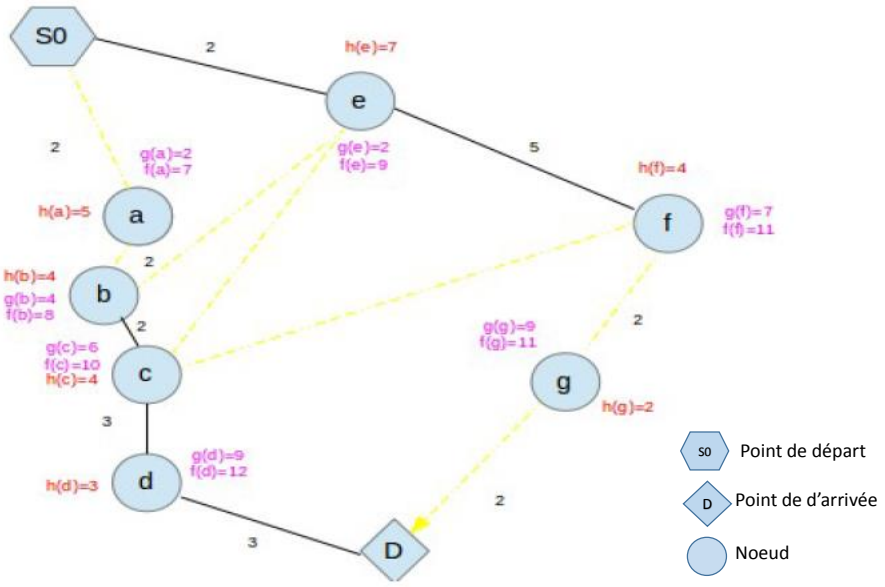


FIGURE 2.1 – Chemin le plus court en appliquant l'algorithme A*

recherche d'un ensemble de bonnes alternatives. On peut aussi être intéressé à "trier" ou "classifier" des alternatives.

– Les Problèmes de conception à critères multiples (problèmes de programmation mathématique objectifs multiples) : Dans ces problèmes, les alternatives ne sont pas explicitement connues. Une alternative (solution) peut être trouvée en résolvant un modèle mathématique. Le nombre de solutions est infini et ne peut pas être comptabilisé (lorsque certaines variables sont continues) ou typiquement très grand s'il est comptabilisé (lorsque toutes les variables sont discrètes).

Qu'elle que soit le problème, les informations sur les préférences des décideurs sont nécessaires afin de différencier les solutions. Les méthodes de résolution des problèmes de MCDM sont généralement classées en fonction de la synchronisation des informations de préférence obtenues à partir du décideur. Les problèmes de MCDM (Multiple Criteria Decision Making) peuvent également être divisés en deux classes principales en ce qui concerne la façon dont les poids des alternatives sont déterminés : Prise de décision compensatoire comme par exemple Analytical hierarchy Process (AHP) et la prise de décision surclassée comme l'ELECTRE (ELimination and Choice Expressing Reality). Le principe de fonctionnement de base de toute méthode MCDM est le même pour les différentes méthodes et il est comme suit :

– Sélection des critères
– Sélection des alternatives
– Sélection des méthodes d'agrégation
– Sélection des alternatives en fonction des poids ou en fonction de la supériorité

Il existe plusieurs méthodes dans chacune des catégories de sélection citée ci-dessus. Les méthodes basées sur la priorité, la supériorité, la distance et les méthodes mixtes sont également appliquées à divers problèmes. Chaque méthode a ses propres caractéristiques et les méthodes peuvent également être classées en méthodes déterministes, stochastiques

et floues. Les différentes méthodes sont décrites comme suit :
– Weighted sum method (WSM)
– Analytical hierarchy process (AHP)
– Preference ranking organization method for enrichment evaluation (PROMETHEE)
– The elimination and choice translating reality (ELECTRE)
– The technique for order preference by similarity to ideal solutions (TOPSIS)
– Multi-attribute utility theory (MAUT)
Les méthodes MCDM les plus couramment utilisées sont les suivantes : optimisation multi-objectifs, AHP, PROMETHEE, ELECTRE, MAUT, méthodes floues et systèmes d'aide à la décision (DSS) qui sont des techniques sophistiquées, interactives et assistées par ordinateur pour faciliter les décisions [44]. Celles-ci peuvent supporter des problèmes complexes qui seraient autrement difficiles à gérer. Les systèmes d'aide à la décision (DSS) doivent permettre aux utilisateurs (décideurs) de prendre les meilleures décisions dans les meilleurs délais. Ce sont des systèmes informatiques qui rassemblent des informations provenant de diverses sources, facilitent l'organisation et l'analyse d'informations et facilitent l'évaluation des hypothèses sous-jacentes à l'utilisation de modèles spécifiques. Les systèmes d'aide à la décision (DSS) et MDCM visent généralement à supporter la prise de décisions non structurées. Malgré le fait que ces deux domaines possèdent des objectifs communs, et malgré les développements observés au cours des quinze dernières années, il existe peu d ?exemples de systèmes opérationnels possédant à la fois le caractère fonctionnel des SAD et la richesse des MDCM [45, 46, 47]. Les systèmes d'aide à la décision nécessitent des algorithmes de tri pour faciliter la recherche de la solution optimale. Dans la section 2.2.3, nous détaillons tout d'abord les algorithmes de tri utilisés dans ce travail.

Algorithmes de tri

Un algorithme de tri est, en informatique ou en mathématiques, un algorithme qui permet d'organiser une collection d'objets selon une relation d'ordre déterminée. Les objets à trier sont des éléments d'un ensemble muni d'un ordre total. Le tri est un processus commun entre plusieurs domaines d'application et il est considéré comme l'un des problèmes les plus connus dans le monde informatique. Pour y parvenir, plusieurs algorithmes sont disponibles dans différents travaux de recherche pour résoudre ce problème [48]. La classification de ces algorithmes est une étape nécessaire et importante pour choisir l'algorithme le plus adapté au problème traité, tout en tenant compte des contraintes imposées par celui-ci. Tout d'abord, nous pouvons distinguer entre deux types d'algorithmes de tri d'application générale, procédant par comparaisons entre des paires d'éléments, et des algorithmes plus spécialisés faisant des hypothèses restrictives sur la structure des données entrées (par exemple, le tri par comptage, applicable uniquement si les données sont prises parmi un petit ensemble connu à l'avance). Divers critères permettent de déterminer quelques grandes classes d'algorithmes de tri : On distingue le tri interne, où l'ensemble des éléments à trier peut être entièrement chargé dans la mémoire de l'ordinateur, et le tri externe, où l'ensemble des éléments à trier est trop grand pour pouvoir être traité en mémoire. En effet, dans le cas de tri externe, l'ensemble des éléments à trier est divisé en des sous-ensembles tenant en mémoire. Ces sous ensembles sont stockés dans des mémoires auxiliaires. Une étape supplémentaire est alors nécessaire pour réaliser la fusion des différents sous ensembles [49]. Parmi les tris internes, on préfèrera souvent des tris sur place, qui n'utilisent qu'un petit nombre constant de variables auxiliaires, à des tris recopiant tout ou partie de la structure contenant les éléments à trier [50]. Cette

distinction est fondamentale en termes de philosophie de travail qu'en ce qui concerne les problèmes d'optimisation. En conséquence, le nombre des entrée-sorties devient un facteur prédominant dans la recherche de l'efficacité de l'algorithme, alors que l'on ne se préoccupe que de réduire le nombre de comparaisons et d'échanges d'enregistrements dans le cas de tri interne. L'efficacité de ces algorithmes est liée au type de traitement de l'information. Elle dépend aussi de temps d'exécution et l'espace-mémoire requis. Les caractéristiques principales de ces algorithmes sont :

- La stabilité : Un algorithme est dit stable s'il ne modifie pas l'ordre initial des clés identiques. Pour définir cette notion, il est nécessaire que la collection à trier soit ordonnée d'une certaine manière en entrée (ce qui est souvent implicitement le cas pour la structure de données utilisée, par exemple lorsque l'entrée est fournie sous forme de liste ou de tableau).
- Le tri sur place : Un tri est dit "sur place" s'il est effectué directement dans la structure de donnée initiale, et ne nécessite pas l'allocation d'une nouvelle structure.
- Le tri en place : Un algorithme est dit en place s'il n'utilise qu'un nombre très limité de variables et qu'il modifie directement la structure qu'il est en train de trier. Ceci nécessite l'utilisation d'une structure de donnée adaptée (un tableau par exemple). Ceci peut être un caractère très important si on ne dispose pas d'une grande quantité de mémoire utilisable.
- Le tri probabiliste : un algorithme probabiliste est un algorithme qui utilise une source de hasard. Plus précisément, le déroulement de l'algorithme fait appel à des données tirées au hasard.

Parmi les caractéristiques principales de ces algorithmes est de différencier les algorithmes de tri par :

- La complexité spatiale (notamment en termes d'espace mémoire utilisé) qui représente, quant à elle, la quantité de mémoire nécessaire pour exécuter l'algorithme. Celle-ci peut dépendre, comme le temps d'exécution, de la taille de l'entrée. Il est fréquent que les complexités spatiales en moyenne et dans le pire des cas soient identiques. C'est souvent implicitement le cas lorsqu'une complexité est donnée sans indication supplémentaire.
- La facilité de parallélisation qui permet d'exploiter les capacités multitâches de la machine.
- La complexité algorithmique est un facteur d'organisation de la typologie de ces algorithmes. Le tableau 2.1 [51] montre la complexité pour différents algorithmes de tri dans trois colonnes différentes (meilleur, moyen et pire). Nous pouvons remarquer que quatre algorithmes (QuickSort, HeapSort, MergeSort, timSort) ont une complexité en moyenne de $O(n \log(n))$. Contrairement aux autres algorithmes de tri, la performance la plus défavorable obtenue par QuickSort $O(\log(n)^2)$.
- La plateforme d'implémentation cible, tels que les processeurs CPU, les GPU, les FPGA [52, 53] et les plateformes hybrides, présente un avantage spécifique : Pour la programmabilité, le CPU est considéré comme simple. Par contre, FPGA est 10 fois plus rapide qu'un CPU.
- La capacité à être portés sur FPGA en terme de gain de temps, mais aussi de ressources utilisées/nécessaires sur le FPGA pour un algorithme donnée.

Tableau 2.1 – Complexité des algorithmes de tri

	Meilleur	Moyen	Pire
BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$
InsertionSort	$O(n)$	$O(n^2)$	$O(n^2)$
SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
HeapSort	$O(n\ \log(n))$	$O(n\ \log(n))$	$O(n\ \log(n))$
ShellSort	$O(n\ \log(n))$	$O(n\ \log(n)^2)$	$O(n\ \log(n)^2)$
QuickSort	$O(n\ \log(n))$	$O(n\ \log(n))$	$O(n^2)$
MergeSort	$O(n\ \log(n))$	$O(n\ \log(n))$	$O(n\ \log(n))$
TimSort	$O(n)$	$O(n\ \log(n))$	$O(n\ \log(n))$

2.3 État de l'art

Divers travaux sont présentés dans la littérature pour l'implémentation des algorithmes de tri sur des plateformes à base de processeur ou FPGA. Dans ce contexte, nous citons les travaux par ordre de le plus intéressant en terme de l'utilisation de la méthode de développement HLS. le travail proposé par [54] qui présente deux nouveaux algorithmes : l'algorithme quick-merge parallel (utilise un algorithme Quicksort pour trier chaque sous ensemble de données sur un cœur de CPU + algorithme MergeSort pour fusionner les résultats obtenus pour chaque cœur de CPU) et l'algorithme hybride (hybrid algorithm) (algorithme parallel bitonic sur le processeur graphique GPU +sequence Mergesort sur CPU)) en utilisant un framework openMP et CUDA. Cette approche compare les deux nouvelles implémentations avec un nombre différent de données. Le résultat obtenu montre que si le nombre d'éléments est petit, alors les algorithmes de tri fonctionnant sur GPU sont plus rapides que ceux fonctionnant sur CPU multicœur. En revanche, les algorithmes de tri exécutés sur multicœur sont plus efficaces pour un grand nombre d'éléments et que l'algorithme hybride est un peu plus lent que l'algorithme efficace quick-merge parallel sur CPU. Par la suite, l'approche présentée dans [55] propose un outil de haut niveau pour le prototypage rapide des algorithmes de Divide et Conquer parallèles sur des plateformes multicœurs. Elle implémente un modèle pour l'architecture multicœur en utilisant open MP, le framework Intel TBB (Intel Threading Building Block) et la programmation parallèle Fastflow. Leurs résultats ont montré que les algorithmes parallèles prototypes consomment moins de temps pour trouver une solution et aussi nécessitent un minimum d'effort de programmation en comparant avec la parallélisation manuelle.
Konstantinos et al. [56] ont proposé une implémentation pour trois algorithmes fondamentaux utilisés dans le domaine de la vidéo et du traitement d'image. Ils ont utilisé uniquement l'algorithme MergeSort pour calculer le cœfficient de corrélation Kendall. Différentes optimisations sont effectuées en utilisant différentes directives HLS (pipeline, unroll, array partition, inline et interface). Les résultats ont montré que l'implémentation matérielle basée sur les FPGA virtex 7 est 5.6x est meilleure que l'implémentation logicielle.
Nous avons proposé dans un travail ultérieur [57] une implémentation optimisée hardware de MergeSort et TimSort en considérant la taille de données à trier varie de 8 à 4096 et aussi en utilisant une description haut niveau et en l'exécutant sur une architecture hétérogène CPU/FPGA. Ces algorithmes sont considérés pour une intégration dans le domaine d'aide à la décision en temps réel et la planification de plan de vol pour les applications avioniques. Nous avons comparé les performances de deux algorithmes en termes

de temps d'exécution et des ressources d'utilisation de FPGA (SLices, LUT et FF). Les résultats obtenus montrent que TimSort est plus rapide que MergeSort lors de l'utilisation de l'implémentation matérielle optimisée.
Le but de l'approche présentée dans [58] est d'exploiter l'FPGA pour construire une implémentation matérielle efficace de l'algorithme MergeSort avec la conception des circuits numériques. Elle mesure l'efficacité, la fiabilité et la complexité de l'algorithme MergeSort avec les circuits numériques en utilisant seulement quatre données à trier en entrées et elle compare l'efficacité du tri MergeSort par rapport aux algorithmes Bubble sort et selection sort en termes de la complexité de circuit. L'inconvénient majeur de ce travail est l'implémentation matérielle de l'algorithme MergeSort dans FPGA [59] car il existe plusieurs autres algorithmes dans la littérature. D'une part, la méthode proposée dans [60] fournit cinq algorithmes de tri parallèles (MapSort, MergeSort, QuickSort Tsigas-Zhangs, QuickSort alternatif et STLSort) qui permettent de trier un très grand nombre de données. Les performances de plusieurs algorithmes sont basées sur deux techniques de tri différentes :
– Tri direct : Les éléments d'un tableau sont échangés directement. Certaines méthodes nécessitent un tampon intermédiaire tandis que d ?autres peuvent trier le tableau d'entrée en place.
– Tri clé / pointeur : Les données d'entrée sont un tableau d'enregistrements de 100 octets. Un tableau temporaire de paires de clés/pointeurs est créé, où la clé est un extrait de 10 octets d'un enregistrement et un pointeur pointe vers l'enregistrement d'entrée correspondant. Une méthode directe trie le tableau des paires clé/pointeur par les valeurs clés. Étant donné le tableau des paires de clés/pointeurs triés et celui de sortie, les éléments du tableau d'entrée peuvent être réordonnés en conséquence. Trois tableaux sont impliqués : celui d'entrée, celui de sortie et le tableau des paires clé/pointeur.

Cette approche utilise deux machines différentes : Nehalem (Intel Xeon 5550, 2,67 GHz) contenant 4 cœurs i7, 8 threads avec 6 Go de mémoire et Westmere (Intel Xeon 5670, 2,93 GHz) comprenant 6 cœurs i7, 12 threads avec 24 Go de Mémoire. Elle compare les différents algorithmes en tenant compte du débit, flexibilité, de l'affinité CPU qui permet de lier un processus ou plusieurs processus à un c'ur de processeur spécifique de manière à ce que les processus ne s'exécutent qu'à partir de ce noyau spécifique et de l'analyse de la microarchitecture. Les résultats de ce travail montrent que ces deux algorithmes sont recommandés si la quantité de données est de 100 000 éléments, car ils nécessitent une taille de tableau intermédiaire. Aussi, le travail présenté dans [61] propose une étude comparative des trois algorithmes de tri parallèle largement utilisé : odd even sort, bitonic sort and rank sort en termes de temps, de vitesse et d'accélération de l'architecture CPU/GPU. Il montre une implémentation d'un algorithme de réseau min-max butterfly pour la recherche d'un minimum et d'un maximum en utilisant un nombre important de collectes de données. Les résultats obtenus montrent que l'algorithme proposé possède une meilleure performance que les trois autres algorithmes.
Le travail de [12] permettent d'accélérer les algorithmes de comparaison existants (MergeSort, Tri binaire, Tri par insertion parallèle) voir par exemple [62, 63] pour travailler à une vitesse typique d'un lien Ethernet de 1 Gbit / s en utilisant des architectures parallèles (FPGAs, processeurs multi-cœurs et GPU). Les résultats obtenus montrent que la plateforme FPGA est la plus flexible puisqu'il supporte plusieurs configurations mais la moins accessible, GPU est par la suite très puissante mais moins flexible que FPGA. Il est difficile à déboguer et nécessitant des transferts de données qui augmentent la latence, puis vient le CPU parfois trop lent malgré de multiples noyaux et plusieurs processeurs,

mais il est le plus facile à approcher.
La méthode de [64] présente une analyse détaillée du merge network avec des étages en pipeline, en termes de latence, de mémoire et de débit. De plus, elle montre que le débit peut être amélioré en remplaçant les étapes log (p) finales par des étapes de bitonic merge, et par conséquent, en utilisant le parallélisme complet des données. La conception proposée peut être utilisée pour obtenir un débit optimal, tout en maintenant des performances élevées en termes de latence et de mémoire par rapport à une conception à la pointe de la technologie pour toutes les valeurs pratiques du parallélisme des données.
Janarbek at al. [65] ont développé un framework qui produit des architecture de tri pour différentes exigences (vitesse, surface, puissance,etc). Ce framework contient dix architectures de tri hautement optimisées, compose facilement des architectures de base pour générer des architectures de tri hybrides. Il permet aux experts non spécialistes du matériel de concevoir rapidement des tri de matériel efficaces et facilite le développement de systèmes de tri hétérogènes FPGA / CPU personnalisés. Les résultats montrent que ce framework génère des architectures fonctionnant au moins aussi bien que les implémentations RTL existantes pour les tableaux de moins de 16 Ko et comparables aux implémentations RTL pour le tri des tableaux plus grands. Ce travail présente un prototype de système de bout en bout en utilisant nos architectures de tri pour les baies de grande taille (16K-130K) sur un système hétérogène CPU/FPGA.
D'autre part, une autre approche détaillée dans [66] propose une méthodologie de mappage efficace en énergie et en mémoire pour obtenir une architecture de tri parallèle en utilisant Clos network pour la communication entre les étages. L'architecture de tri est paramétrable en respectant le parallélisme des données, la taille du problème et la largeur des données. Elle montre que l'architecture de tri construite peut traiter des flux de données continus sans aucun conflit de mémoire (accès en lecture ou en écriture simultanés à plus d'un mot dans une mémoire à port unique). Elle propose le streaming permutation network (SPN) entièrement pipeline en mappant le Clos Network avec un parallélisme de données plus petit que la taille du problème d'entrée. De plus, elle développe un algorithme de permutation temps réel et sur place pour le SPN afin de traiter des flux de données continus. Cet algorithme permet d'utiliser une mémoire à un seul port plutôt qu'une mémoire à deux ports (un port de lecture et un port d'écriture) et réduit la consommation de mémoire de 50%. Elle explique que pour un parallélisme de données donné p (un diviseur de N), le SPN est programmable pour réaliser tous les modèles d'interconnexion dans bitonic sorting network, avec une surcharge logique faible. Cette approche démontre le compromis entre débit, latence et surface en utilisant deux conceptions illustratives, notamment une conception à débit élevé et une conception à faible coût. Les deux modèles sont paramétrables. L'analyse détaillée des performances montre que pour la conception à haut débit, en supposant que le parallélisme de données disponible soit p (2 ' p ' N / log(N)), le temps de latence pour le tri de la séquence de clés N est égal à O(N/p), le débit (nombre de clés triées par cycle)) est égal à O (p) et la consommation de mémoire est égale à O (N). Les résultats effectuées sur un dispositif FPGA indique que notre architecture démontre une amélioration de l'efficacité énergétique de 1,3 x 1,6 fois et une efficacité de la mémoire de 1,5 x 5,3 fois supérieure à celle des conceptions les plus récentes.
Koch et al. [59] ont proposé une implémentation d'un algorithme de tri sur FPGA hautement évolutif après une analyse minutieuse des architectures de tri existantes pour augmenter les performances sur le processeur et les GPU. De plus, ils montrent comment une configuration partielle au moment de l'exécution peut être utilisée pour économiser

près de la moitié des ressources FPGA ou pour améliorer la vitesse.
Purnomo et al. [67] ont présenté une implémentation matérielle efficace de l'algorithme Bubblesort. La mise en œuvre a été effectuée en série et en parallèle. Ils ont comparé le tri à bulle en série qui compare séquentiellement deux nombres de gauche à droite et le tri à bulle en parallèle qui crée un échange parallèle en termes de mémoire, de temps d'exécution et les ressources utilisées Slices et LUT. Les résultats expérimentaux montrent que le tri à bulles en série utilise moins de mémoire et de ressources que le tri à bulles en parallèle. En revanche, le tri à bulles en parallèle est plus rapide que le tri à bulles série en utilisant la plateforme FPGA.
D'autres recherches sur la partie parallèlisation à grande vitesse pour le tri des données sur FPGA sont présentées dans [68, 69]. Le tri parallèle réalisé par Sogabe et al. [70] et Martínez et al. [68]. Enfin, l'étude de comparaison de nombreux algorithmes de tri est important couvrant le tri par fusion parallèle, le tri par comptage parallèle et le tri à bulles parallèle sur FPGA [71].

Approches	Algorithmes			Platforme				Avionic application	outil haut niveau
	Noms	Complexités	Haute performance	Parallélisation	CPU	GPU	FPGA	Mixte		
Chhugani et al. [48]	MergeSort	O(n log (n))	Yes	Yes	Yes	No	No	No	No	No
Satish et al. [11]	RadixSort	O(nK)	Yes	Yes	Yes	No	No	No	No	No
	MergeSort	O(n log(n))								
Zurek et al. [54]	Parallel quick merge Hybrid algorithm	-	Yes	Yes	Yes	Yes	No	No	No	Yes
Abirami et al.[72]	MergeSort	O (n log (n))	Yes	Yes	No	No	Yes	No	No	No
	MapSort	O(n/P(log(n))								
	MergeSort	O(n log(n))								
Pasetto et al. [60]	Tsigas-Zhangs	O(n log(n))	Yes	Yes	Yes	No	No	No	No	No
	Alternative QuickSort	O(n log(n))								
	STLSort	-								
Danelutto et al. [55]	Divide and conquer	O (n log(n))	Yes	Yes	Yes	No	No	No	No	Yes
Jan et al. [61]	min-max butterfly	-	Yes	Yes	Yes	Yes	No	No	No	No
Konstantinos et al. [56]	MergeSort	O(n log(n))	Yes	Yes	No	No	No	Yes	No	Yes
Chen et al. [66]	BitonicSort	O(n log(n)\textasciicircum{}2)	Yes	Yes	No	No	Yes	No	No	No
Koch et al. [59]	Highly scalable sorter	-	Yes	Yes	Yes	Yes	No	No	No	No
Purnomo et al. [67]	BubbleSort	O(n\textasciicircum{}2)	Yes	Yes	No	No	Yes	No	No	No
	MergeSort	O(n log(n))								
[12][73][70][68]	Parallel CountingSort	O(n+k)	Yes	Yes	No	No	Yes	No	No	No
	Parallel BubbleSort	O(n\textasciicircum{}2)								
Previous work [57]	MergeSort	O(n log(n))	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
	TimSort	O(n log(n))								
Our work	BubbleSort	O(n\textasciicircum{}2)								
	InsertionSort	O(n\textasciicircum{}2)								
	SelectionSort	O(n\textasciicircum{}2)								
	QuickSort	O(n log(n))	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
	HeapSort	O(n log(n))								
	MergeSort	O(n log(n))								
	ShellSort	O(n(log(n)\textasciicircum{}2))								
	TimSort	O(n log(n))								

2.4 Vue globale sur les plateformes d'exécution et les algorithmes de tri

Dans cette section, nous avons montrons une étude sur les différentes plateformes existantes dans la littérature. Nous citons les processeurs CPU, Les cartes graphiques GPU et la carte électronique sur puce FPGA. Ensuite, nous présentons une vue globale sur les algorithmes de tri (BubbleSort, InsertionSort, SelectionSort, ShellSort, HeapSort, QuickSort, MergeSort et TimSort).

2.4.1 Plateformes d'exécution

Une plateforme est composée de plusieurs composants spéciaux qui déterminent les performances globales. Les paradigmes d'échelle verticale les plus répandus sont les clusters de calcul haute performance (HPC), les processeurs multicœurs CPU, les unités de traitement graphiques (GPU) et les cartes électroniques sur site (FPGA). Nous décrivons chacune de ces plates-formes et leurs capacités dans les sections suivantes.

Clusters High Performance Computing (HPC)

Les systèmes clusters HPC [74], également appelés super ordinateurs qui sont des machines dotées de milliers de cœurs. Ils peuvent avoir une variété différente d'organisation de disque, de cache, de mécanisme de communication, etc. en fonction des besoins de l'utilisateur. Ces systèmes utilisent un matériel puissant, bien construit et optimisé pour la vitesse et le débit. En raison de la qualité du matériel haut de gamme, la tolérance aux pannes de ces systèmes ne pose aucun problème, car les défaillances matérielles sont extrêmement rares.

CPU

L'histoire de l'unité centrale de traitement (CPU) est courte, mais elle a révolutionné presque tous les aspects de nos vies. Aujourd'hui, l'utilisation des processeurs est devenue une étape primordiale dans notre vie quotidienne allant des ordinateurs de bureau aux téléphones portables. Actuellement, nous remarquons que la plupart des personnes ne restent pas plus de quelques heures sans interagir d'une manière ou d'une autre avec un processeur. Malgré sa popularité indiscutable, la plupart ne savent pas comment toute cette histoire a commencé. De nos jours, il est courant de voir des processeurs de fréquence supérieure à 1 GHz sur des appareils plus petits que nos téléphones portables. En un peu plus de 40 ans, nous sommes passés de 740 kHz au niveau du GHz (augmentation de plus de 1300%) et nous avons augmenté le nombre de transistors sur puce de 2 300 à plus d'un milliard (augmentation de 434 000%). Nous produisons maintenant des processeurs avec plusieurs cœurs sur la même puce, capables de prendre en charge une fonctionnalité de plus en plus importante connue sous le nom de calcul parallèle.

Graphics Processing Unit (GPU)

Les GPU ont évolué au point où de nombreuses applications du monde réel sont facilement implémentées sur ces systèmes et fonctionnent beaucoup plus rapidement que sur des systèmes multi-cœurs. Tout comme les ordinateurs en général, les GPU ont progressé rapidement au cours des 30 dernières années depuis leur introduction sur le marché. Comme les GPU ont progressé au fil des années, leurs fonctions principales sont restées les mêmes telles que l'accélération et le traitement des images. L'introduction des unités graphiques est arrivée au début des années 1980 où Intel et IBM ont mis sur le marché des produits spécialisés. D'autres sociétés telles que Commodore et Texas Instruments ont également ajouté des capacités graphiques simples sur puce ou en utilisant une carte externe. Ces cartes avaient une fonctionnalité simpliste et étaient relativement chères.

FPGA

Freeman, co-fondateur de la société Xilinx, a inventé Field Programmable Gate Arrays (FPGA) en 1986 alors qu'il travaillait pour la société Zilog. Les premiers FPGA lancés sur le marché ne disposaient pas de plusieurs milliers de portes logiques et présentaient plusieurs désavantages pour leurs homologues se sont les ASIC. Ils étaient plus lents, consommaient plus de puissance et avaient une fonctionnalité limitée. L'industrie des FPGA a progressé lentement au cours des années 1990. En 1992, le département américain de guerre de surface navale a terminé son projet sur les FPGA qui ont mis en place 600 000 portes logiques. Pendant ce temps, les principales applications pour les FPGA étaient la mise en réseau et les télécommunications. FPGA a commencé à entrer dans de nombreuses autres industries en raison du faible délai entre le développement et l'introduction sur le marché. Aujourd'hui, les FPGA peuvent répondre à de nombreuses applications différentes. Différentes séries et familles sont spécifiques aux applications et disposent d'une logique supplémentaire pour prendre en charge des processus plus rapides. Les FPGA ont une grande capacité pour les processus de parallélisation et de pipelining. D'une part, FPGA est largement utilisé, en particulier dans l'industrie. L'avantage d'utiliser FPGA est sa grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court [75]. En outre, il est également mentionné que FPGA a une performance considérable [72]. Souvent, il est utilisé comme périphériques aux processeurs pour exécuter des processus spécifiques qu'un processeur a du mal à gérer. D'autre part, l'utilisation de la carte FPGA permet de construire des applications complexes avec des performances élevées, qui sont réalisées en recevant un grand nombre de matrices programmables disponibles. Il permet d'implémenter des architectures massivement parallèles [7]. La croissance de la complexité des applications a conduit à des méthodologies de conception avec des niveaux d'abstraction plus élevés [76]. Par conséquent, des outils de synthèse de haut niveau (HLS) ont été développés pour améliorer la productivité des conceptions basées sur FPGA. Malgré cela, les premières générations de HLS n'ont pas répondu aux attentes des concepteurs de matériel ; certaines raisons ont encouragé les chercheurs à continuer de produire des conceptions matérielles plus performantes. Parmi ces raisons, nous citons la forte croissance de la capacité de silicon ; les conceptions récentes utilisent des systèmes sur puce et des accélérateurs hétérogènes, l'utilisation de conceptions comportementales à la place de la conception RTL (Register-Transfer-Level) permet d'améliorer la productivité des conceptions, de réduire les délais de commercialisation et de détacher l'algorithme de l'architecture pour permettre une exploration étendue

des solutions de mise en œuvre [77].

2.4.2 Algorithmes de tri

Étant donné une structure linéaire (tableau, liste, etc.) contenant des valeurs d'un type ordonné, il faut trier les éléments en ordre croissant (ou décroissant). Dans ce cas, il existe des dizaines d'algorithmes [78] répondant à ce problème en utilisant quelques principes de base auxquels nous ajoutons des variantes.

BubbleSort

L'algorithme BubbleSort [79, 70, 80] est un algorithme simple et bien connu dans le monde de calcul. En outre, c'est la première invention de l'algorithme de tri. Par conséquent, c'est un algorithme stable car il ne modifie pas l'ordre initial des clés identiques et adaptable c'est-à-dire qu'il sait tirer parti de l'ordre pré-existant dans une séquence. Cependant, il est inefficace de trier un grand nombre d'éléments car leur complexité est très importante O (n^2) en moyenne et dans le pire des cas. Le tri à bulles est divisé en quatre étapes. Tout d'abord, Le tri à bulle consiste à comparer successivement tous les éléments adjacents d'un tableau et à les échanger si le premier élément est supérieur au second. On recommence cette opération tant que tous les éléments ne sont pas triés. À chaque étape de l'algorithme l'élément maximal est déplacé à la fin de la suite. Ces étapes sont répétés jusqu'à obtenir un tableau trié (voir Figure 2.2). Une amélioration possible du tri à bulles consiste à utiliser une variable booléenne "drapeau" qui permet de stopper le tri si plus aucune permutation n'a lieu.

InsertionSort

InsertionSort [81] est un autre algorithme important, utilisé pour trier un petit nombre d'éléments comme montre la Figure 2.3. Néanmoins, il a de meilleures performances que le tri à bulles et le tri par selection. Le tri par insertion a une implémentation simple pour trier les éléments. Il est moins efficace lors du tri d'un nombre important d'éléments et beaucoup plus lent que d'autres algorithmes comme Quicksort, heapsort, et mergesort car leur complexité est très importante O (n^2) en moyenne et dans le pire des cas. De même cet algorithme est stable (conservant l'ordre d'apparition des éléments égaux) et adaptatif. Cependant, l'algorithme de tri par insertion intègre un nouveau élément dans chaque itération et le compare par rapport aux valeurs des éléments de la liste. Si la valeur de cet élément est inférieure à une valeur courante de l'élément, l'étape d'échange est obligatoire. Cette étape sont répétés jusqu'à n-1.

SelectionSort

L'algorithme de tri de sélection [82, 80] est un algorithme simple à analyser. Cet un algorithme de tri par comparaison. Par conséquent, c'est un algorithme de tri très facile à comprendre et très utile lorsqu'il s'agit d'un petit nombre d'éléments. Cependant, il est inefficace de trier un grand nombre d'éléments car leur complexité est O (n^2) dans tous les cas où n est le nombre d'éléments dans le tableau à trier. Cet algorithme est appelé

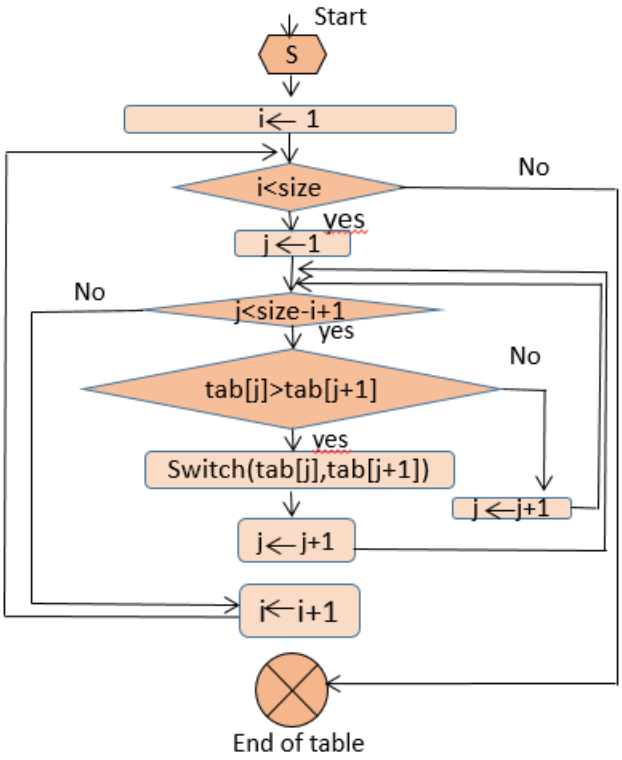


FIGURE 2.2 – Algorithme BubbleSort

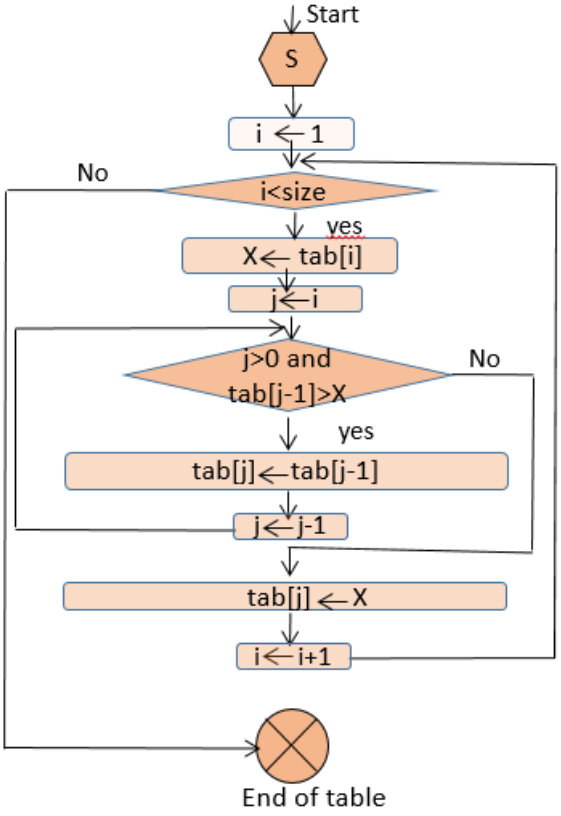


FIGURE 2.3 – Algorithme InsertionSort

tri par sélection car il fonctionne en sélectionnant un minimum d'éléments dans chaque étape de tri. Le principe est que pour classer n valeurs, il faut rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0, puis la plus petit valeur dans les valeurs restante et l'échanger avec l'élément d'indice 1 et ainsi de suite. Répéter ces étapes jusqu'à ce que vous trouviez le tableau trié comme montre la Figure 2.4. Le tri par sélection est un tri en place (les éléments sont triés directement dans la structure) mais c'est un algorithme non stable (l'ordre d'apparition des éléments égaux n'est pas préservé).

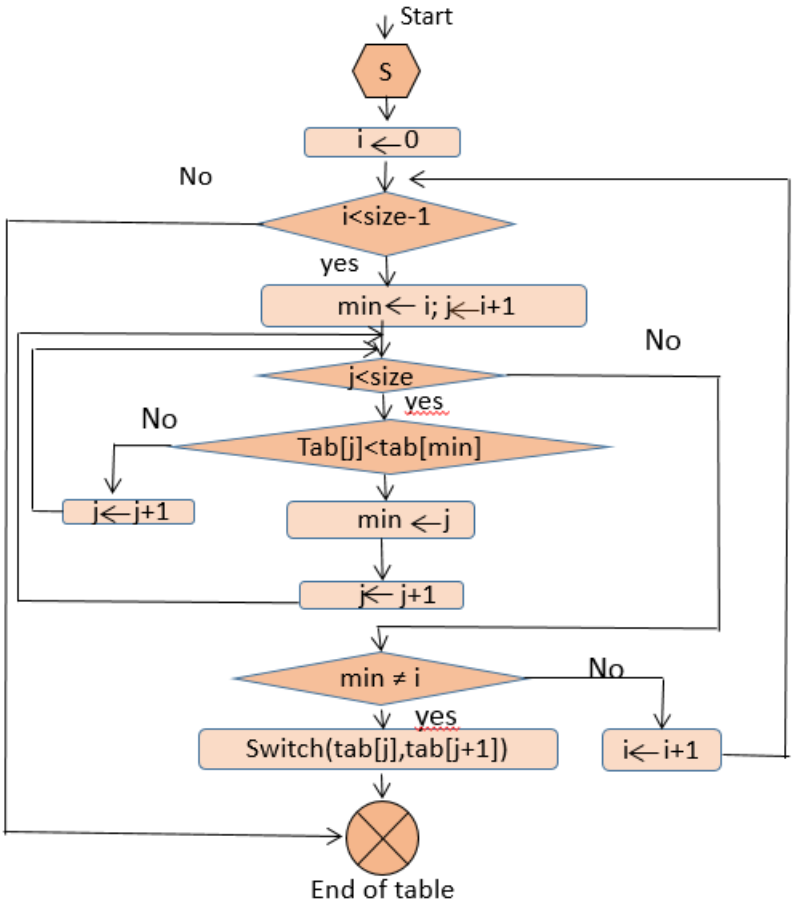


FIGURE 2.4 – Algorithme SelectionSort

HeapSort

HeapSort [83, 80] est une technique de tri basée sur la structure de données très utile qui est le tas binaire. Il est similaire au tri pas selection, car il recherche l'élément maximum dans la liste et place cet élément à la fin. Nous répétons le même processus pour l'élément restant. HeapSort est un tri de comparaison optimal, réalisant des performances $O(n \log(n))$ pour tout ordre de saisie où n est la longueur du tableau. Cet algorithme est de complexité asymptotiquement optimale, c'est-à-dire que l'on démontre qu'aucun algorithme de tri par comparaison ne peut avoir de complexité asymptotiquement meilleure. Sa complexité est proportionnelle à $O(n \log(n))$ où n est la longueur du tableau à trier. Il est relativement facile à le mettre en œuvre comme un tri sur place et non récursif [84]. Cet algorithme est le meilleur de point de vue complexité étant donné que sa complexité

est $O(n\ \log(n))$. En outre, cet algorithme est divisé en deux étapes de base comme montre la Figure 2.5 :

– Créer une structure de données Heap (Max-Heap ou Min-Heap) avec le premier élément du tas est le plus grand ou le plus petit (en fonction de Max-Heap ou Min-Heap).
– Répétez cette étape en utilisant les éléments restants pour sélectionner à nouveau le premier élément de tas et placez cet élément à la fin de la table jusqu'à obtenir un tableau trié.

Heapsort est un algorithme de tri très rapide et n'est pas stable car les opérations sur le tas peuvent modifier l'ordre relatif des éléments égaux. D'après le tableau 2.1, il a une complexité temporelle en $O(n\ \log(n))$ dans le meilleur et le pire des cas. Il est très largement utilisé pour trier un certain nombre de données.

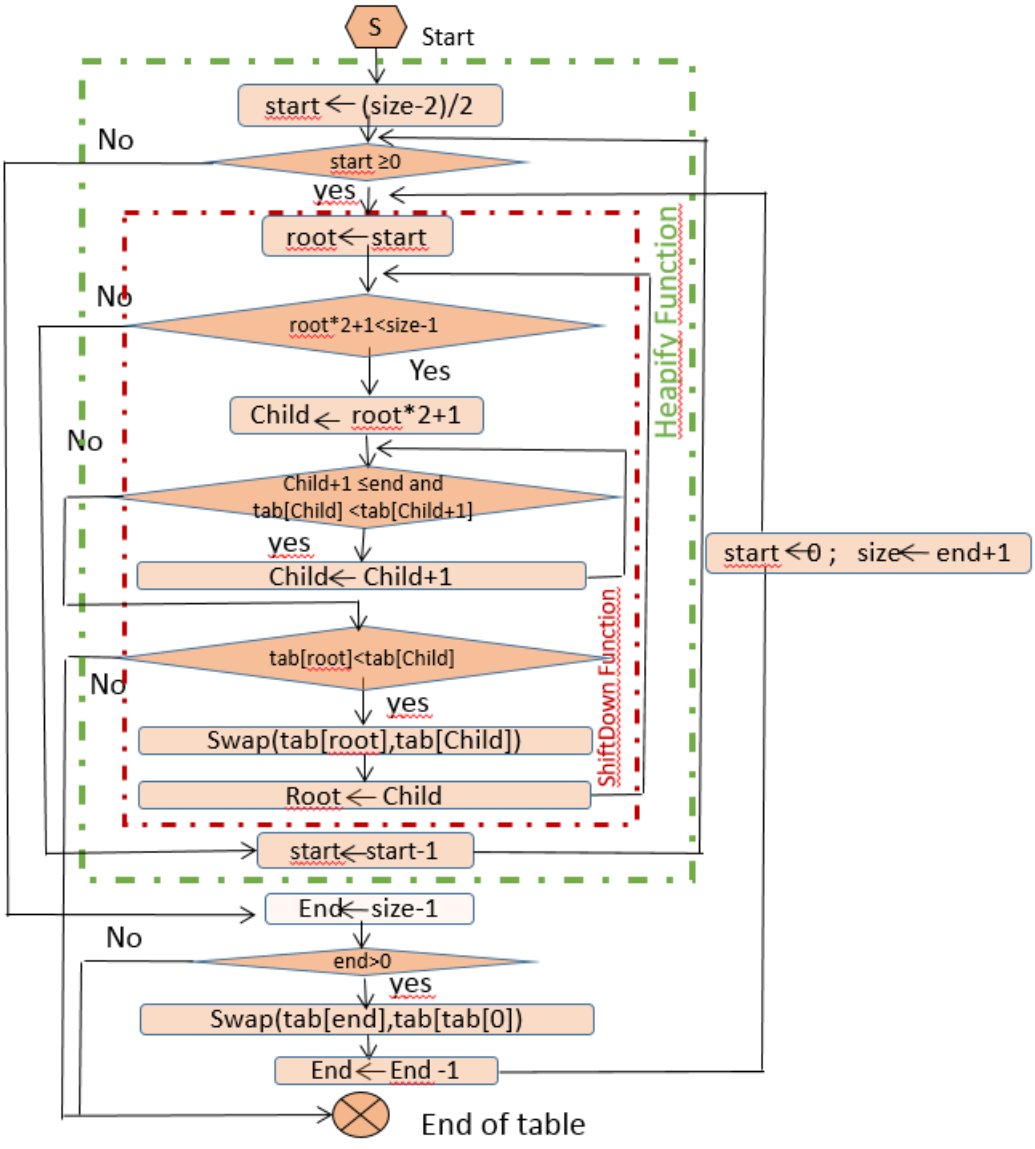


FIGURE 2.5 – Algorithme HeapSort

ShellSort

Shellsort [80] est un algorithme de tri polyvalent inventé par Shell en 1959 [85, 86]. Les résultats empiriques montrent qu'il est compétitif avec les algorithmes de tri les plus rapides, en particulier lorsque N le nombre d'éléments à trier n'est pas trop important. Il améliore l'efficacité du tri par insertion en déplaçant rapidement les valeurs vers leur destination. Il est un tri non stable en place. L'algorithme effectue plusieurs passages dans la liste, et chaque fois il trie un nombre de jeux de taille égale à l'aide du tri par insertion. Le rôle principal de cet algorithme est de calculer la valeur de h qui divise la liste en sous-listes plus petites d'intervalles égaux à h. Après cela, il trie chaque sous-liste qui contient un grand nombre de données en utilisant le tri par insertion. Enfin, cette étape est répété jusqu'à obtenir une liste triée [79]. La complexité temporelle de l'implémentation ci-dessus de shellsort est $O(n \log(n)^2)$. Il n'est pas très utilisé dans la littérature. Les étapes nécessaires pour l'algorithme shellSort sont présentées dans la Figue A.6.

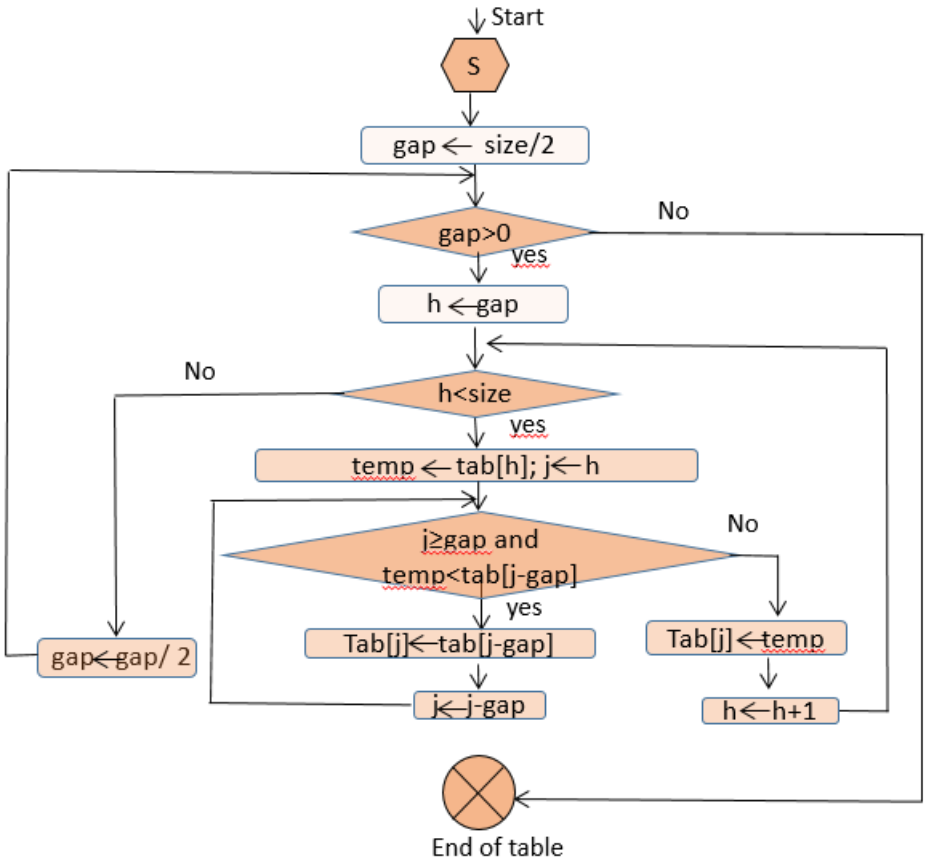


FIGURE 2.6 – Algorithme ShellSort

QuickSort

QuickSort [80] est un algorithme de division et de conquête. Il est basé sur une opération de partitionnement [87] : Tout d'abord, cet algorithme divise le tableau en deux sous-tableaux courts : les petits éléments et les grands éléments. La Figure 2.7 montre les différentes étapes suivre avec cet algorithme :
– Sélectionner un élément dans le tableau, nommé pivot.

– Partitionnement : déplacez toutes les valeurs des éléments les plus petits avant le pivot, et déplacez tous les éléments plus gros après. Cela peut être fait efficacement en temps linéaire et en temps court. Après ce partitionnement, le pivot est dans sa position finale, appelée opération de partition.
– Répéter récursivement les différentes étapes pour deux sous-tableaux avec des valeurs d'éléments plus petites et plus grandes.

Quicksort est un algorithme de tri rapide. La complexité moyenne du tri rapide pour n éléments est proportionnelle à $O(n\ \log(n))$, ce qui est optimal pour un tri par comparaison, mais la complexité dans le pire des cas est quadratique. Malgré ce désavantage théorique, c'est en pratique un des tris les plus rapides, et donc un des plus utilisés. Cependant, l'implémentation de quicksort n'est pas stable et il est un algorithme complexe, mais fait partie des algorithmes de tri rapides en pratique. Le problème le plus complexe dans le tri rapide est de sélectionner un bon élément de pivot. Dans ce cas et à chaque étape, le quicksort sélectionne la médiane comme pivot pour obtenir une complexité de $O(n\ \log(n))$ car les mauvais choix des pivots peuvent toujours conduire à des performances médiocres ($O(n^2)$).

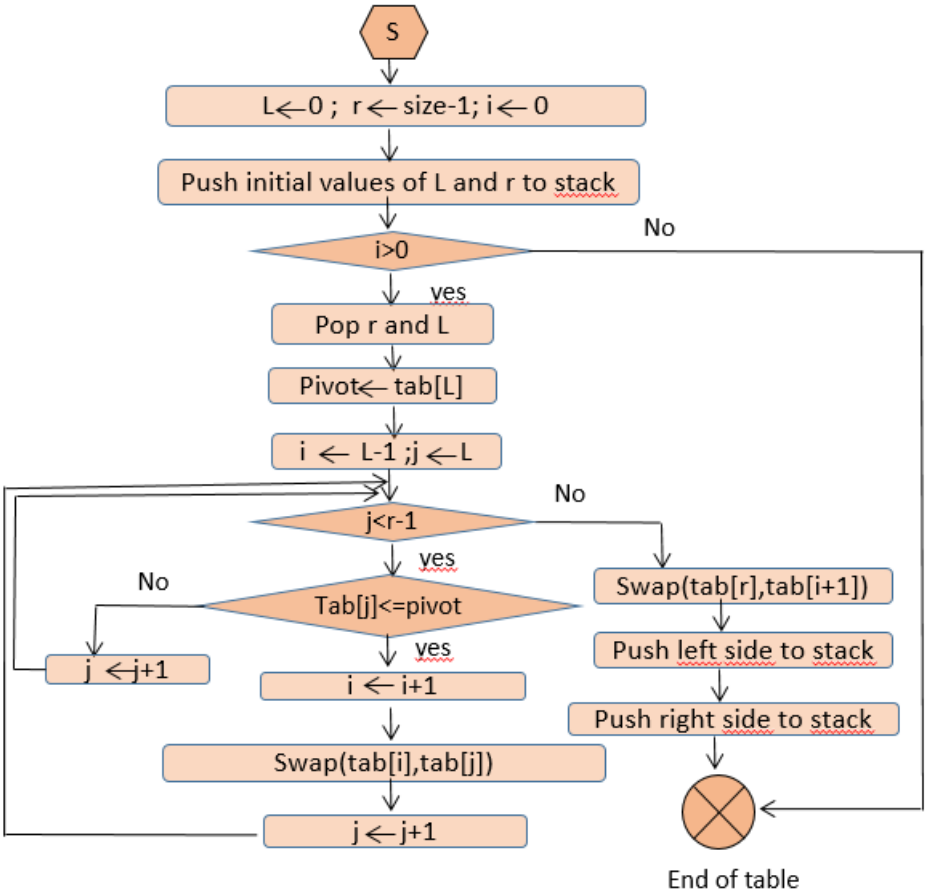


FIGURE 2.7 – Algorithme QuickSort

MergeSort

MergeSort [73, 80] a été créé pour la première fois en 1945 par John von Neumann. C'est un algorithme efficient et stable car son implémentation conserve l'ordre d'entrée

dans la sortie. MergeSort est basé sur le célèbre paradigme *diviser pour régner*. Il est un algorithme de tri par comparaison stable. Sa complexité temporelle pour une entrée de n valeurs est de l'ordre de $O(n \log(n))$, ce qui est asymptotiquement optimal. MergeSort n'est pas en place car il nécessite un espace mémoire supplémentaire pour stocker les tableaux auxiliaires. Il est plus efficace et fonctionne plus rapidement que le tri rapide si le nombre de données n est grand. Par contre, il est moins efficace et plus lent si n est petit. Les étapes nécessaires de l'algorithme de tri [88]sont présentées dans la Figure 2.8 sont comme suit

– Le tableau est divisé en deux sous-tableaux.
– Ces deux tableaux sont triés récursivement et séparément.
– Les deux tableaux triés sont fusionnés pour obtenir le résultat.

Il effectue au plus $O(n \log(n))$ opérations, qui est la complexité de temps dans le pire des cas.

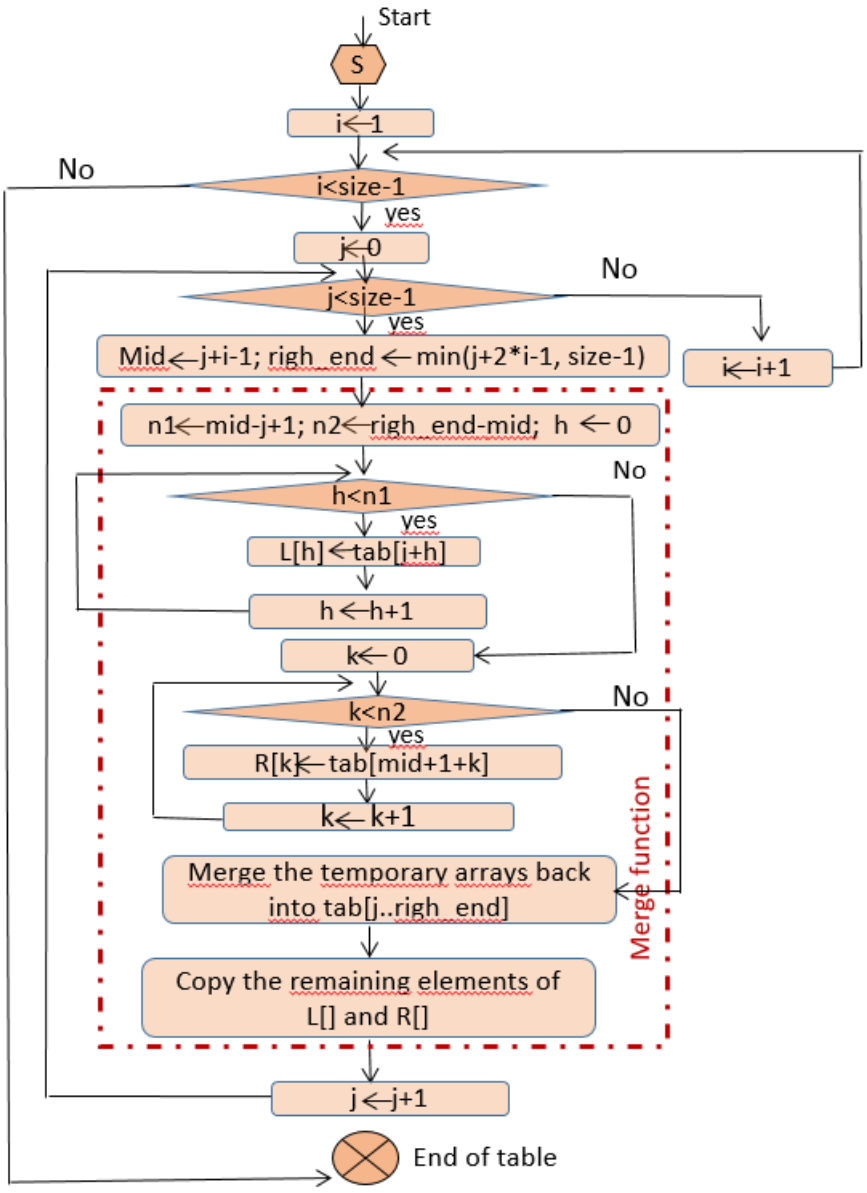


FIGURE 2.8 – Algorithme MergeSort

TimSort

TimSort [57, 89, 90] est un algorithme de tri conçu en 2002 par Tim Peters pour être utilisé dans le langage de programmation Python. Il a ensuite été implémenté dans d'autres langages de programmation tels que Java et C/C++. C'est un algorithme très bien élaboré, mais son principe de haut niveau est plutôt simple : La séquence S à trier est décomposée en séquences monotones (sous-séquences non croissantes ou non décroissantes de S), qui sont fusionnées en paires selon certaines règles spécifiques. Afin de comprendre et d'analyser la stratégie de fusion de TimSort (c'est-à-dire l'ordre dans lequel les fusions sont effectuées), nous considérons une classe entière d'algorithmes de tri qui repose sur une pile pour décider dans quel ordre les fusions sont effectuées. D'autres moyens d'ordonner les fusions ont été envisagés dans la littérature : l'algorithme classique MergeSort, le NaturalMergeSort [91] de Knuth et la stratégie de fusion optimale proposée par Barbay et Navarro [92], en utilisant les idées de Huffmann. Timsort est un algorithme de tri hybride dérivé du tri fusion et du tri par insertion, stable et conçu pour fonctionner de manière efficace sur des données réelles car leur complexité est de $O(n \log(n))$ dans la moyenne et le pire des cas. La fonctionnalité de cet algorithme est basée sur le paramètre optimal (OP) pour basculer entre deux algorithmes. La valeur de ce paramètre est fixée à 64 pour une architecture séquentielle (Processeur Intel i7). Pour l'architecture parallèle, différentes valeurs sont utilisés pour ce paramètre et nous notons que les temps d'exécution sont presque similaires. Ainsi, ce paramètre est fixé à 64 dans cette thèse car c'est la valeur normalisée utilisée par d'autres auteurs mais aussi pour permettre des comparaisons croisées. En fonction du nombre d'éléments à trier, nous pourrions suivre deux voies différentes. Si la taille du tableau est inférieure à 64 éléments, InsertionSort sera sélectionné ; sinon, MergeSort est pris en compte dans l'étape de tri comme indique la Figure 2.9.

2.5 Étude de la complexité des algorithmes de tri

Dans cette section, nous expliquons tout d'abord la notion de permutation et de la réplication. Par la suite, en se référant aux travaux de la littérature, nous cherchons à étudier la complexité des algorithmes de tri en utilisant les différentes tailles d'éléments allant de 8 à 4096. Tout au long de cette thèse, nous avons besoin au maximum de 4096 éléments à trier. Ces éléments sont considérés dans le domaine d'aide à la décision temps réel comme des solutions. La prise de décision permet de réduire le nombre d'accident pour les systèmes avioniques comme mentionner dans les travaux de thèse de ma collégue K. Nikolajevic [13] qui a comme objectif de sa thèse est de répondre à la problématique ambitieuse de la réduction des accidents de type opérationnel. Elle s'inscrivent dans le domaine des systèmes d'alarmes pour l'évitement de collision qui ne font pas une analyse approfondie des solutions d'évitement par rapport à la situation de danger. Afin de répondre au problème des situations d'urgence en vol et de rendre objective la notion de danger, les travaux de recherche présentés dans [13] mettre en confrontation la capacité d'évolution d'un aéronef dans son environnement immédiat avec une enveloppe physique devenant contraignante. Afin de mesurer ce danger, elle a conduit à construire un module de trajectoires capable d'explorer l'espace en 3D. Cela a permis de tirer des enseignements en termes de flexibilité des manœuvres d'évitement possibles à l'approche du sol. De plus l'elicitation des connaissances des pilotes et des experts d'Airbus Hélicoptères mis en situation d'urgence

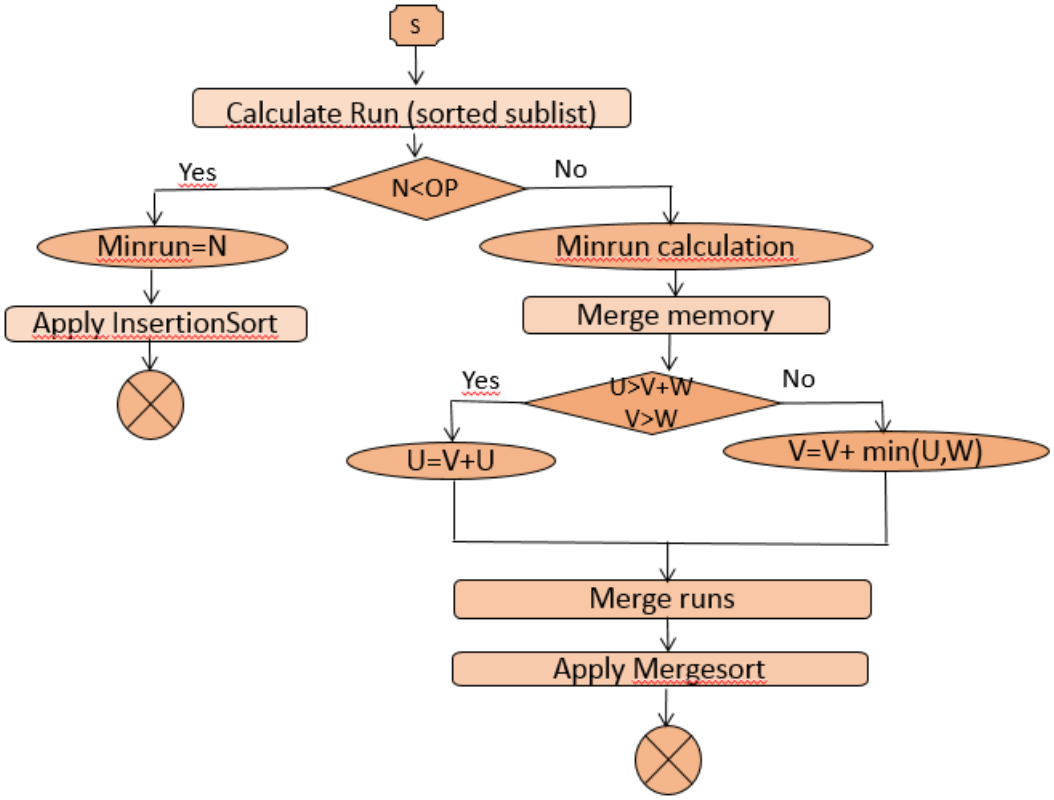


FIGURE 2.9 – Algorithme TimSort

dans le cas d'accidents reconstitués en simulation a conduit à un ensemble de paramètres pour l'utilisation de la méthode multicritère PROMETHEE II dans le processus de prise de décision relatif au choix de la meilleure trajectoire d'évitement et par conséquent à la génération d'alarmes anti-collision. Dans ce cas, elle a besoin au max de 4096 solutions pour prendre une décision temps réel. Pour cette raison, nous fixons le nombre de maximum à trier à 4096. Ensuite, nous calculons le temps d'exécution, l'écart type et le coefficient de variation des algorithmes de tri sur un environnement à base d'une machine dotée d'un système d'exploitation Linux et possédant les caractéristiques suivantes :
– Un processeur Intel I3 de fréquence 1.33 GHz.
– Une mémoire RAM de 4 Go.
Finalement, nous comparons ces résultats en termes du temps d'exécution et de stabilité en utilisant 47 permutations.

2.5.1 Permutation

Premièrement, la permutation est utilisée avec différents problèmes d'optimisation combinatoire, spécifiquement sur un domaine mathématique. Généralement, une permutation est une disposition d'un ensemble de n objets 1,2,3, ..., n dans un ordre spécifique et chaque élément se produit juste une seule fois[8]. Par exemple, il y a six permutations de l'ensemble 1,2,3, (1,2,3), (1,3,2), (2,1,3), (2,3,1) , (3,1,2) et (3,2,1). Ainsi, le nombre de permutations ne dépend que des n objets. Dans ce cas, il y a exactement n ! permutations. Deuxièmement, une permutation π est une fonction bijective d'un ensemble 1,2, ..., n à lui-même (c à d chaque élément i d'un ensemble S possède une image unique j dans S et apparaît exactement une seule fois comme valeur d'image). Nous considérons aussi $pos_{\pi}(i)$

qui est la position de l'élément i dans la permutation π, $\pi(i)$ est l'élément à la position i dans π et S_n est un ensemble de toutes les possibilités de permutation de taille n. A partir des deux permutations π_1 et π_2 une autre permutation peut être générée en multipliant les deux.

$$\pi(i) = \pi_1(\pi_2(i)) \tag{2.2}$$

$$\pi_1\pi_2 \neq \pi_2\pi_1 \tag{2.3}$$

Parmi les méthodes utilisées pour traiter le problème de la génération des permutations, nous utilisons dans ce travail la méthode de Lehmer.

2.5.2 Méthode de Lehmer

La méthode de Lehmer est une méthode spécifique pour encoder chaque permutation possible d'un ensemble de n objets. C'est un exemple de schéma de permutation de numérotation et elle est la même méthode que la table d'inversion T(π) qui est définie par un vecteur b_1 b_2 ... b_n avec b_i est le nombre d'éléments supérieur à i et apparaissent à gauche de i et 0<b_i<n-i. Par exemple si la permutation est 52143 alors le code de lehmer est 21210 qui est la table d'inversion de cette permutation puisque 5 est plus grand que 1, 2, 3 et 4, aussi 2 est supérieur à 1 et 4 est supérieur à 3. Une inversion dans une permutation π est une paire d'indices (i, j) tels que i <j et $\pi_i > \pi_j$. Par exemple si π = 52143 alors (5,2), (5,1), (5,4), (5,3), (2,1) et (4,3) sont leurs inversions. $(L(\pi))_i$ est le nombre d'inversions (i, j) avec i fixe et j variable. La permutation d'identité est la seule permutation qui n'inclut pas l'inversion. Un même code pour une table d'inversion est un code lehmer qui est lié à une séquence unique = l_1 l_2 ...l_n avec l_i est le nombre d'éléments inférieur à i et apparaissant à droite de i et 0<l_i <n-i. Par exemple si π = 52143 alors Lehmer code d= 41010.

Pour générer une permutation à partir du code lehmer, on applique l'équation suivante :

$$\pi(i) = N_i[l_i + 1] avec N_i = N - \{\pi(1), \pi(2), ...\pi(i-1)\} \tag{2.4}$$

Avec $\pi(i)$ est les éléments de permutation et i est la position de cet élément. Avec le même exemple :

$$\left.\begin{array}{l} \pi(1) = N_1[4+1] = 5 \\ \pi(2) = N_2[1+1] = 2 \\ \pi(3) = N_3[0+1] = 1 \\ \pi(4) = N_4[1+1] = 4 \\ \pi(5) = N_5[0+1] = 3 \end{array}\right\} \Rightarrow Permutation = 52143 \tag{2.5}$$

Un code de Lehmer est considéré comme un nombre factoriel, ce qui permet de calculer le nombre décimal en utilisant l'équation suivante :

$$Nombre\_dcimale = \sum_{i=0}^{n} d_i(n-i)! \tag{2.6}$$

Afin de trouver toutes les permutations, nous générons le nombre factoriel à partir du nombre décimal dans la première étape et les permutations dans la deuxième étape.

Tableau 2.2 – Temps d'exécution et écart type des algorithmes de tri sur le processeur Intel

	BubbleSort (us)	InsertionSort (us)	SelectionSort (us)	HeapSort (us)	QuickSort (us)	ShellSort (us)	MergeSort (us)	TimSort (us)
8	1.57 (2.753)	1.074 (2.173)	1.243 (2.186)	2.296 (3.631)	2.316 (2.989)	1.514 (2.486)	2.639 (3.198)	1.818 (2.911)
16	5.725 (4.786)	2.781 (3.654)	3.730 (3.941)	5.040 (4.524)	4.725 (4.413)	4.471 (4.41)	4.253 (4.102)	2.870 (3.66)
32	24.375 (7.31)	9.011 (5.671)	15.293 (9.772)	12.371 (6.075)	13.197 (6.301)	12.184 (6.123)	14.641 (6.713)	13.084 (6.4)
64	96.871 (112.167)	28.516 (9.209)	41.079 (9.895)	48.304 (23.719)	41.555 (58.629)	42.830 (69.133)	32.052 (9.919)	34.702 (10.317)
128	432.990 (36.686)	98.884 (10.277)	164.989 (42.599)	83.592 (13.835)	93.241 (13.875)	122.11 (104.016)	52.753 (7.573)	86.434 (10.721)
256	1037.802 (182.061)	403.147 (302.001)	580.504 (83.059)	296.508 (557.587)	299.465 (431.402)	329.927 (74.621)	159.046 (19.687)	192.016 (75.775)
512	3779.371 (844.048)	1394.601 (757.201)	2127.602 (284.322)	468.886 (168.523)	1136.260 (757.716)	1145.126 (389.055)	275.380 (116.047)	507.135 (782.516)
1024	12290.912 (2519.38)	4548.036 (676.104)	7249.245 (1854.144)	1504.325 (1239.939)	3291.473 (437.65)	3528.034 (1362.46)	774.106 (409.297)	1140.902 (1164.588)
2048	47304.790 (4825.091)	24829.288 (1993.282)	27876.093 (2400.166)	2093.825 (210.328)	10978.576 (2207.375)	12180.592 (585.655)	1554.525 (85.225)	1911.241 (243.134)
4096	181455.365 (9895.405)	66851.367 (3179.333)	107568.834 (6339.409)	4281.869 (756.228)	41848.195 (3462.193)	46796.158 (1431.203)	2985.149 (159.427)	4098.597 (539.352)

2.5.3 Temps d'exécution et écart type

Pour la mesure du temps d'exécution, nous avons fixé le nombre de réplications (itérations) R à 1000 pour chaque permutation. Par la suite, nous avons calculé la moyenne des temps d'exécution obtenus suite à chaque réplication afin d'obtenir des résultats similaire à l'implémentation réelle. Cependant, nous calculons le temps d'exécution et l'écart type des algorithmes de tri sur le processeur pour chaque réplication et pour chaque permutation/vecteur afin de vérifier la satisfaction de la contrainte temps réel. Les résultats de cette étude sont détaillés dans le tableau 2.2.

D'après les résultats fournis dans le tableau 2.2 et la Figure 2.10, nous constatons que le temps d'exécution des algorithmes de tri BubbleSort, InsertionSort et SelectionSort est très élevé si le nombre d'élément est supérieure à 64. Sinon, l'algorithme InsertionSort est le meilleur algorithme. De plus, la Figure 2.11 présente un zoom sur les cinq algorithmes en termes de temps d'exécution moyen. Nous concluons que MergeSort est 1.9x, 1.37x,1.38x et 1.9 plus rapide que QuickSort, HeapSort, TimSort et ShellSort respectivement en l'exécutant sur le processeur de la machine. Par la suite, nous calculons aussi les performances des algorithmes avec un autre critère de performance qui est l'écart type sur la durée d'exécution pour vérifier si les écarts présentés dans le tableau 2.2 sont statistiquement significatifs. La Figure 2.12 donne la même conclusion pour l'écart type que pour le temps d'exécution. Après, nous calculons le coefficient de variation qui est une mesure relative de la dispersion des données autour de la moyenne. Le coefficient de variation se calcule comme le ratio de l'écart-type rapporté à la moyenne, et s'exprime en pourcentage comme montre l'équation 2.7.

$$CV = 100 * \sigma/\mu \tag{2.7}$$

où σ représente l'écart type d'une permutation donnée sur R réplications et μ la moyenne des temps pour une permutation sur R réplications. Ce coefficient permet de comparer le degré de variation d'un échantillon à un autre, même si les moyennes sont différentes. D'une part, lorsque l'écart-type et la moyenne proviennent des mesures répétées 1000 fois

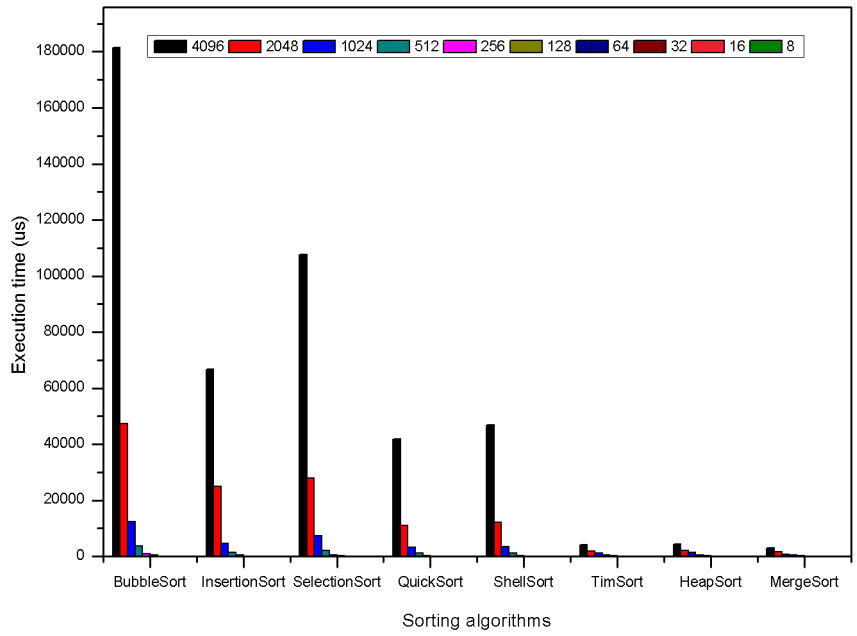


FIGURE 2.10 – Temps d'exécution sur le processeur

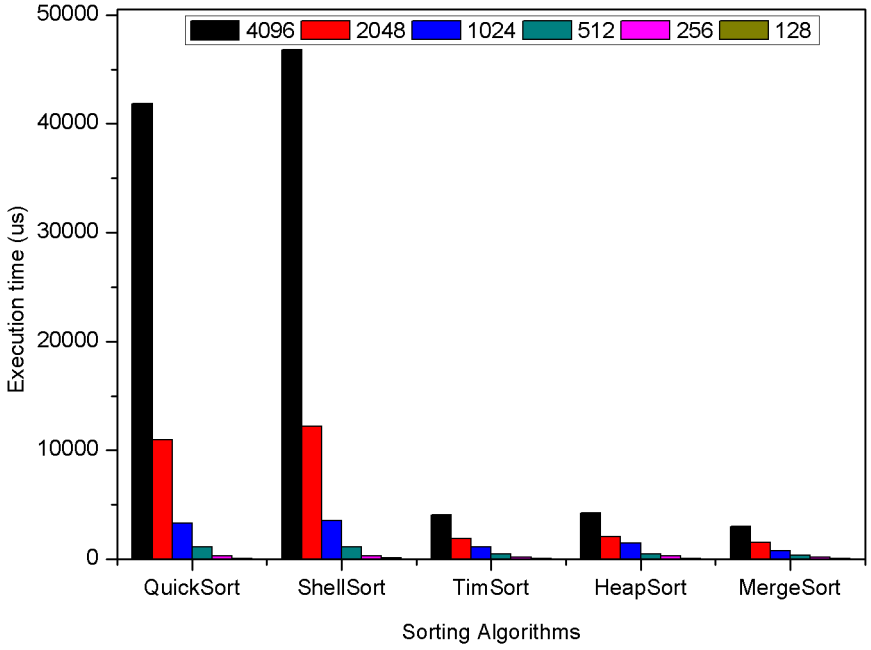


FIGURE 2.11 – Temps d'exécution des algorithmes HeapSort, QuickSort, MergeSort, ShellSort et TimSort sur le processeur

sur une même permutation, le coefficient de variation devient une mesure importante de la fiabilité. D'autre part, l'écart-type seul ne permet le plus souvent pas de juger de la

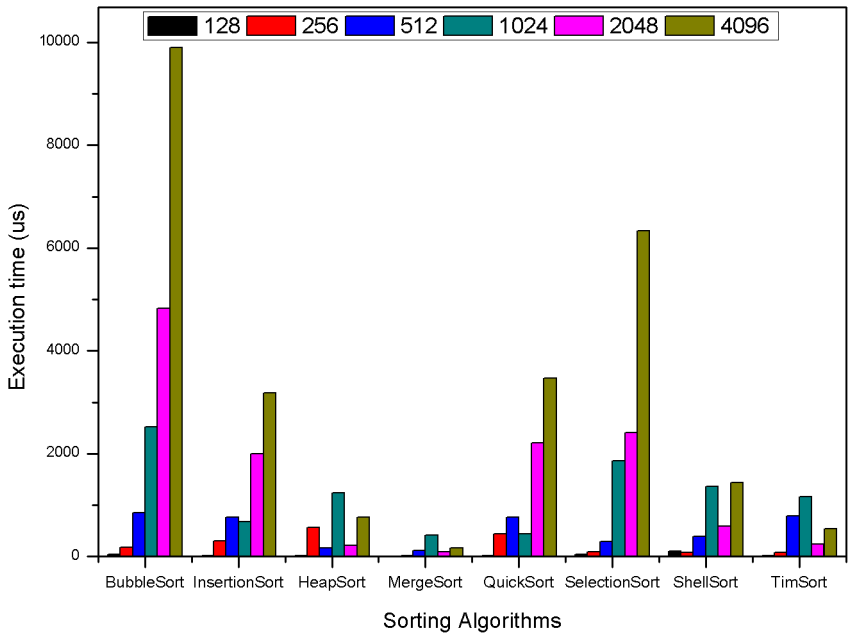


FIGURE 2.12 – Ecart type des temps d'exécution pour les algorithmes de tri en l'exécutant sur le CPU

dispersion des valeurs autour de la moyenne. Pour cela, plus la valeur du coefficient de variation est élevée, plus la dispersion autour de la moyenne est grande. Cependant, Un coefficient de variation est inférieur à 5% est considéré comme suffisant en générale pour que la moyenne soit représentative, pourvu que le nombre d'éléments dans la population soit suffisant (i.e. empiriquement >=30)et que la distribution des valeurs dans la population suive une loi gaussienne. Par contre, si le coefficient est supérieure à 5 alors on a intérêt à représenter graphiquement les résultats avec des boites à moustaches qui permettent de donner une idée de la dispersion (i.e. la répartition des durées effectives à +/- 2 à 3 σ) (voir annexe A). Nous pouvons conclure que la stabilité des algorithmes de tri en termes de temps d'exécution est significatif si le coefficient de variation est inférieur à 5%. Nous

Tableau 2.3 – Coefficient de variation des algorithmes de tri sur le processeur

	BubbleSort (us)	InsertionSort (us)	SelectionSort (us)	HeapSort (us)	QuickSort (us)	ShellSort (us)	MergeSort (us)	TimSort (us)
8	175.35	202.35	175.86	158.144	129.05	164.2	121.18	160.12
16	83.59	131.39	105.65	89.76	93.36	98.635	96.44	127.52
32	29.98	62.93	63.89	49.1	47.74	50.25	45.85	48.91
64	115.79	32.29	24.08	49.1	141.08	161.41	30.94	29.73
128	8.47	10.39	25.82	16.55	14.88	85.18	14.35	12.4
256	17.54	74.91	14.3	188.05	144.05	22.61	12.36	39.46
512	22.33	54.29	13.36	35.94	66.68	33.97	42.14	154.3
1024	20.49	14.86	25.57	82.42	13.29	38.61	52.87	102.07
2048	10.2	8.027	8.61	10.04	20.1	4.8	5.48	12.72
4096	5.45	4.75	5.89	17.66	8.27	3.05	5.34	13.15

remarquons d'après le tableau 2.3 que le coefficient de variation pour les algorithmes de

tri est en moyenne supérieure à 5. Pour cela, nous pouvons représenter les résultats sous forme des Boxplots (Boite à moustaches) pour visualiser et comparer les distributions de réplications sur une même échelle. Cette représentation graphique peut être une solution pour approcher les concepts abstraits de la statistique. Pour lire et interpréter une boîte à moustaches, il est nécessaire de connaître sa construction. La boîte à moustaches utilise six valeurs qui résument des données : le minimum, la moyenne, les trois quartiles Q1, Q2 (médiane) qui est la donnée de la série qui sépare les 50 % inférieurs des données , Q3 qui est la donnée de la série qui sépare les 75 % inférieurs des données et le maximum. Les quartiles Q1 qui est la donnée de la série qui sépare les 25 % inférieurs des données, Q2, Q3 sont les éléments essentiels de ce graphique. Pour étudier expérimentalement la stabilité des algorithmes, tout d'abord il faut vérifier que le pourcentage de nombre d'outliers supérieur est inférieure à 5%. De plus, il faut vérifier que la boite de moustache est symétrique par rapport la médiane et la moyenne. Dans ce cas, nous pouvons dire que cet algorithme suit la loi normale. Ensuite, si la valeur de la médiane sont très différente d'une distribution à une autre. Donc, la durée de tri varie et l'algorithme n'est pas stable en fonction de données en entrée. Sinon il est stable ce qui est particulièrement intéressante pour les applications temps réelle visée. Nous avons utilisé le langage R [93] pour tracer les boites à moustaches. Ce langage dispose d'un ensemble riche de bibliothèques statistiques pour compenser cet écart. Nous avons tracé 80 graphes pour chaque algorithme de tri et plus précisément pour chaque taille d'éléments (8->4096) afin de visualiser et comparer les permutations qui ont comme objectif le changement de l'ordre des entrées pour obtenir plusieurs variations temporelles en termes de la moyenne des temps d'exécution et de la médiane des boites à moustache. Chaque graphe contient 47 permutations (47 Boxplots). Nous ne pouvons pas mettre tous les graphes dans ce chapitre (Voir Annexe A). Comme indiqué auparavant, l'algorithme MergeSort et insertionSort sont plus rapide en termes de temps d'exécution et d'écart type si N>64 et N<=64. Pour cela, nous présentons dans cette parti les différents Boxplots de l'algorithme MergeSort pour les nombres de données 128, 256, 512, 1024, 2048, 4096. La Figure 2.14 et le tableau 2.5 montrent les cinq valeurs qui résument les données : Q0(Min), Q1, Q2(Median), Q3 et Q4(Max) pour les 47 permutations. Les valeurs sont les critères de comparaison entre les différentes distributions. D'une part, Nous observons dans la figure 2.14 des valeurs aberrantes (outliers) qui donne des renseignements sur les éventuelles erreur de mesure (Par exemple : erreur de transmission, erreur liée au bruit gnéré par le système d'exploitation, bufferisation des E/S). Ces valeurs sont définies par Chandola et al [94] en tant que "modèles de données qui ne sont pas conformes à une notion bien définie de comportement normal". Cette définition est très générale et repose sur la manière dont les observations ou les modèles diffèrent du comportement normal. En d'autres termes, les anomalies représentent des données situées en dehors d'une plage normale spécifique. La figure 2.13 présente un exemple de cinq anomalies étiquetées V, W, X, Y et Z qui sont clairement isolées avec la plage de données normale.

Tableau 2.4 – Pourcentages d'outliers pour l'algorithme MergeSort

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
MergeSort	128	9.5	5
	256	4	3,4
	512	3.91	2,7
	1024	3.93	3,9
	2048	2.77	2,7
	4096	8.18	8,18

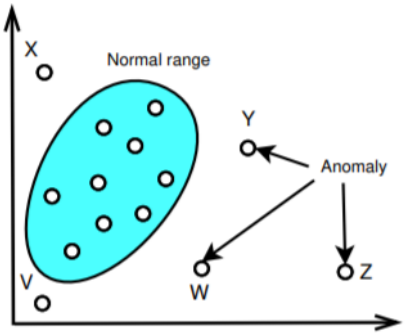


FIGURE 2.13 – Nombres aberrantes

Par la suite, la figure 2.14 montre l'asymétrie des moustache. Nous observons que le nombre d'outliers est important. Pour cela, nous avons calculé le pourcentage d'outliers (surtout sur la moustache supérieure) pour chaque taille d'élément. Le tableau 2.4 illustre les pourcentages d'outliers pour chaque taille des permutations. Nous constatons que le pourcentage pour N=128/256/512/1024/2048 est varié entre 2% et 4%. Donc, nous pouvons conclure que ces distributions suivent la loi normale. Par contre, le nombre de données aberrantes est important pour N=4096 pour un taille important de données. Ensuite, la figure 2.15 présente les variations temporelles pour N>64 de la médiane et de la moyenne des permutations trouvés quand on lance 1000 itérations, nous constatons que les valeurs de la médiane est presque proche dans la plus part des cas par contre la valeur de la moyenne varié. Nous pouvons déduire que la moyenne est une mesure qui est sensible fasse aux données aberrantes au niveau des données calculés à partir de la formule suivante :

$$Pourcentage\_Outliers = 100 * N\_Out/N\_R * N\_P \tag{2.8}$$

Avec N_Out est le nombre de données aberrantes supérieur, N_R est le nombre de réplication utilisé dans ce travail (fixé à 1000) et N_P est le nombre de permutation qui est fixé à 47. A partir de l'équation 2.8, nous calculons la valeur du pourcentage pour tous les algorithmes de tri. Nous constatons que cette valeur augmente avec l'augmentation du nombre N.

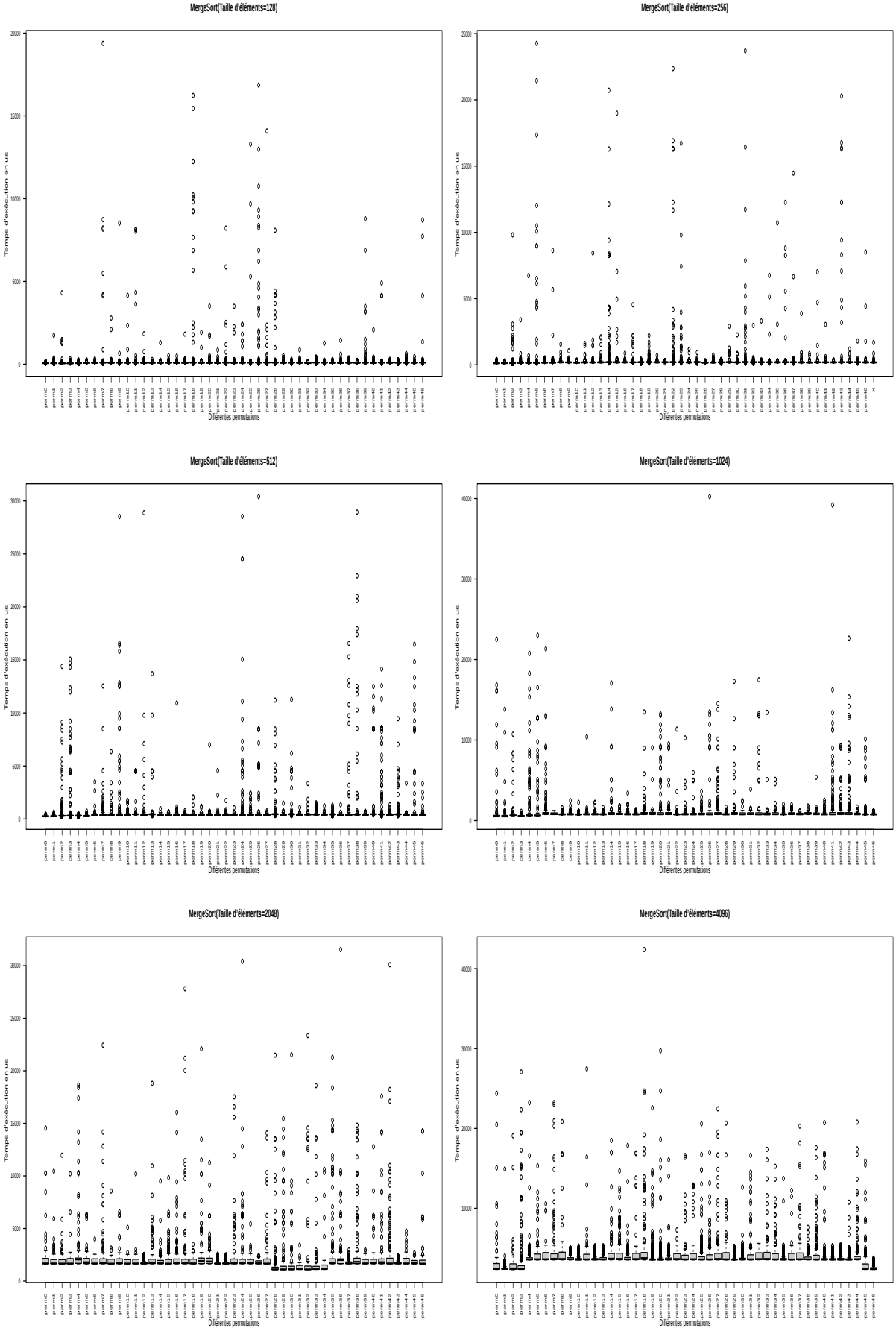


FIGURE 2.14 – Boxplots de l'algorithme MergeSort avec N>64

Tableau 2.5 – Critères de performances pour l'algorithme MergeSort (N>64)

	4096						2048						1024						512						256						128					
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	2341	2368	2445,5	2773,407	3087	24101	719	1629	1645	1674	1909,943	2148	14550	503	498	513	525	697,883	659,25	22530	146,25	229	231	251	268,675	297	464	66	103	103	123	129,479	144	408	41	42	55	61	60,07	65	177	10
perm1	2371	2380	2473	2723,341	3059,25	19081	693,25	1631	1647	1661,5	1835,669	2040,25	11972	393,25	499	561	654	686,211	686	10733	125	229	295	305	419,504	329	14390	34	103	133	138	174,201	152	968,10	19	42	43	55	67,51	62	431,5	19
perm2	2317	2355	2473	2723,341	3059,25	19081	693,25	1631	1647	1661,5	1835,669	2040,25	11972	393,25	499	561	654	686,211	686	10733	125	229	295	305	419,504	329	14390	34	103	133	138	174,201	152	968,10	19	42	43	55	67,51	62	431,5	19
perm3	2355	2379	2402	2851,851	2810,25	270,77	431,25	1630	1646	1663	1841,42	2135,25	102303	489,25	503	518	528	578,479	627,25	1152	109,25	232	300	3241,5	486,488	337	15077	37	117	143	156	165,929	170	3402	27	43	43	56	53,742	56	321	13
perm4	3539	3564	3591,5	3823,683	3808,5	23228	244,5	1636	1684,75	2050,5	2106,476	2185	18641	500,25	498	515	659,5	817,119	700	20764	185	229	298	318	319,112	335	507	37	103	135	150	157,482	164	6736	29	43	62	65	67,598	69	239	7
perm5	3536	3566	3617,5	3930,793	4274,75	15317	706,75	1631	1656	1787	1929,522	2151	6336	495	502	518	617	784,86	694	23034	176	231	284	284,712	317	718	84	115	141	151	296,154	156	24263	15	68	69	81	83,589	93	257	24
perm6	3537	3575	3674	4094,303	4311,5	23221	736,5	1636	1657	1755	1976,723	2108	22432	451	751	765	833,5	986,789	965	1238	200	337	351	430	464,697	462	12542	111	145	158	160	194,367	195	8639	37	73	92	93	130,442	93	19380	1
perm7	3536	3575	3674	4094,303	4311,5	23221	736,5	1636	1657	1755	1976,723	2108	22432	451	751	765	833,5	986,789	965	1238	200	337	351	430	464,697	462	12542	111	145	158	160	194,367	195	8639	37	73	92	93	130,442	93	19380	1
perm8	3540	3572	3657	4051,84	4510,25	20849	936,25	1629	1646	1656	1822,902	2075	8563	429	732	755	767	823,153	837	19386	82	337	352	423,5	430,805	452	6356	100	103	135	150	157,482	164	6736	29	43	62	65	67,598	69	239	7
perm9	3539	3569	3622	3811,449	3821,25	5036	252,25	1629	1649	1724,5	1894,417	2109,25	6578	460,25	749	752	768	829,209	873,25	2478	121,25	335	348	413	509,475	453	28523	105	147	160	192	190,19	201	10566	41	71	90	93	104,067	98	8522	8
perm10	3537	3558	3573	3720,466	3699,25	5279	141,25	1635	1650	1665	1807,663	2041,75	5105	391,75	749	751	763	798,215	793,5	2308	42,5	3334	3347	3409	385,545	4291,25	1739	73,25	1146	1159	1160	181,552	199	359	40	67	81	92	97,944	95	4160	14
perm11	3535	3561	3607,5	3928,448	4250	27430	689	1633	1649	1671,5	1841,104	2071,25	10198	422,25	732	756	768	835,968	869	10399	113	3395	429	447	518,571	459	4575	30	1146	1160	1160	191,32	200	1615	40	71	88	93	125,023	94	8153	6
perm12	3536	3563	3596	3732,468	3700	5308	137	1635	1652	1671	1745,713	1741,25	2531	89,25	732	756	769	835,501	886,5	2248	130,5	3336	350	435	477,989	455	28882	105	1146	1159	1185	198,46	200	8449	41	68	69	81	87,368	95	1839	26
perm13	3540	3568	3601	3811,509	3827,5	5323	239,5	1632	1657	1765	1930,87	2076,25	18803	419,25	732	756	768	817,625	851,25	1640	95,25	3337	428	442	489	481	13607	53	1146	1160	1193	199,368	215	2114	55	68	69	90,5	87,538	96	282	27
perm14	3543	3590	3695	4039,085	4346,25	18503	794,25	1635	1654	1700,5	1838,384	1961	9517	307	730	764	911	952,165	986	17090	222	3336	350	406,5	469,164	451	974	101	1146	1198	203	379,922	216	20720	18	68	72,75	89	88,334	95	1301	22,25
perm15	3538	3573,75	3608	4043,678	4379,25	14683	805,5	1631	1647	1660	1862,764	2053,25	8829	406,25	750	765	810	847,826	921,25	1857	156,25	3336	349	381	469,672	447	699	98	1145	1162	197	233,632	215	18099	53	69	82	94	94,879	99	519	17
perm16	3539	3565	3613	3844,881	3849	17874	284	1634	1654	1724	1944,126	2097	16041	443	753	759	770	835,139	883	3412	124	3336	351	418	424,045	447	10593	96	1146	1161	194	194,725	212	878	51	69	70	82	86,891	96	494	26
perm17	3537	3569	3632	3949,807	4358,25	16932	780,25	1634	1655	1724	1991,78	2074,5	27985	4119,5	748	751	764	838,077	867,25	1371	116,25	3336	351	353	390,332	431	800	80	1146	1170	194	209,343	201	4534	22	67	68	80	85,8	93	1818	25
perm18	3540	3600,75	3764	4254,35	4464,75	42392	864	1635	1650	1677	1843,361	2078	34497	428	751	765	940	936,651	1004	13405	239	3336	349	551	383,833	414,25	2083	65,25	1146	1192	201	201,667	215	429	23	72	92	95	226,12	96	16229	4
perm19	3541	3564	3571	3858,27	3840,25	22582	76,25	1637	1657	1905	2026,835	2183,25	22087	526,25	750	765	974	917,386	1004	9048	239	3335	349	360	401,01	449	12202	100	1146	1197	211	224,653	225	2212	28	67	68	90	91,037	98	1922	30
perm20	3521	3551	3586,5	3873,844	3680,25	29725	129,25	1634	1675,75	1863	1977,25	2173	112252	497,25	732	766	791	1006,911	991	13239	225	3337	351	352	391,926	420	6993	69	1145	1160	196	196,57	217	711	57	67	68	81	91,394	92	3507	24
perm21	3538	3565	3602	3811,367	3773,25	16076	208,25	1635	1649	1657	1704,029	1693	2564	44	749	763	938	945,599	1004	9541	241	3336	356	424	422,719	455,25	4579	99,25	1147	1188	198	199,588	215	323	27	68	68	77	83,632	93	855	25
perm22	3535	3567	3633,5	3910,451	4323	7292	736	1633	1650	1663	1745,143	1720	2504	70	751	755	766	822,308	823,75	11356	68,75	3336	349	372	398,023	435	1743	86	1147	1197	204	344,289	216,25	22365	19,25	73	92	93	112,687	93	8219	1
perm23	3534	3564,75	3606	3905,222	4121,75	16607	557	1638	1655	1690	1938,409	2092,5	17324	4375	732	755	768	842,87	874,25	10268	119,25	3336	357	425	417,307	451	1058	94	1148	1193	200	258,773	219	16717	26	68	69	83	92,273	93	3498	24
perm24	3536	3568	3626	3938,705	4328,25	12923	760,25	1639	1656	1712	1971,592	2071	30411	415	752	756	769	849,327	904,25	5972	148,25	3336	349	3693,5	568,85	449	28539	100	1146	1159	190	191,102	200	11177	41	68	77	90	99,215	96	2419	19
perm25	3541	3567	3601	3940,755	3935,75	20580	366,75	1635	1658	1752	1868,486	2057,25	5440	399,25	751	757	775	854,042	935,25	3027	178,25	3336	351,75	428	432,048	455,25	2435	103,5	1146	1197	210	212,138	226	933	29	75	92	93	121,099	94	13289	2
perm26	3538	3566	3604,5	3924,13	4176,75	16986	610,75	1633	1652	1690,5	1796,263	1924,25	2444	272,25	732	756	771,5	981,646	966	40247	210	3335	349	418	500,737	451	30410	102	1145	158	158	170,169	181,25	293	23,25	68	90	95	214,495	98	16680	8
perm27	3537	3571,75	3661	4067,406	4356,25	22458	784,5	1632	1647	1663	1895,114	2093	14084	446	751	764	914,5	1005,824	993,25	14543	229,25	3336	350	392	389,045	436	764	86	1146	159	178,5	189,762	207	746	48	73	92,75	95	117,418	95	14084	2,25
perm28	3533	3562	3612	3998,609	4448,25	20603	886,25	1632	1647	1663	1823,559	1314	21483	216	751	754	767	822,524	864	1679	110	3338	345	456	526,012	480	11206	35	1147	186	195	196,126	204	403	18	69	92	96	136,935	98	8087	6
perm29	3541	3566	3578	3691,964	3635,25	5241	69,25	1631	1646	1663	1866,095	1407,5	15456	311,5	732	768	942	966,697	991,25	17200	223,25	3337	353	420	419,097	456	1495	103	1146	179	198	208,015	224	2919	45	68	73	91	89,567	97	513	24
perm30	3536	3566	3604	3795,095	3750	12647	184	1634	1699	1716	1805,106	1410,75	21522	311,75	750	755	779	855,655	970	2472	215	3336	430	449,5	504,192	476	11265	46	1145	161	192	194,749	198	2265	37	71	90	93	94,412	97	296	7
perm31	3538	3563	3609,5	3989,576	4175,75	16630	612,75	1634	1710	1739	1926,329	1484	3412	374	751	767	840	887,672	983,25	3803	216,25	3335	348	532	384,52	424	1092	76	1146	194	203	297,984	219	23699	25	68	76	90	89,937	95	863	19
perm32	3546	3573	3645	3926,934	4435,75	5614	862,75	1634	1696	1704	1952,661	1421	23345	325	748	752	778	964,75	977,25	17489	225,25	3336	386	440	433,221	454	3351	68	1146	194	200	207,943	218	2977	24	68	68	69	80,448	93	174	25
perm33	3544	3577	3664	4031,388	4483,25	17381	906,25	1634	1656	1679	1832,518	2051,25	2681	309,25	751	756	797,5	864,456	956,25	5377	200,25	3337	351	388	401,844	437,25	726	86,25	1146	159	189	193,528	200	3872	41	68	69	90	88,563	98	230	29
perm34	3537	3565	3585	3835,724	3716	13730	151	1635	1656	1748	2083,213	2111	21279	455	752	756	771	819,042	838	2047	82	3337	419	446	443,133	461	1299	42	1146	160	192	203,573	210	10712	50	67	68	88	85,054	94	282	26
perm35	3540	3565	3585	3835,724	3716	13730	151	1635	1656	1748	2083,213	2111	21279	455	752	756	771	819,042	838	2047	82	3337	419	446	443,133	461	1299	42	1146	160	192	203,573	210	10712	50	67	68	88	85,054	94	282	26
perm36	3541	3566	3593	3898,874	4349,5	12238	783,5	1636	1656	2003,5	2029,14	2110,25	31528	454,25	750	768	889,5	897,365	980,25	1986	212,25	3334	347	382,5	404,364	452	13389	105	1170	197	201	253,55	213	12271	16	67	67	80	84,855	94	1446	27
perm37	3537	3567	3625,5	3997,222	4408,5	20288	841,5	1633	1651	1685	1777,784	1821	2935	170	750	755	779	843,471	922	1199	167	3338	352	425	526,7	468,25	16561	116,25	1147	192	201	226,707	218	14469	26	69	81	93	93,477	98	384	17
perm38	3541	3568	3620	3818,283	3893,25	7619	325,25	1634	1656	1824	2074,251	2168	14830	512	749	787	936	912,356	983,25	1823	196,25	3338	393,75	438	621,145	455	28945	61,25	1146	159	189	193,528	200	3872	41	68	69	90	88,563	98	230	29
perm39	3537	3560	3602	3972,677	4115,75	17602	555,75	1636	1652	1679	1832,518	2051,25	2681	309,25	751	756	797,5	864,456	956,25	5377	200,25	3337	351	388	401,844	437,25	726	86,25	1146	159	174,5	186,249	204	856	45	70	87	92	125,451	99	8786	12
perm40	3538	3564	3587	3816,195	3694	20716	130	1632	1652	1731,5	1903,827	2071	12779	419	751	755	767	842,937	868,75	2533	113,75	3338	349	456	574,405	459,25	12495	10,25	1146	174	194	207,778	210	7024	36	68	69	81	87,388	94	2079	25
perm41	3536	3560	3588	3741,061	3693	5332	133	1636	1657	1790,5	1973,282	2099,25	17389	4422,25	753	766	777	1043,851	1006,25	39203	240,25	3336	426,75	444	628,694	458	14140	31,25	1146	162,75	195	198,36	211	3052	48,25	79	92	94	110,953	96	4902	4
perm42	3535	3560	3582,5	3750,429	3721,25	5300	161,25	1636	1651	1674	2027,13	2171,25	30084	520,25	751	766	928,5	990,757	1006	9337	240	3337	352	395	405,388	432	1345	80	1147	161	188,5	188,803	209	373	48	68	75	92	90,392	98	293	23
perm43	3539	3565	3578,5	3730,493	3669	10765	104	1637	1653	1678	1743,389	1763	24403	110	751	765	838,5	1049,607	1000,25	22945	235,25	3336	435	445	498,368	468,25	9460	43,25	1146	178	196	325,132	211	20288	33	74	90	93	93,765	95	265	5
perm44	3542	3576	3635	3937,205	3938,25	20794	382,25	1639	1655	1674,5	1867,201	2153	4735	498	752	766	782	847,367	932,5	1608	166,5	3336	349	372	392,177	425	3376	76	1145	189	196	204,453	226	11935	37	71	89	92	95,985	98	643	9
perm45	2340	2369	2440,5	2760,241	3034,25	13912	665,25	1629	1648	1690	1801,68	1954,25	26099	3406,25	751	755	767	880,171	814,25	10119	59,25	3336	349	398	540,963	452	16478	103	1147	193	202	203,846	215	13804	22	67	76,75	92	90,544	96	461	17,25
perm46	2339	2362	2395	2539,302	2568	3762	206	1633	1647	1656	1844,059	1860	14281	333	748	752	763	815,053	840	12334	88	3336	349	376	406,148	443,25	3337	94,25	1145	159	192	204,308	209	8520	50	75	95	95	117,652	96	8711	1

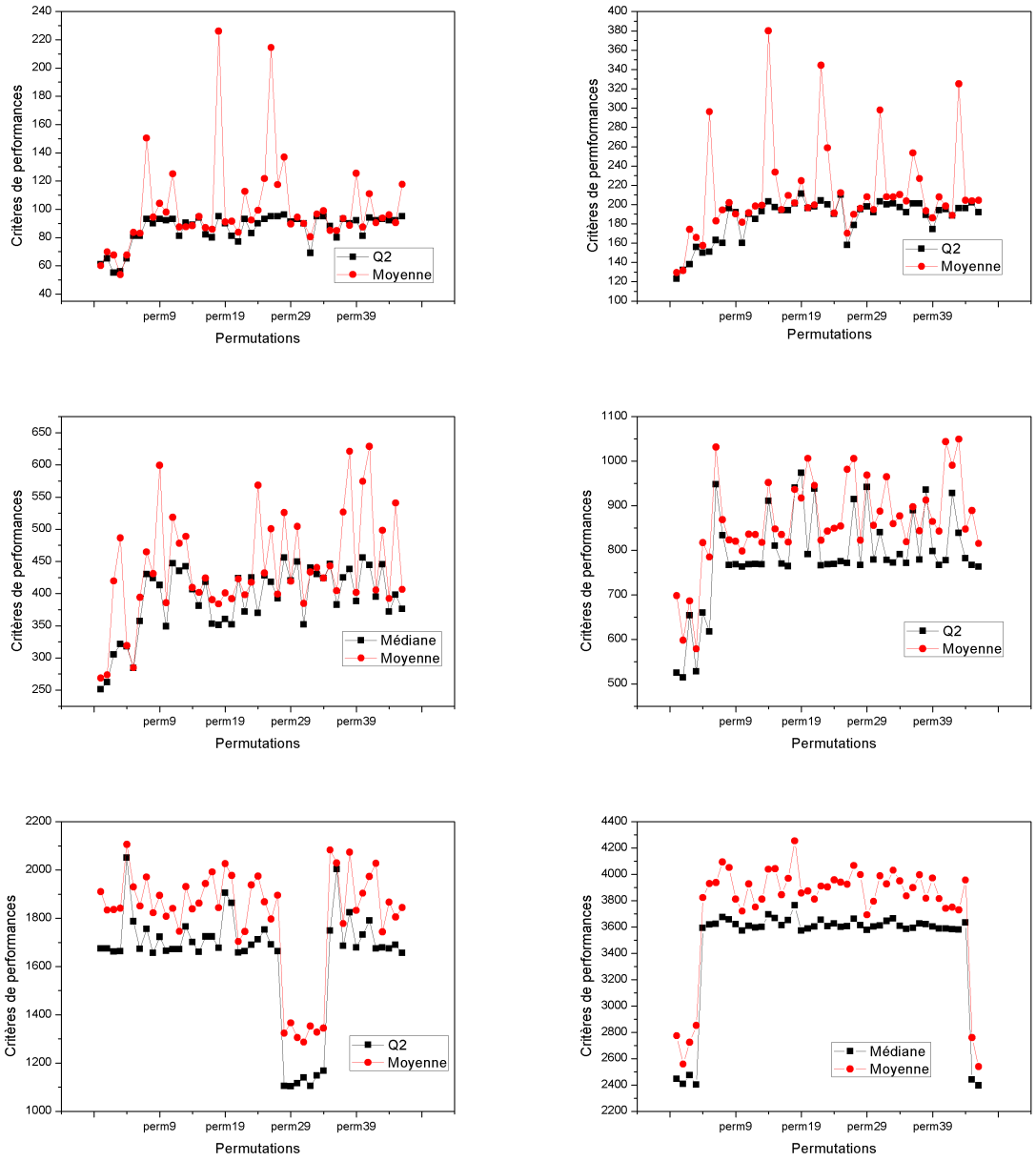


FIGURE 2.15 – Médiane et Moyenne de l'algorithme MergeSort avec N>64

Par la suite, nous étudions statiquement l'algorithme InsertionSort en prenant dans ce cas N=8, 16, 32 et 64. Alors, nous obtenons les résultats affichés dans les Figures 2.16 et le tableau 2.7. Tableau 2.7 montre que les valeurs pour Q0, Q1, Q2, Q3 et Q4 tendent vers 0 pour la taille d'élément égale 8 c'est à dire que si IQR=Q3-Q1=0 alors l'écart type est proche de 0 donc les durées sont quasi identiques dans ce cas. De plus, nous choisissons un autre critère de performance qui est la valeur de la moyenne. Nous observons que les boites à moustache n'est pas symétrique par rapport la médiane. Nous passons maintenant à calculer le pourcentage d'outliers.

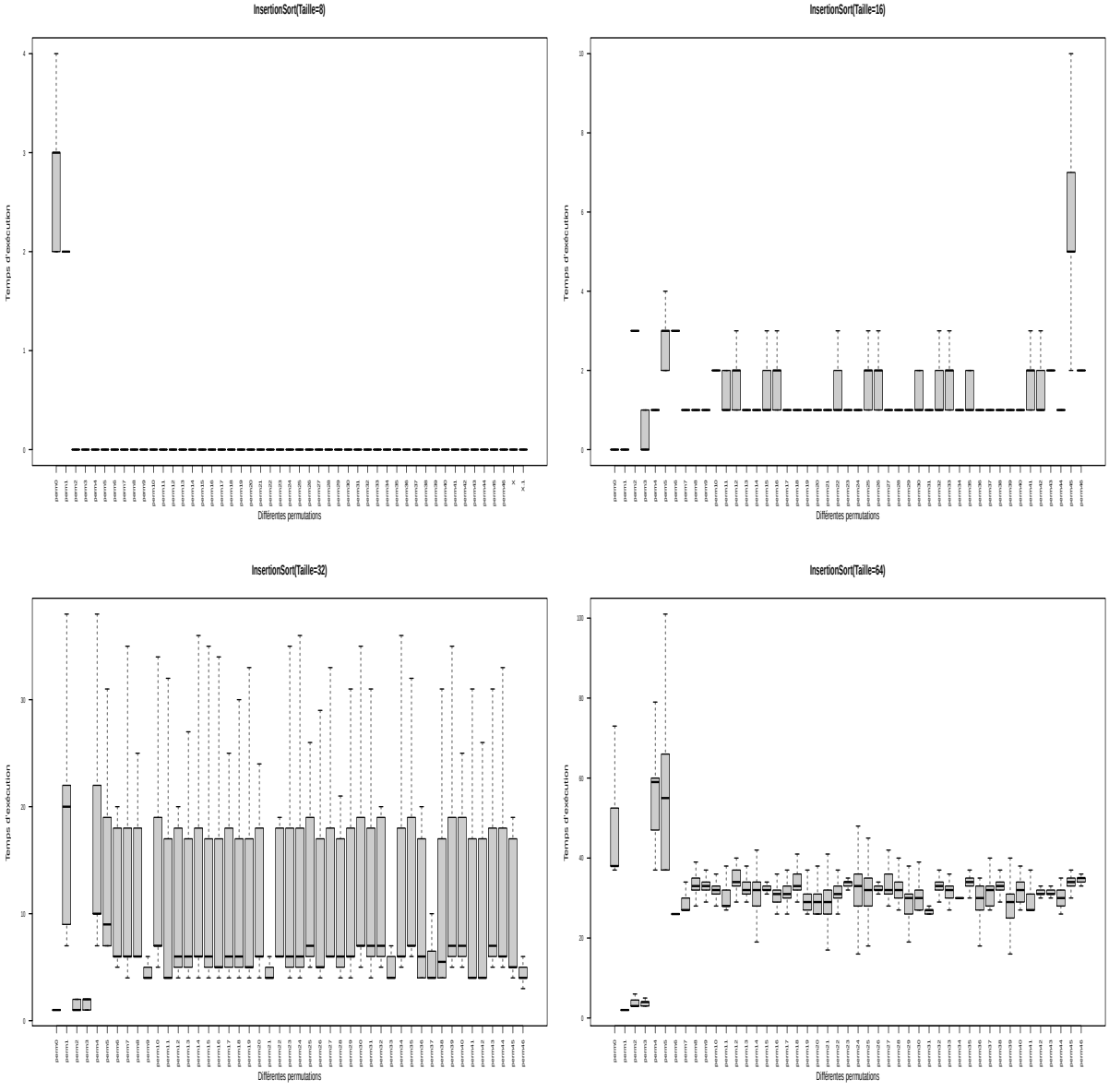


FIGURE 2.16 – Boxplots de l'algorithme InsertionSort avec N<=64

Tableau 2.6 – Critères de performances pour l'algorithme InsertionSort (N<=64)

	8						16						32						64					
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	2	2	3	4,334	3	46	1	0	0	0	0,57	0	32	0	1	1	1	1,84984985	1	20	0	37	38	38	48,462	52,25	234	14,25
perm1	1	2	2	3,466	2	118	0	0	0	0	0,521	0	13	0	7	9	20	16,5605606	22	165	13	2	2	2	5,618	2	1424	0
perm2	0	0	0	0,473	0	106	0	2	3	3	6,09	3	217	0	1	1	1	2,7027027	2	35	1	3	3	3	6,027	4,25	61	1,25
perm3	0	0	0	0,611	0	72	0	0	0	0	1,127	1	87	1	1	1	2	2,9049049	2	91	1	3	3	4	32,188	4	12903	1
perm4	0	0	0	0,404	0	13	0	0	1	1	1,528	1	14	0	7	10	10	15,4684685	22	294	12	37	47	59	81,392	60	9117	13
perm5	0	0	0	0,365	0	13	0	2	2	3	4,881	3	102	1	7	7	9	13,5015015	19	227	12	37	37	55	54,298	66	810	29
perm6	0	0	0	0,563	0	13	0	2	3	3	4,69	3	15	0	5	6	6	14,3673674	18	4251	12	26	26	26	25,029	26	155	0
perm7	0	0	0	0,619	0	54	0	1	1	1	2,181	1	66	0	4	6	6	11,6386386	18	557	12	14	27	27	26,838	30	283	3
perm8	0	0	0	0,575	0	13	0	1	1	1	2,02	1	35	0	6	6	6	36,1791792	18	8735	12	15	32	33	44,341	35	6707	3
perm9	0	0	0	8,558	0	8095	0	1	1	1	2,001	1	84	0	4	4	4	7,38738739	5	154	1	17	32	33	91,953	34	17364	2
perm10	0	0	0	0,498	0	13	0	1	2	2	3,271	2	15	0	5	7	7	11,5755756	19	87	12	15	31	32	34,523	33	1167	2
perm11	0	0	0	0,633	0	194	0	1	1	1	2,522	2	95	1	4	4	4	8,32832833	17	58	13	15	28	28	30,16	32	430	4
perm12	0	0	0	0,544	0	13	0	1	1	2	2,986	2	63	1	4	5	6	10,003003	18	160	13	16	33	34	39,959	37	3406	4
perm13	0	0	0	0,325	0	13	0	1	1	1	2,151	1	85	0	4	5	6	9,47047047	17	191	12	17	31	32	70,618	34	17618	3
perm14	0	0	0	0,407	0	13	0	1	1	1	2,09	1	28	0	4	6	6	10,3553554	18	258	12	15	28	32	32,028	34	192	6
perm15	0	0	0	0,408	0	13	0	1	1	1	5,164	2	2465	1	4	5	6	9,46146146	17	129	12	16	32	32	34,581	33	229	1
perm16	0	0	0	0,456	0	13	0	1	1	2	2,771	2	34	1	4	5	5	8,65365365	17	64	12	14	29	31	30,823	32	127	3
perm17	0	0	0	0,366	0	13	0	1	1	1	1,781	1	36	0	4	5	6	22,7637638	18	4076	13	14	30	31	32,989	33	492	3
perm18	0	0	0	0,46	0	13	0	1	1	1	2,224	1	52	0	4	5	6	9,03203203	17	30	12	17	32	33	38,31	36	2054	4
perm19	0	0	0	0,459	0	13	0	1	1	1	2,217	1	15	0	4	5	5	13,2162162	17	4692	12	14	27	29	28,504	31	140	4
perm20	0	0	0	0,781	0	322	0	1	1	1	2,539	1	387	0	4	6	6	11,9179179	18	413	12	14	26	29	31,789	31	2584	5
perm21	0	0	0	0,314	0	13	0	1	1	1	2,21	1	62	0	4	4	4	7,44644645	5	132	1	14	26	29	28,745	32	184	6
perm22	0	0	0	0,354	0	47	0	1	1	1	2,603	2	143	1	6	6	6	50,2292292	18	16066	12	14	30	31	31,941	33	236	3
perm23	0	0	0	0,421	0	13	0	1	1	1	2,37	1	14	0	4	5	6	11,018018	18	483	13	17	33	34	36,204	34	246	1
perm24	0	0	0	0,527	0	137	0	1	1	1	2,729	1	243	0	4	5	6	10,0710711	18	126	13	16	28	33	34,534	36	332	8
perm25	0	0	0	0,604	0	13	0	1	1	2	3,05	2	72	1	5	6	7	12,9309309	19	1020	13	15	28	32	32,93	35	144	7
perm26	0	0	0	0,368	0	13	0	1	1	2	3,476	2	421	1	4	5	5	8,9669697	17	91	12	16	32	32	33,613	33	139	1
perm27	0	0	0	0,319	0	13	0	1	1	1	2,307	1	54	0	6	6	6	16,2642643	18	3621	12	17	31	32	37,778	36	2714	5
perm28	0	0	0	0,634	0	255	0	1	1	1	2,646	1	655	0	4	5	6	9,47847848	17	110	12	15	30	32	33,063	34	224	4
perm29	0	0	0	0,67	0	13	0	1	1	1	2,224	1	61	0	4	6	6	23,998999	18	8194	12	14	26	30	29,255	31	262	5
perm30	0	0	0	0,339	0	13	0	1	1	1	2,221	2	33	1	5	7	7	11,7927928	19	83	12	14	27	30	32,008	32	2170	5
perm31	0	0	0	0,32	0	59	0	1	1	1	2,466	1	141	0	4	6	7	30,6926927	18	12144	12	14	26	27	25,94	27	243	1
perm32	0	0	0	0,392	0	13	0	1	1	1	3,09	2	218	1	5	6	7	12,1521522	19	998	13	15	32	33	39,836	34	3116	2
perm33	0	0	0	0,287	0	13	0	1	1	2	2,856	2	64	1	4	4	4	7,61761762	6	149	2	15	30	32	72,143	33	8143	3
perm34	0	0	0	0,386	0	13	0	1	1	1	2,196	1	25	0	5	6	6	10,6086086	18	64	12	16	30	30	38,792	30	6987	0
perm35	0	0	0	0,303	0	13	0	1	1	1	2,639	2	86	1	6	7	7	12,03003	19	92	12	15	33	34	36,633	35	643	2
perm36	0	0	0	0,354	0	13	0	1	1	1	2,134	1	14	0	4	4	6	9,0090901	17	152	13	14	27	30	29,708	33	121	6
perm37	0	0	0	0,24	0	13	0	1	1	1	1,923	1	51	0	4	4	4	7,77577578	6,25	352	2,25	15	28	32	32,629	33	977	5
perm38	0	0	0	2,685	0	1976	0	1	1	1	2,177	1	98	0	4	4	5,5	8,79479479	17	86	13	15	32	33	35,105	34	706	2
perm39	0	0	0	0,499	0	51	0	1	1	1	1,898	1	14	0	5	6	7	11,1631632	19	198	13	13	25	29	30,753	31	1113	6
perm40	0	0	0	0,379	0	13	0	1	1	1	2,318	1	14	0	5	6	7	14,8348348	19	1543	13	15	29	32	32,86	34	193	5
perm41	0	0	0	0,554	0	29	0	1	1	2	10,586	2	7594	1	4	4	4	7,95995996	17	201	13	15	27	27	27,601	31	217	4
perm42	0	0	0	1,176	0	686	0	1	1	1	2,713	2	33	1	4	4	4	7,70570571	17	66	13	14	31	31	32,83	32	410	1
perm43	0	0	0	0,534	0	13	0	1	2	2	11,304	2	8045	0	5	6	7	12,4794795	18	711	12	17	31	31	31,834	32	132	1
perm44	0	0	0	0,261	0	13	0	1	1	1	2,259	1	31	0	5	6	6	10,491491	18	70	12	13	28	30	30,182	32	189	4
perm45	0	0	0	0,386	0	13	0	2	2	3	4,33433433	3	46	1	4	5	5	8,42342342	17	85	12	28	33	34	43,568	35	743	2
perm46	0	0	0	0,66	0	254	0	1	2	2	3,466	2	118	0	3	4	4	7,06606607	5	92	1	18	34	35	114,927	35	21842	1

Tableau 2.7 – Pourcentage d'outliers pour InsertionSort

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
InsertionSort	8	4.8	4,5
	16	12.3	11,3
	32	3.5	3,5
	64	16.13	9,3

Le tableau 2.7 montre que la pourcentage d'outliers est important pour N=16 et N=64, ce qui signifie que ces distribution ne suit pas la loi normale. Figure 2.17 indique les

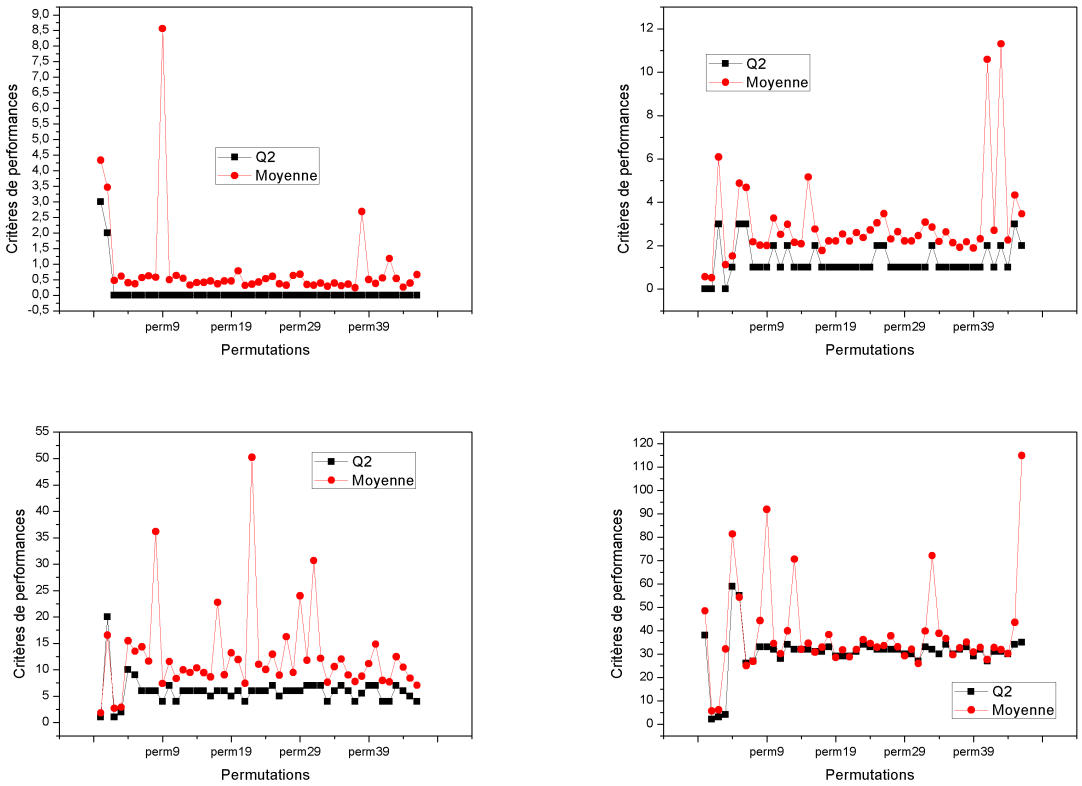


FIGURE 2.17 – La Médiane et la moyenne de l'algorithme InsertionSort avec N<=64

résultats obtenus pour la médiane et la moyenne de l'algorithme InsertionSort si N<64. Nous constatons que la valeur de la médiane reste presque constant avec R=1000 réplications. Par contre, la moyenne des temps d'exécution est varié à cause des perturbations matérielles. Afin de comparer la mesure expérimentale de la complexité en temps des algorithmes de tri , nous avons calculé la moyenne des temps d'exécution de chaque algorithme en fonction du taille de tableau pour donner une idée sur la variation temporelle de l'algorithme en fonction de N. Pour cette finalité, nous avons procédé à calculer la dispersion du temps d'exécution dont les résultats sont montrés dans les figures 2.18 et 2.19.

2.5.4 Dispersion statistique

La dispersion statistique apparaît lorsque des mesures répétées de la même taille sont effectuées. En effet, si nous mesurons le même phénomène de manière répétée avec un appareil suffisamment précis, nous pouvons avoir un résultat différent. Cela est dû à des phénomènes perturbateurs ou, pour des mesures extrêmement précises, à la nature aléatoire du phénomène. L'erreur due à la dispersion statistique est ensuite estimée à l'aide de la formule suivante :

$$D = k * Ecart\_type \tag{2.9}$$

Avec k représente une valeur constante. En physique, on définit souvent k = 3, ce qui correspond à un intervalle de confiance de 99,73%, c'est-à-dire que 99,73% des valeurs xi sont dans l'intervalle de confiance et 0,27% seront en dehors de cet intervalle [95]. Pour chaque algorithme, nous mesurons à la fois la marge d'erreur observée due à la dispersion statistique et la marge d'erreur calculée due à la dispersion statistique.

- **Marge d'erreur observée** : La marge d'erreur observée est calculée comme suit : Nous avons un ensemble de donnée à trier. Nous calculons la limite supérieure (avec formule 2) ainsi que la limite inférieure (formule 3) en utilisant les formules suivantes :

$$LBound = Temps - ecart\_type \tag{2.10}$$

$$UBound = Temps + ecart\_type \tag{2.11}$$

  Avec Temps désigne la moyenne de la série temporelle formée par la valeur du temps de d'exécution et écart type désigne l'ecart type de la même série. Après avoir calculé ces deux bornes, nous vérifions si tous les échantillons se trouvent dans le niveau de confiance.
- Marge d'erreur calculée : La marge d'erreur calculée en raison de la dispersion statistique est calculée à l'aide de la formule (1).

Dans ce qui suit, nous discutons les résultats de la variation temporelle pour chaque algorithme e tri comme montre les figures 2.18 et 2.19.

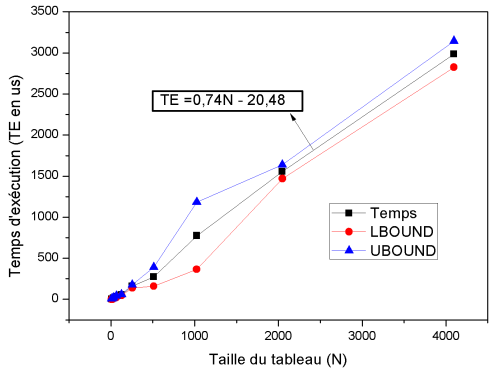


FIGURE 2.18 – MergeSort

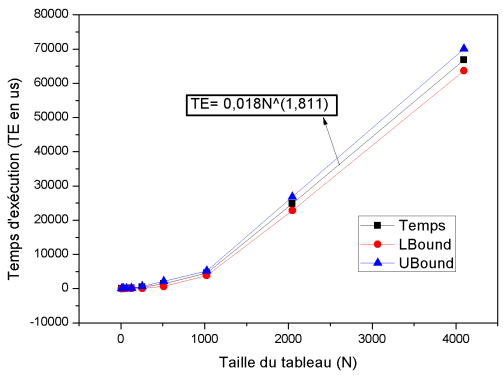


FIGURE 2.19 – InsertionSort

La Figure 2.18 présente la variation temporelle de l'algorithme MergeSort, LBound et UBound en fonction de N. Nous observons que la courbe du temps est presque linéaire. Par contre, les Figure 2.19 et 2.17 montrent que la fonction de l'algorithme InsertionSort est quadratique en termes de N et la valeur de la médiane reste constante. Dans ce cas

et à partir des Boxplots, nous pouvons donner une idée sur la stabilité des algorithmes MergeSort et InsertionSort.

2.6 Conclusion

Dans ce chapitre, nous avons détaillé les systèmes de transport intelligent. Parmi ces systèmes, nous avons cité les avantages des systèmes avioniques (planification du plan de vol, aide à la décision et les algorithmes de tri). Par la suite, nous avons effectué une étude bibliographique sur l'exécution de plusieurs algorithmes de tri sur plusieurs plateformes. Ensuite, nous avons présenté une vue globale sur les plateformes d'exécution (CPU, GPU et FPGA) et les algorithmes de tri (BubbleSort, InsertionSort, SelectionSort, HeapSort, QuickSort, ShellSort, MergeSort et TimSort). Dans ce chapitre, nous avons réalisé une étude préliminaire sur le processeur qui a montré que l'algorithme MergeSort est plus performant et stable en termes de temps d'exécution et de l'écart type si le nombre d'éléments est supérieur à 64. Sinon, InsertionSort est choisi comme le meilleur algorithme. Ceci est dû à la complexité des algorithmes de tri. Nous avons exprimé les résultats avec des Boxplots pour visualiser et comparer les différentes permutations. De ce fait, le chapitre suivant sera consacré pour l'étude et l'implémentation logicielle des algorithmes de tri sur la plateforme Zynq ZC 702.

Chapitre 3

Implémentation logicielle des algorithmes de tri sur la plateforme Zynq Zedboard

Sommaire

3.1 Introduction 45
3.2 Architecture du SoPC et méthodologie de conception 45
3.2.1 Généralités 45
3.2.2 Architecture du SoPC 46
3.3 Présentation de la carte Xilinx Zedboard 49
3.3.1 La partie programmable FPGA : Famille Xilinx Zynq 7000 . . 49
3.3.2 Système microprocesseur (PS) 53
3.3.3 Outils de synthèse et d'analyse de conceptions HDL 54
3.4 Implémentation logicielle des algorithmes de tri 55
3.4.1 Environnements de développement 55
3.4.2 Étude de la complexité des algorithmes de tri 57
3.5 Conclusion 68

3.1 Introduction

De nos jours, les systèmes embarqués intègrent des fonctionnalités spécifiques nécessitant une puissance de calcul qu'un seul processeur ne peut fournir. D'autre part, ils sont caractérisés par des ressources limitées et possèdent une capacité de communication limitée. Ils doivent aussi respecter des contraintes plus ou moins sévères tel que la consommation, la rapidité de fonctionnement, le coût, la taille, les performances,... En outre, il est important d'évoluer ces systèmes à supporter les contraintes extérieurs à titre d'exemple la température, l'humidité, vibrations, chocs, les variations d'alimentation, les interférences RF, les radiations, etc...Les applications temps réel embarquées doivent répondre aux évènements externes ou internes. Parmi ces applications, nous citons les systèmes de transports intelligents : automobile, aéronautique.

Dans ce chapitre, nous décrivons la conception logicielle des algorithmes de tri sur la plateforme Zedboard [96] à base d'un processeur ARM Cortex A9 [97]. Nous commençons tout d'abord par citer les caractéristiques de la carte Zedboard. Ensuite, nous détaillons les résultats de la validation des algorithmes de tri sous l'environnement *Standalone*. Finalement, nous concluons ce chapitre par une étude comparative entre ces différents algorithmes sur le processeur ARM Cortex A9 en considérant différents critères de performances (temps d'exécution, Écart type et la consommation d'énergie).

3.2 Architecture du SoPC et méthodologie de conception

3.2.1 Généralités

Les architectures SoPC reconfigurables (hardware et software) [98, 1] fournissent avec l'évolution des technologies sur FPGAs une amélioration importante de la flexibilité d'utilisation et de la minimisation des systèmes comme montre la Figure 3.1. La totalité des composants électroniques matériels et logiciels sont intégrés dans la même puce (ASIC ou FPGA). Pour cela, la complexité des SoPCs ne cesse pas de s'accroitre. D'où vient la notion d'utilisation de l'outil de conception HLS [56] qui permet de convertir la programmation haut niveau (C/C++, system C) en bas niveau (VHDL/Verilog). La conception de ces systèmes a besoin de créer une nouvelle méthodologie en considérant des spécifications logicielles et des accélérateurs matériels.

Dans cette section, nous présentons tout d'abord l'architecture hétérogène interne des SoPCs qui permet d'améliorer les performances (temps de conception, consommation d'énergie) dans la couche matérielle, la flexibilité et la facilité de programmation dans la couche logicielle. Ensuite, une présentation de la carte Xilinx Zedboard est nécessaire pour exécuter des implémentations logicielles et matérielle des algorithmes de tri(BubbleSort, InsertionSort, SelectionSort, HeapSort, QuickSort, ShellSort, MergeSort et TimSort). Finalement, nous présentons une étude comparative entre les différents algorithmes en termes du temps d'exécution.

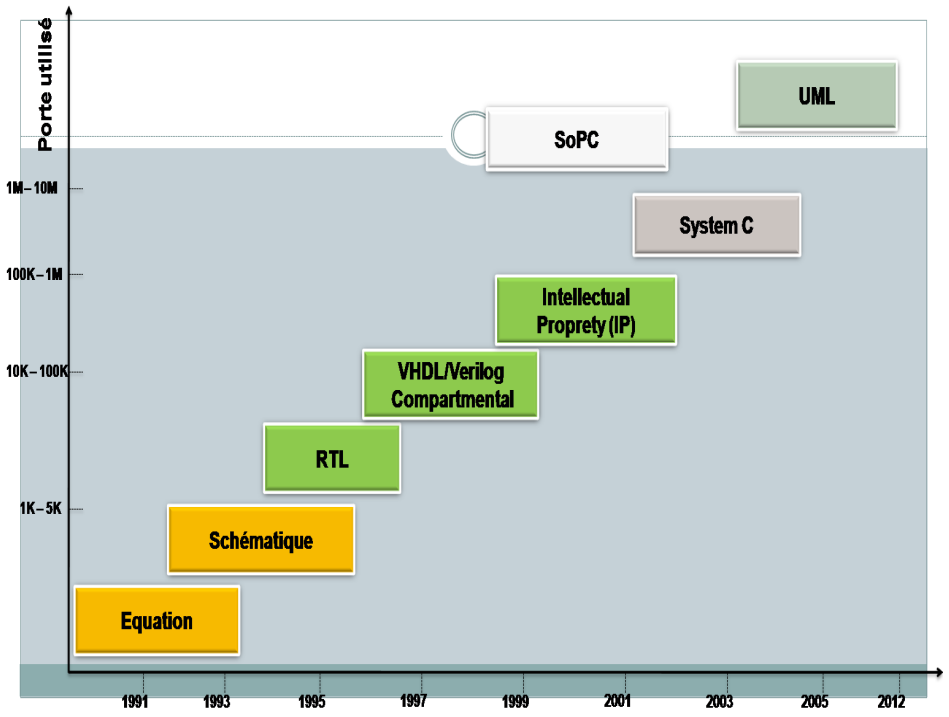


FIGURE 3.1 – Diagramme d'évolution des technologies de conception sur FPGA

3.2.2 Architecture du SoPC

Un SoPC [2] est une famille de circuit intégré fabriquée par Xilinx. Il intègre la totalité des composants sur le même circuit intégré reprogrammable de type FPGA. Les dispositifs SoPC ont principalement quatre blocs qui sont le bloc des processeurs embarqués (ARM, MIPS, PowerPC, ST-Micrœlectonics, etc.), des blocs mémoires, des blocs d'interfaçage et des accélérateurs matériels IP pour effectuer des traitements complexes sur une puce. Ces processeurs se différent des processeurs ordinaires par une architecture optimisée afin de réduire la consommation d'énergie. Ces quatre blocs sont interconnectés avec un réseau de communication. Les principaux avantages d'un SoPC sont la rapidité de programmation et la simplicité de reconfiguration.

Processeurs généraux pour les SoPCs

Nous pouvons distinguer deux différents types de processeurs pour les SoPC : des cœurs de processeur hard (hardcore) et des cœurs de processeurs soft (softcore) dont le choix se fait en fonction de leur nature.

(a) Le processeur softcore

Les processeurs soft (ou processeurs reconfigurable ou virtuels) sont des processeurs qui peuvent être implémentés sur un circuit reconfigurable (FPGA). Nous citons à titre d'exemple le processeur Microblaze de la famille Xilinx et NIOS de la famille Altera. Aussi, les processeurs ARM se sont des versions softcore de la famille ARM. Nous constatons qu'il y'a d'autres processeurs softcore librement disponibles comme Leon [99] et OpenRisc qui sont compatible avec les processeurs Microblaze [100] et Sparc. Les principaux avantages de ces types de processeurs sont l'intégration de la flexibilité dans leurs systèmes par rapport aux processeurs classiques et

la facilité d'intégration dans la technologie reconfigurable l'adaptation aux besoins applicatifs. Cette flexibilité est obtenue en utilisant des outils de programmation.

(b) Le processeur hardcore

Le processeur hard peut être un circuit intégré comprenant une partie reconfigurable et une partie non-configurable ou bien un circuit intégré dans un FPGA/-SoPC. Par contre, ce processeur peut être disponible sous forme de masque pour la technologie ASIC. L'utilisation des processeurs hard dans les circuits intégrés est un peu flexible. Les processeurs hard doivent être plus puissants, volumineux et consomment plus d'énergie qu'un processeur qui aurait été conçu spécifiquement pour l'application (ASIP).

Accélérateurs matériels

Les accélérateurs matériels [101] sont des fonctions de contrôle d'entrée/sortie (UART, VGA, PS2, etc) ou de traitement intensif (FFT, application avionique, algorithme de tri, etc.). Ce sont des composants matériels spécialisés pour ce but afin de faciliter le fonctionnement sur le processeur (ou microcontrôlleur). Ils permettent d'obtenir des résultats de haute performance. En effet, ces composants peuvent être des composants matériels conçus ou des propriétés intellectuelles (IPs) réutilisables. Une propriété intellectuelle se réfère à un produit inventé par une entreprise ou une personne pour lesquels des droits exclusifs sont distribués. Ces produits sont souvent optimaux et permettent d'accomplir certaines fonctions arithmétiques complexes telles que FIR [102], DCT [103], FFT [104], etc. La commercialisation des IPs débute depuis l'année 1990 dont les principaux fondateurs sont Synopsys et ARM. Il existe 2 types d'IP [105] :

(a) IP logiciel (Soft core)

IP soft est un processeur ou circuit programmable. Il est synthétisable et ne peut pas être prédictif en termes de consommation et de temps. Cet IP est portable et son code source peut être modifiable. Par contre, il ne donne pas des solutions optimisées.

(b) IP matériel (Hard core)

IP hard est un composant définit dans la génération de Bitstream. En effet, ces composants ne sont pas flexibles et utilisent une seule technologie. Ils augmentent les performances lors de la diminution de la consommation et du temps d'exécution.

Le flot de conception SW/HW avec le logiciel Vivadeo

Le flot de conception SW/HW [5, 6] sur FPGA est organisé en plusieurs étapes pour accélérer les différentes applications (système décisionnel en temps réel, planification de plan de vol en 3D) comme montre la figure 3.2.

Tout d'abord, on a une application logicielle divisée en plusieurs tâches développées en langage de description logicielle C/C++. Ces tâches doivent être exécutées sur une plateforme hétérogène CPU/FPGA. Cependant, nous devons optimiser l'application en utilisant différentes directives de l'outil HLS (voir en détail Figure 3.3) pour obtenir à la fin une implémentation hardware efficace.

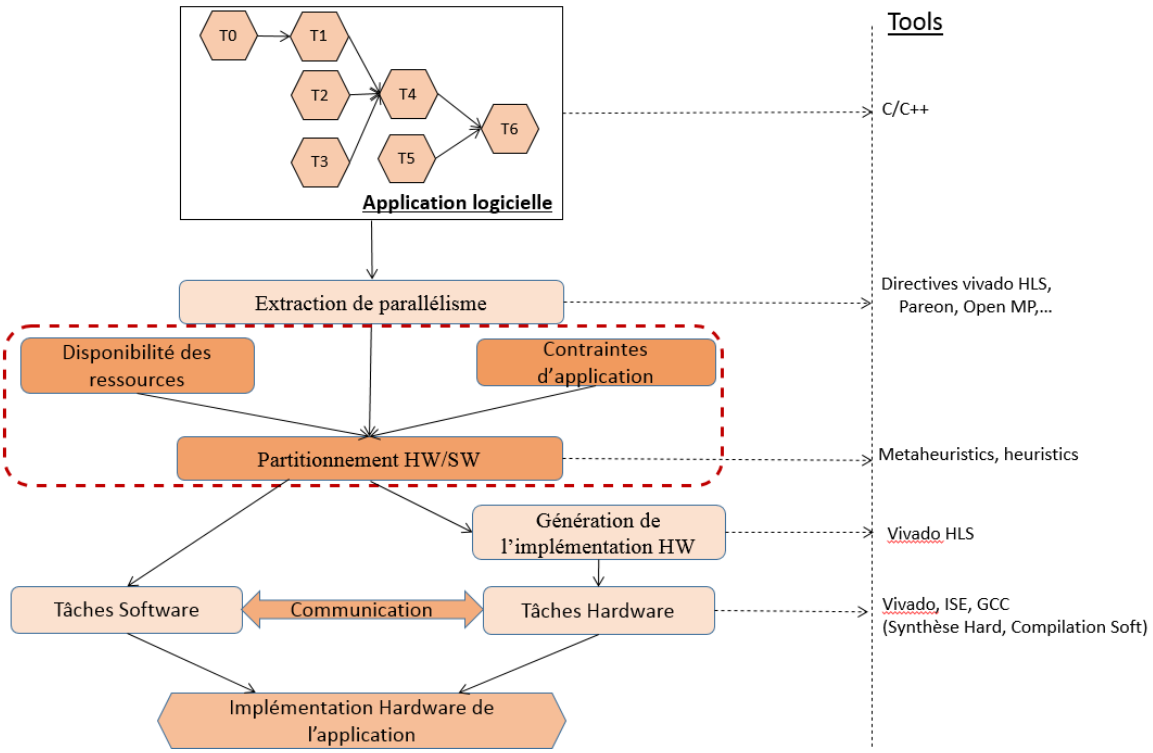


FIGURE 3.2 – Flot de conception SW/HW

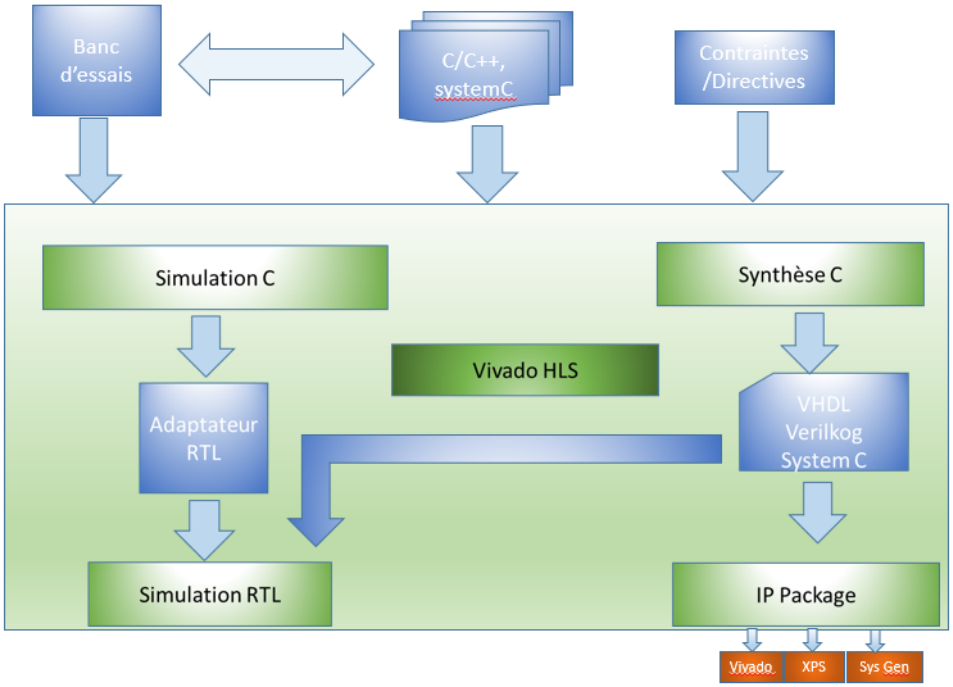


FIGURE 3.3 – Vue globale de Vivado HLS

Ensuite, l'algorithme optimisé sera diviser en deux tâches : logicielles et matérielles. Ce partitionnement se fait à l'aide des différentes méthodes d'ordonnancement MHEFT tout en respectant les contraintes de l'application et aussi la disponibilité des ressources. Cette étape est proposée dans la littérature [30]. La tâche hardware est générée à travers l'outil Vivado HLS [64] à partir du code C/C++. Enfin, les tâches software et hardware

interagissent ensemble en utilisant un bus de communication afin de trouver le résultat final. Cette communication est faite avec des outils ISE ou Vivado.

3.3 Présentation de la carte Xilinx Zedboard

Xilnix fournit la carte Zedboard dans le cadre de d'évaluation et de développement du SoC [106, 3] comme indique la Figure 3.4. Elle est divisée en deux parties combinatoire PS *(Processing System)* et PL *(Programmable Logic)*. La partie PS intègre le processeur ARM Cortex A9 qui communique avec la partie PL via des bus internes de communication.
Les fonctionnalités fournies par le ZedBoard sont :

- FPGA Xilinx XC7Z020-1CLG484C Zynq-7000 AP SoC
- 512 MB de mémoire DDR3 et 256 Mbit de QSPI Flash.
- 1 connecteur USB-JTAG Digilent.
- 1 connecteur Ethernet 10/100/1G.
- 1 connecteur USB OTG 2.0 et une interface pour la carte SD.
- 1 connecteur série (UART).
- 1 interface HDMI.
- 1 interface VGA.
- Fréquence PL 100 MHz et 33.33 MHz PS.
- 5 connecteur Digilent Pmod (2x6) (1 PS, 4 PL)
- 1 connecteur LPC FMC
- 1 connecteur AMS
- 2 bouttons Reset (1 PS, 1 PL)
- 7 bouttons poussoirs (2 PS, 5 PL)
- 8 dip/slide switches (PL)
- 9 LEDs (1 PS, 8 PL)
- DONE LED (PL)

3.3.1 La partie programmable FPGA : Famille Xilinx Zynq 7000

La famille Zynq-7000 (Figure 3.5) [107] est basée sur l'architecture SoC Xilinx All Programmable. Aussi, elle fait partie de la famille de série 7 des FPGA dont les performances basculent entre les deux familles Artix (7010, 7015 et 7020) et Kintex (7030, 7045 et 7100) (Figure 3.6). Il est caractérisé par 85 000 Slices (comprennent de la logique combinatoire et ressource de registre) ,560 KB bloc de RAM, avec 220 DSP Slices.

Cependant, Cette famille intègre un système de traitement (PS) basé sur ARM Cortex A9 à deux cœurs ou un seul cœur riche en fonctionnalités avec une logique programmable d'un FPGA Xilinx de 28 nm (PL) dans une seule puce. Par conséquent, elle offre une haute performance, une flexibilité, une scalabilité, une consommation d'énergie et une facilité d'utilisation généralement associé aux ASIC et aux ASSP.

(a) CLB/Slice

Dans la partie programmable du Zynq, il existe deux *Slices* par CLB *(Configurable Logic Block)* : les slices M (ROM, logique, RAM et registres à décalage), et les slices L (ROM et logique). La figure 3.7indique que chaque *Slice* contient 4 LUTs (*LookUp Table)* à 6 entrées et 8 bascules D *(Flip Flop)*.

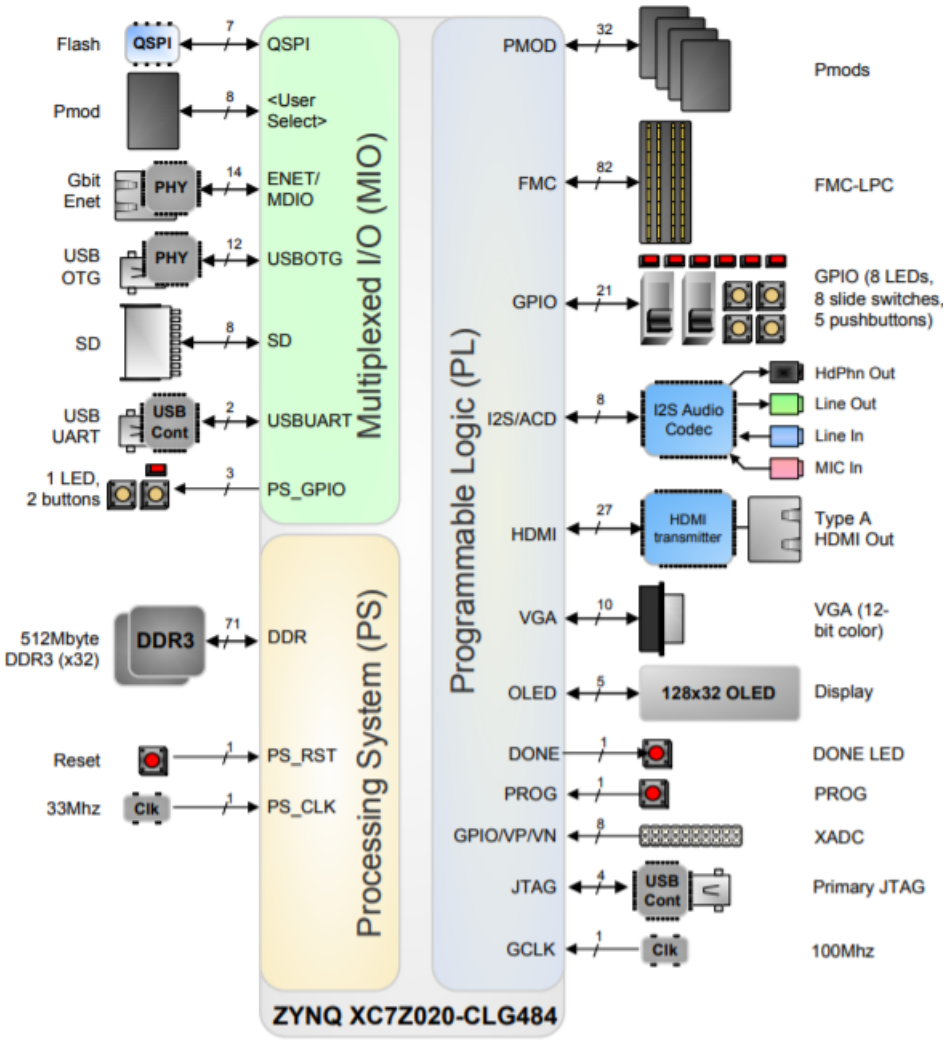


FIGURE 3.4 – Schéma synoptique de la carte Xilinx Zedboard

(b) Les blocs DSP

Les FPGAs implémentent des algorithmes parallèles et spécifiques. Pour cela, ils sont un système efficace pour les applications de traitement de signal numérique (DSP). La figure 3.8 montre l'architecture du bloc DSP qui contient de nombreux multiplicateurs et accumulateurs binaires dans un bloc MAC *(Multiply and ACcumulate)* qui est l'élément principal d'un bloc DSP. Tous les FPGAs de la série 7 ont de nombreuses tranches de DSP, spécifiques, de faible puissance et flexibles. Les blocs DSP améliorent les performances des applications de traitement de signal numérique en termes de la vitesse et de l'efficacité. Plus précisément, FPGA zynq 7000 intègres 220 blocs DSP pour des tâches de multiplication avec une entrée de 18x25 bits.

(c) Les blocs RAM

Les blocs RAM de FPGA sont de taille 36 kbits. Ces blocs peuvent être utilisés en deux blocs indépendants de taille 18 kbits. Ils possèdent un double port comme montre la Figure 3.9. Ces blocs sont aussi configurables en FIFO sans logique externe *(First in*

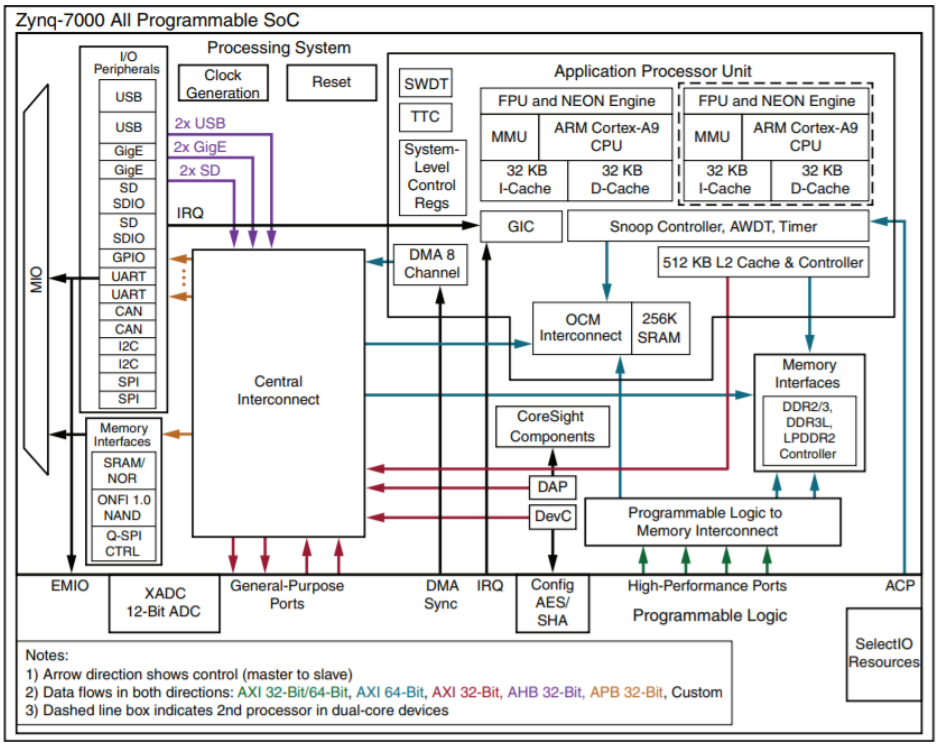


FIGURE 3.5 – Architecture de l'FPGA Zynq 7000

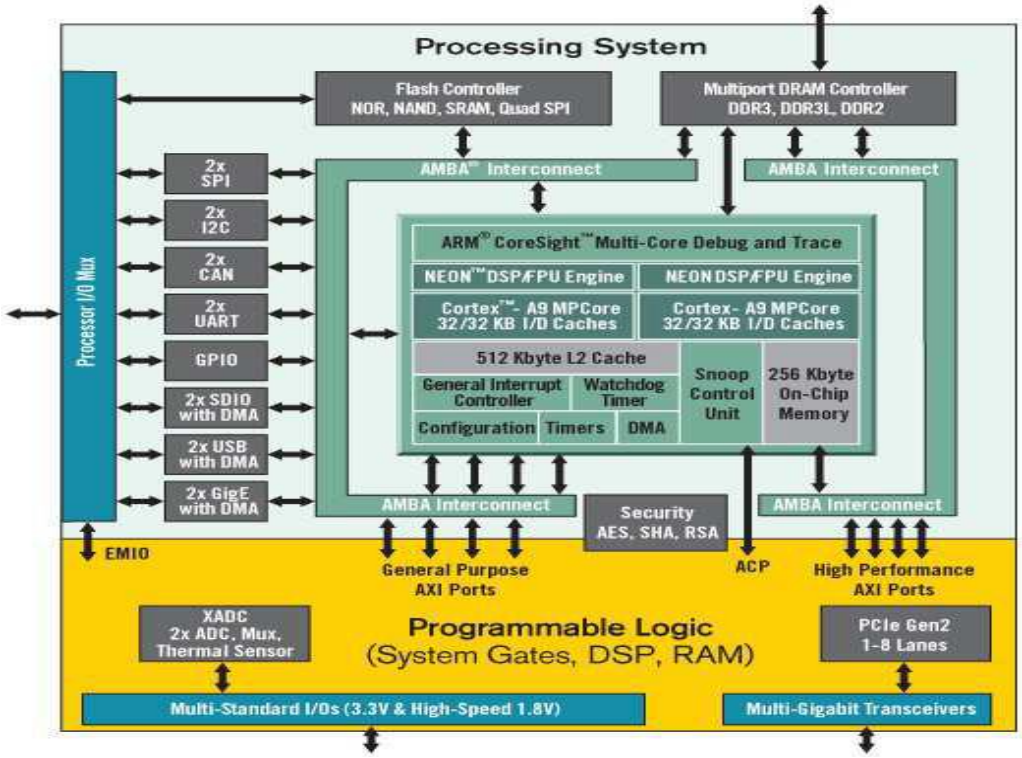


FIGURE 3.6 – Architecture de l'FPGA Zynq 7020

First out).

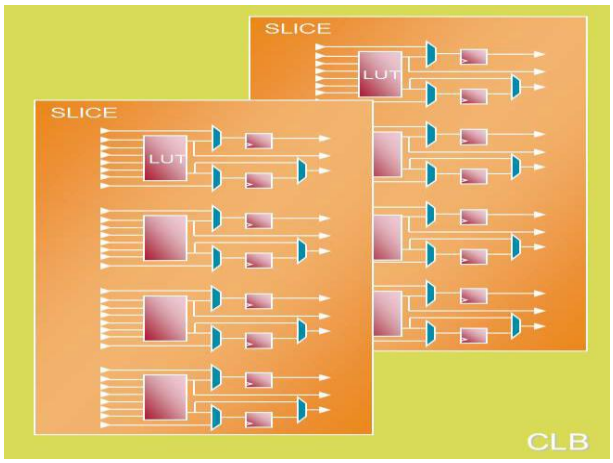


FIGURE 3.7 – Architecture du *Slice*

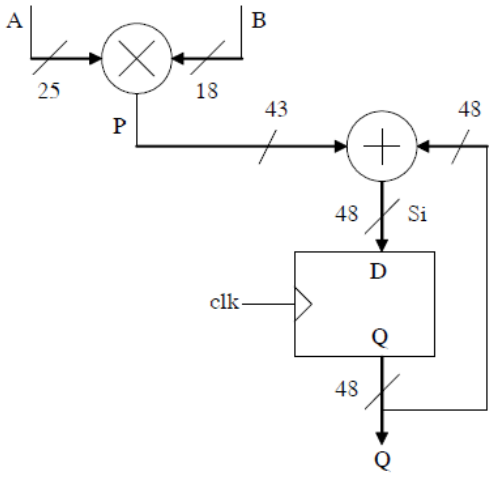


FIGURE 3.8 – Architecture du bloc DSP

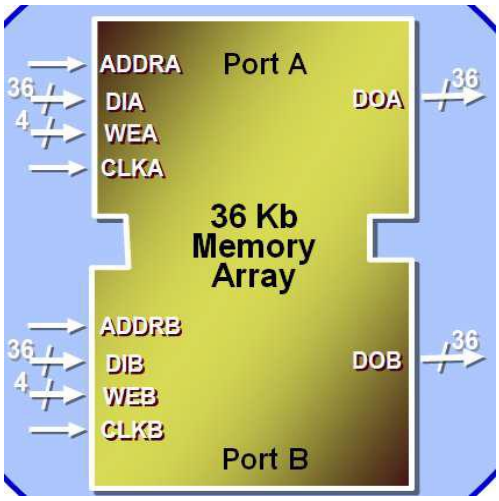


FIGURE 3.9 – Architecture du bloc RAM

3.3.2 Système microprocesseur (PS)

Les concepteurs ont intégrés dans les années 2000 des cœurs de processeurs matériels au sein des FPGAs (Par exemple ARM pour Altera ou Power PC pour Xilinx). Depuis 2005, les FPGAs (Virtex 5, 6, 7 chez Xilinx) ne contiennent pas des cœurs de processeurs. Par contre, l'intégration de processeurs apparaît de nouveau dans les générations de FPGAs de la série 7 proposées par Xilinx en 2012. La famille d'FPGA zynq se compose d'un ou plusieurs cœurs ARM qui intègre également une mémoire et un ensemble de périphériques d'E/S comme indique la Figure 3.10. Par conséquent, ARM Cortex A9 est un processeur hardcore performant qui peut fonctionner avec une fréquence maximal égal 667 MHz. Il possède une mémoire cache L1 de 32 KB sur les adresses et sur les données. Par la suite, une mémoire cache L2 de 512 KB.
Depuis les années 1996, L'Advanced Microcontroller Bus Architecture (AMBA) [108] est

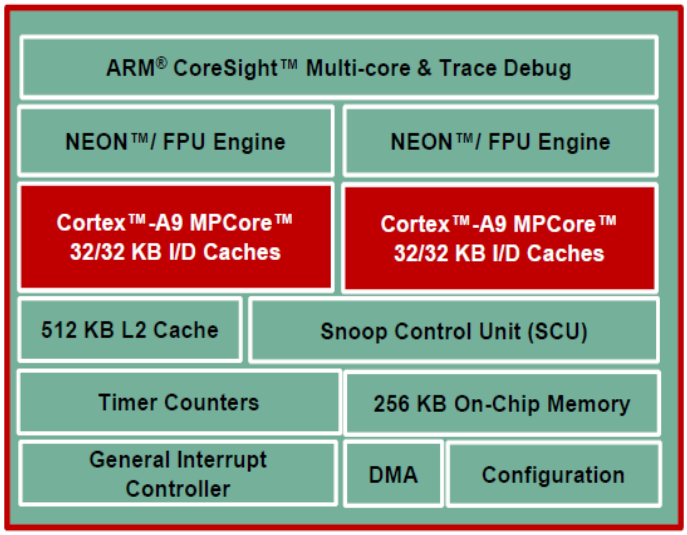


FIGURE 3.10 – La partie PS du Zynq 702

considéré dans les system-on-a-chip (SoC) comme un bus interne à base des processeurs ARM. Xilinx a adopté le protocole AXI (Advanced eXtensible Interface) pour les cœurs de propriété intellectuelle (IP) en commençant par les dispositifs Spartan-6 et Virtex-6. Par la suite, le protocole AXI est utilisé pour des accélérateurs matériels de la série 7. La première version d'AXI a été introduite dans AMBA 3.0, sortie en 2003. AMBA 4.0, sorti en 2010, inclut la seconde version d'AXI, AXI4. D'une part, AXI4 apporte des améliorations à l'ensemble des produits Xilinx en offrant des avantages en termes de productivité, de flexibilité et de disponibilité. D'autre part, il est issu de 2 principaux protocoles de communication : AX4-Lite et AXI4-Stream [109].

AXI4-Lite

AXI4-Lite est une interface de communication mappée en mémoire unique et légère. Il est considéré comme une interface simple pour la conception et l'utilisation. La communication entre PS et PL est effectuée en utilisant 4 ports AXI4-Lite appelés "Gerneral-Purpose" (GP) à 32 bits, deux maîtres et deux esclaves. A chaque transaction, si l'un entre eux prend en charge la fonction maitre alors il déclenche et contrôle le traitement

séquentiel des données codées sur 32 bits. La Figure 3.11 présente la communication des données entre PS et PL en utilisant l'interface AXI-Lite.

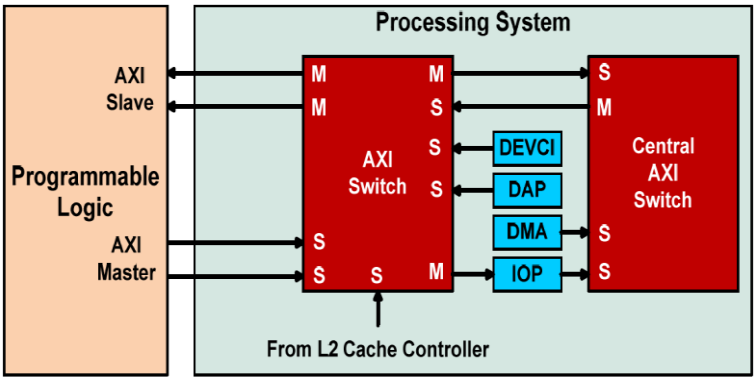


FIGURE 3.11 – Communication entre PS et PL avec le bus AXI4-Lite

AXI4-Stream

AXI Stream est l'un des protocole AMBA conçu pour transporter des flux de données de largeur arbitraire de taille 32/64 bits dans le matériel. Ces données sont généralement représentées sous forme de données vectorielles côté logiciel qu'on peut transférer 4 octets par cycle. Typiquement, l'interface AXI4-Stream est utilisée avec un contrôleur DMA *(Direct Memory Access)* pour transférer une grande partie de données du processeur au FPGA. Le contrôleur DMA lit les données vectorielles de la mémoire et les transmet à IP FPGA via l'interface AXI4-Stream. L'IP de FPGA peut contenir une interface AXI4-Lite pour le réglage des paramètres et les signaux de contrôle. L'interface AXI4-Stream transfère les données beaucoup plus rapidement que l'interface AXI4-Lite, ce qui la rend plus adaptée au chemin de données d'un algorithme (Figure 3.12).

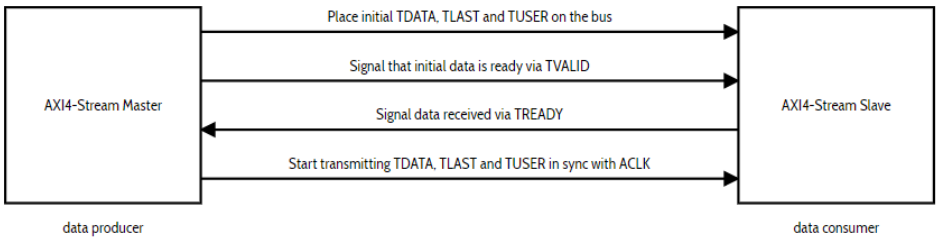


FIGURE 3.12 – Communication entre PS et PL avec le bus AXI4-Stream

3.3.3 Outils de synthèse et d'analyse de conceptions HDL

ISE (Integrated Synthesis Environment)

Xilinx ISE est un outil logiciel produit par la société Xilinx pour la synthèse et l'analyse de conceptions HDL, permettant au développeur de synthétiser ("compiler") leurs conceptions, d'effectuer une analyse de synchronisation, d'examiner des diagrammes RTL,

de simuler la réaction d'une conception à différents stimuli et de configurer le périphérique cible avec le programmeur. Xilinx ISE est un environnement de conception pour les produits FPGA de Xilinx et il est étroitement lié à l'architecture de telles puces. Il ne peut pas être utilisé avec des produits FPGA d'autres fournisseurs. Le Xilinx ISE est principalement utilisé pour la synthèse et la conception de circuits, tandis que ISIM ou le simulateur logique ModelSim sont utilisés pour les tests au niveau du système. Les autres composants livrés avec Xilinx ISE incluent le kit de développement embarqué (EDK : Embedded Development Kit) qui est une suite d'outils et d'IP que vous pouvez utiliser pour concevoir un système de processeur embarqué complet à implémenter dans un périphérique FPGA Xilinx, un kit de développement logiciel (SDK : Software Development Kit) qui est un environnement de développement intégré, complémentaire de XPS (l'environnement de développement utilisé pour concevoir la partie matérielle de votre système de processeur intégré), utilisé pour la création d'applications logicielles intégrées C / C ++.

PlanAhead

PlanAhead est un logiciel de conception et d'analyse utilisé pour concevoir la conception de FPGA. Il fournit un environnement intégré et intuitif pour l'ensemble du processus de mise en œuvre du FPGA. Il permet d'améliorer les performances du circuit en analysant les sources RTL de conception, les listes de liens synthétisées et les résultats de mise en œuvre.

Vivado Design Suite

Vivado Design Suite est une suite logicielle produite par Xilinx pour la synthèse et l'analyse des conceptions HDL, remplaçant Xilinx ISE avec des fonctionnalités supplémentaires pour le développement de système sur puce et la synthèse de haut niveau. Il fournit un flux de conception centré sur IP qui vous permet d'ajouter des modules IP à votre conception à partir de diverses sources de conception.

3.4 Implémentation logicielle des algorithmes de tri

Dans cette partie, nous calculons la complexité des algorithmes de tri sur le processeur ARM Cortex A9 en termes de temps d'exécution et écart type. Pour cette raison, nous allons détaillé les différents environnements de développement existante : environnement Linux embarqué et *Standalone*.

3.4.1 Environnements de développement

Linux embarqué

Un système d'exploitation est un logiciel qui gère les dispositifs matériels dans un circuit électronique et traite des instructions complexes demandé par l'utilisateur ou d'autres logiciels (ou applications). Il offre ainsi différents services pour mieux trouver la complexité

des systèmes embarqués : Il est un système multitâche puisque plusieurs tâches peuvent être exécuté simultanément. Parmi ces systèmes, nous citons Linux embarqué. Le développement et l'implémentation d'un système Linux est caractérisé par plusieurs avantages qui sont détaillés ci-dessous :

- Facilitation du test des nouvelles fonctionnalités : Un système Linux permet facilement d'obtenir un logiciel et de l'évaluer. Il permet d'étudier plusieurs options. Par conséquent, il est trop facile et moins cher que d'acheter ou d'utiliser des versions d'essai propriétaire.
- Faible coût : C'est l'un des principaux avantages. Il est un logiciel libre disponible gratuitement au niveau source qui peut être dupliqué sur autant de composants sans frais.
- Réutilisation de composants et de codes : D'une part, Linux fournit un nombre important des composants et des codes prédéfinissent tels que bibliothèque, multimédia, graphique, protocole, etc. D'autre part, il permet de développer rapidement des produits complexes basés sur des composants disponibles. Pour cela, il n'est pas obligatoire de redévelopper le même code par différents développeurs.
- Haute qualité : Un grand nombre de développeurs et utilisateurs partagent leurs connaissances et leurs codes.
- Un système RTOS (Real Time Operating System)
- Contrôle totale sur le logiciel : Le développeur peut accéder au code source de tous les composants ce qui permet des modifications et optimisation illimités sans verrouillage de la part du fournisseur.

Linux embarqué a plusieurs avantages mais aussi il a des inconvénients :

- Manque de drivers pour un périphérique spécifique.
- Une lente compilation du noyau Linux et une haute complexité d'apprentissage pour le développement.
- Système instable en cas de mal génération

Le mode Standalone

Le mode *Standalone* permet aux utilisateurs de compiler simplement leurs programmes et de les exécuter discrètement sur le matériel sans besoin d'un système d'exploitation. Ce mode est supporté pour les plateformes suivantes :

– Zc702
– Zc706
– Zedboard
– microzed
– Zybo

Au paravent, avec les anciens outils PlanAhead et ISE (Integrated Synthesis Environment), l'utilisation de Linux embarqué est obligatoire pour valider des applications complexes qui exigent l'usage d'un système de fichier pour la gestion de la lecture et l'écriture d'une ou plusieurs séquences d'images. Pour cette raison, le choix des applications compatibles avec ce mode d'exécution est limité. Par ailleurs, avec l'émergence des nouvelles versions de Vivado et plus précisément à partir de la version 14.4, on est aujourd'hui capable de travailler en mode Standalone sans avoir besoin d'utiliser un système d'exploitation. En effet, des nouvelles bibliothèques ont été ajoutées dans le support de bibliothèque de SDK pour gérer les opérations de lecture et écriture dans un fichier.

Pour cela, plusieurs librairies ont été ajoutées dans l'outil SDK pour la gestion des fichiers "ff.h" (appelée "Fat File System"). Pour l'ouverture, l'écriture et lecture dans un fichier, il faut utiliser les fonctions mentionnées dans la Figure 3.13.

Overture des fichiers :

```
fopen (
 FIL* fS,            /* Pointeur sur fichier */
  const TCHAR* path,    /* nom du fichier*/
  BYTE open_mode         /*  mode d'ouverture ( read ou write)*/
);
```

Lecture des fichiers :

```
TCHAR* f_gets (
TCHAR* buffer, /* pointeur sur la chaine de caractères */
int  length,    /* taille de la chaine de caractères */
FIL* fs     /* pointeur sur le fichier */
);
```

Ecriture dans un fichier :

```
FRESULT f_write (
 FIL* fp,        /* Pointeur sur le fichier */
const void* buffer     /* Pointeur sur les données écrites */
UINT btr,       /* Nombre de byts écrits */
 UINT* bt        /* Pointeur sur la variable qui retourne le nombre de caractères écrits */
);
```

FIGURE 3.13 – Les fonctions définies dans la bibliothèque *"Fat File System"*

3.4.2 Étude de la complexité des algorithmes de tri

La complexité des algorithmes de tri sur la plateforme Zedboard a été réalisée comme celles calculé dans le chapitre 2. En fait, nous nous sommes intéressés par huit algorithmes de tri : BubbleSort, InsertionSort, SelectionSort, HeapSort, ShellSort, QuickSort, MergeSort et TimSort en utilisant des données codées en entrée sur 32 bits. Dans cette partie, nous utilisons seulement le mode *Standalone* afin de stocker les données dans un fichier d'entrée. Ce mode est simple à utiliser et ne nécessite pas un OS. A partir de cette étude, nous comparons le temps d'exécution, écart type et la consommation d'énergie des algorithmes sur la plateforme Zedboard afin d'intégrer le meilleur algorithme dans les applications avioniques. Pour la mesure de la consommation, nous avons connecté le module "USB Interface Adapter" de Texas Instruments à la carte Zedboard comme le montre la Figure 3.14. Pour le calcul de temps d'exécution et l'écart type, nous avons exécuté chaque algorithme sur la partie logiciel plus précisément sur le processeur ARM Cortex A9 en considérant plusieurs tailles d'éléments 8/16/32/64/128/256/512/1024/2048/4096. Aussi, nous avons fixé le nombre de permutation à 47 par algorithme et le nombre de réplications à 1000 par permutation. Les résultats obtenus sur le temps d'exécution sont détaillés dans la table 3.1 :

Tableau 3.1 – Temps d'exécution des algorithmes de tri sur ARM Cortex A9

	BubbleSort (us)			InsertionSort			SelectionSort			HeapSort			ShellSort			Quicksort			MergeSort			TimSort	
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
8	2.98	3.07	3.59	2.24	2.28	3.23	2.01	2.05	2.73	2.64	2.72	4.63	2.07	2.13	3.45	3.81	3.87	5.15	3.24	3.29	4.91	1.14	1.18	2.31
16	11.01	11.02	11.59	7.29	7.33	8.24	6.71	6.75	7.61	6.6	6.71	9.63	5.57	5.63	7.24	10.34	10.41	11.92	7.74	7.79	9.57	4.05	4.09	5.07
32	42.19	42.22	42.71	25.28	25.33	26.28	22.46	22.51	23.69	17.24	17.36	22.03	13.37	13.43	15.12	31.88	31.94	33.27	17.86	17.94	19.64	14.29	14.37	15.42
64	165.07	165.13	165.78	98.83	98.9	99.76	84.53	84.58	85.75	44.22	44.4	50.07	31.49	31.55	33.17	105.69	105.76	107.22	40.56	40.67	42.76	45.85	46.32	56.89
128	572.01	572.03	573.14	280.29	280.33	281.65	311.14	311.31	314.29	107.72	107.96	114.95	76.5	76.59	78.92	164	164.24	166.69	93.43	93.5	96.46	102.38	103.04	114.01
256	2129.29	2129.33	2129.84	930.51	930.55	931.89	1187.88	1187.94	1189.55	255.56	255.76	263.83	168.48	168.58	171.14	266.19	266.49	270.02	207.71	207.84	212.36	219.95	220.07	228.44
512	7934.49	7934.96	7935.81	3319.82	3320.21	3321.55	4639.45	4639.84	4641.48	583.16	583.33	591.55	437.29	437.38	439.59	499.95	500.2	503.56	536.14	536.57	539.28	552.62	553.82	565.81
1024	7647.36	7647.39	7648.2	12734.21	12735	12736.26	18376.52	18376.58	18378.96	1323.48	1323.75	1332.57	1028.69	1029.52	1032.58	1213.67	1214.4	1226.63	1198.27	1198.48	1207.75	1202.41	1205.97	1220
2048	130468.98	130478.5	130488.95	4931.71	49316.29	49318.57	72987	72989.01	73007.75	2948.67	2949.01	2958.12	2105.04	2116.45	2116.83	3490.58	3499.45	3510.1	2628.83	2629.1	2635.11	2623.61	2628.01	2642.43
4096	518665.34	518691.28	518702.72	194007.48	194020.42	194021.36	290803.88	290836.91	2908540.72	6449.33	6451.24	6455.06	3272.06	3291.38	3304.15	11838.28	11843.88	11873.29	5672.75	5684.51	5696.79	5703.5	5741.88	5752

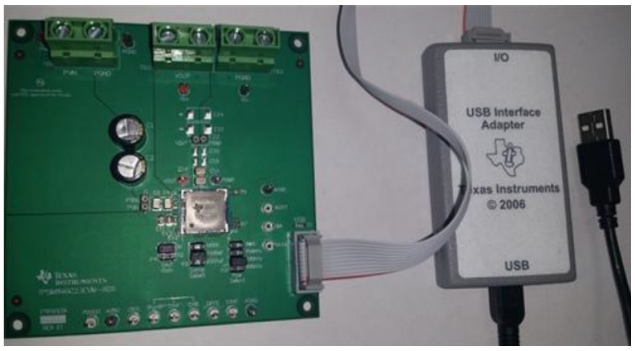


FIGURE 3.14 – Module de la consommation d'énergie

Tout d'abord, nous considérons dans ce travail que les permutations sont générés à l'aide de la méthode de Lehmer. Pour cela, les résultats fournis dans le tableau 3.1 présente le minimum, la moyenne et le maximum du temps d'exécution de tous les algorithmes de tri sur le processeur ARM Cortex A9 en utilisant différents tailles d'éléments allant de 8 à 4096 et un nombre important de réplications (R=1000). Ensuite, la Figure

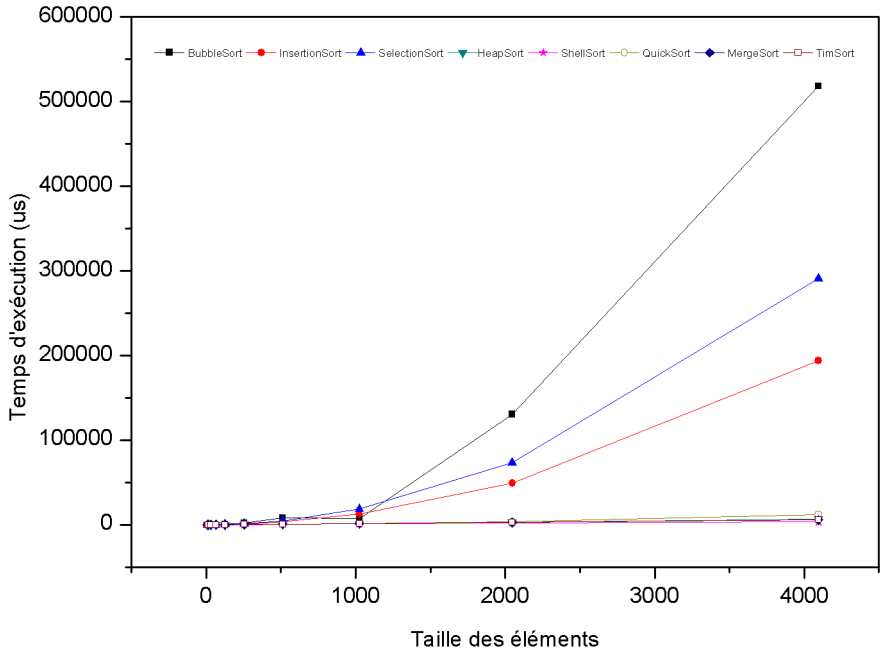


FIGURE 3.15 – Temps d'exécution des algorithmes de tri exécutés sur le processeur

3.15 présente en moyenne le temps d'exécution en microseconde des différents algorithmes (BubbleSort, InsertionSort, SelectionSort, HeapSort, ShellSort, QuickSort, MergeSort et TimSort). Par exemple, si N=4096, le temps d'exécution est 518691us, 194007.48 us, 290836.91 us, 11843.88 us, 6451.24 us, 5752 us, 5684.51 us et 3291.33us pour BubbleSort, InsertionSort, SelectionSort, QuickSort, HeapSort, TimSort, MergeSort et ShellSort respectivement. Finalement, nous pouvons conclure que si le nombre de données est important alors les algorithmes BubbleSort, InsertionSort et SelectionSort consomment beaucoup de temps. Pour cela, nous n'avons pas pris en considération ces algorithmes pour

choisir l'algorithme la plus performant en termes de temps, d'écart type et de la consommation. Dans ce cas, nous allons s'intéresser seulement sur les cinq autres algorithmes si le nombre de données est supérieur à 64. La figure 3.16 montre que l'algorithme TimSort et

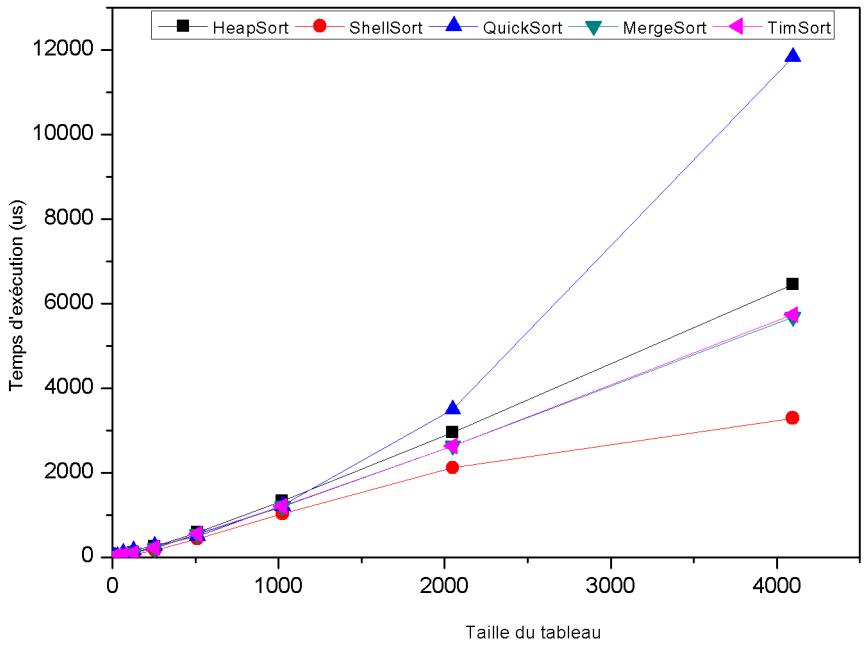


FIGURE 3.16 – Zoom des cinq algorithmes de tri

MergeSort donne presque la même valeur du temps si la taille du tableau est important. De plus, l'algorithme QuickSort possède un temps d'exécution très important par rapport aux autres. Par la suite, nous remarquons que ShellSort est plus rapide de 42.1% jusqu'à 72% en exécutons sur le processeur ARM Cortex A9 si N>64 sinon l'algorithme TimSort est le plus performant. De point de vue performance, nous avons réfléchi à ajouter un autre critère de performance pour comparer ces algorithmes en termes de l'écart type sur la durée d'exécution. D'une part, Les tableaux 3.2 et 3.3 présentent les résultats obtenus des écarts type pour les algorithmes de tri dans les deux cas (N<=64 et N>64). Pour cela, nous calculons la valeur de l'écart type pour les différents algorithmes de tri pour un nombre d'éléments trop petit (<64) et nous détaillons dans cette partie seulement les résultats obtenus de HeapSort, ShellSort, MergeSort et TimSort pour N>64. Nous constatons que la valeur de l'écart type varie entre 0 et 79.514. Table 3.2 montre la difficulté de choisir un meilleur algorithme parmi les huit algorithmes en termes de l'écart type si le nombre d'élément est inférieur à 64. Par contre, il est clair que l'algorithme ShellSort a un meilleur écart type par rapport aux autres. Nous prenons par exemple pour N=2048, l'écart type égale à 62.612 pour HeapSort, 71.61 pour MergeSort, 50.121 pour TimSOrt et 49.94 pour ShellSort.

D'autre part, les valeurs mesurées pour la consommation d'énergie demeurent presque constantes en passant d'un algorithme de tri à une autre et en variant la taille du tableau. Nous calculons la valeur de la tension (V) et l'intensité (A) pour les algorithmes de tri. La Figure 3.17 donne les résultats de la puissance et de la tension pour un exemple d'algorithme MergeSort.

Tableau 3.2 – Écart type pour les algorithmes de tri N<=64

	BubbleSort	Insertion	Selection	Quicksort	Heapsort	ShellSort	MergeSort	TimSort
8	0.145	0	0.41	0	0.369	0.05	0	0.05
16	0.256	0.202	0.271	0.307	0.625	0.06	0.021	0.08
32	0.214	0.339	0.176	0.196	0.671	0.175	1.65	0.619
64	0.255	0	2.145	0.107	2.194	0.180	2.55	3.28

Tableau 3.3 – Ecart type pour les meilleurs algorithmes de tri N>64

	Heapsort	MergeSort	ShellSort	TimSort
128	4.28	4.069	7.15	7.4815
256	6.405	8.1	11.05	14.388
512	14.18	36.26	11.083	21.7
1024	35.88	32.83	28.86	30.69
2048	62.612	71.61	49.94	50.121
4096	79.514	60.08	59.6	67.45

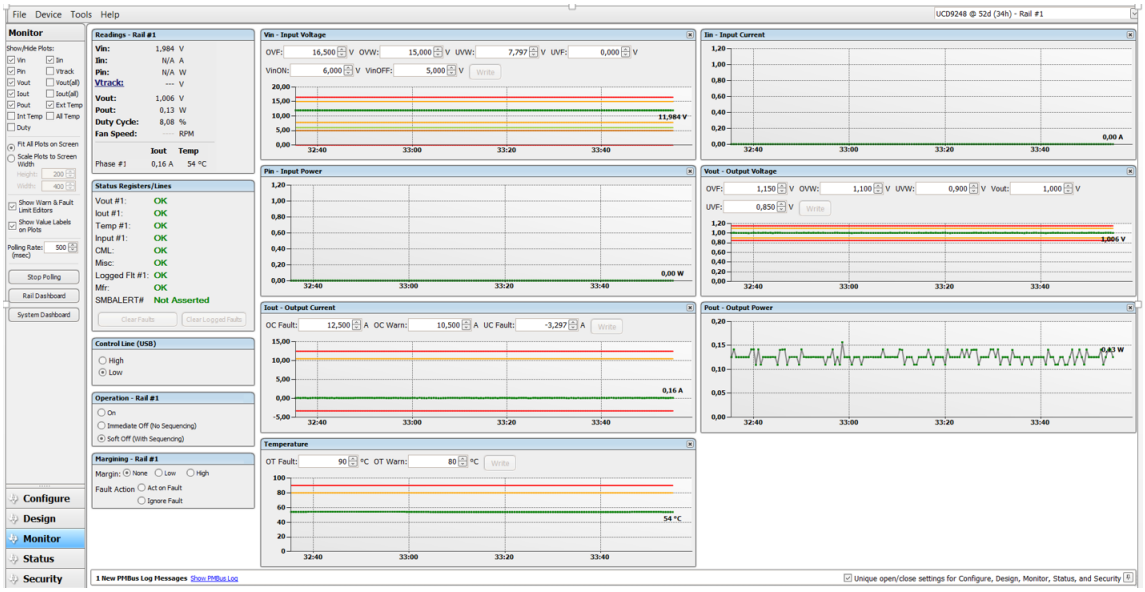


FIGURE 3.17 – Interface de sortie de la consommation de puissance

En se référant aux résultats illustrés dans la Figure 3.17, nous constatons que la valeur de la puissance est presque constantes pour tous les algorithmes de tri (P=0.12W). Donc, nous pouvons conclure que si l'algorithme est de haute performance en termes de temps d'exécution et d'écart type alors il est considérée comme le meilleur en termes de la consommation d'énergie qui est calculée à l'aide de l'équation suivante :

$$E(J) = Puissance(W) * Temps(s) \tag{3.1}$$

Tableau 3.5 – Coefficient de variation pour les algorithmes de tri N<=64

	BubbleSort	Insertion	Selection	Quicksort	Heapsort	ShellSort	MergeSort	TimSort
8	4.733	0	20	0	13.566	2.34	0	1.77
16	2.32	2.755	4.0414	2.94	9.31	1.06	0.256	1.026
32	0.5	1.339	0.78	0.613	3.86	1.07	9.19	1.33
64	0.15	0	2.537	0.1	4.94	0.57	6.26	8.064

Tableau 3.6 – Coefficient de variation pour les meilleurs algorithmes de tri N>64

	Heapsort	MergeSort	ShellSort	TimSort
128	3.96	4.35	9.335	7.26
256	2.5	3.897	1.25	6.534
512	2.43	6.758	2.533	3.91
1024	2.71	2.73	2.803	2.544
2048	2.2	2.723	2.35	1.9
4096	1.23	1.05	1.8	1.17

Tableau 3.4 – Consommation d'énergie des algorithmes de tri

	BubbleSort (uJ)	InsertionSort (uJ)	SelectionSort (uJ)	HeapSort (uJ)	ShellSort (uJ)	QuickSort (uJ)	MergeSort(uJ)	TimSort (uJ)
8	0.3684	0.27	0.246	0.3264	0.2556	0.4644	0.394	0.1416
16	1.3224	0.88	0.81	0.8052	0.6756	1.2492	0.9348	0.4908
32	5.066	3.0396	2.71	2.0832	1.7316	3.8328	2.1528	1.724
64	19.8156	11.86	10.149	5.328	3.786	12.6912	4.884	5.558
128	68.6436	33.639	37.3572	12.955	9.1908	19.706	11.22	12.364
256	255.51	111.666	142.55	30.6912	20.229	31.978	24.84	26.408
512	952.195	398.425	556.78	70	52.4856	60.024	64.388	66.458
1024	917.68	1528.2	2205.189	158.85	123.542	145.728	143.8176	144.716
2048	15657.42	5917.95	8758.68	353.88	253.97	419.934	315.492	315.36
4096	62242.953	23282.45	34900.42	774.14	394.96	1421.2656	682.1412	689.025

La Table 3.4 indique la consommation d'énergie pour les différents algorithmes de tri. Nous remarquons que ShellSort consomme moins d'énergie lorsque le nombre d'éléments est supérieure à 64. À partir des Tables 3.1 et 3.4 et si le nombre de données $N \geq 64$, nous pouvons conclure que ShellSort est l'algorithme le plus performant en termes de temps d'exécution, d'écart type et de la consommation d'énergie en l'exécutant sur le processeur ARM Cortex A9. Sinon, TimSort est choisi.

A partir des tableaux 3.5et3.6 et en se référant au chapitre 2, nous remarquons que le coefficient de variation pour l'algorithme ShellSort, heapsort, MergeSort et TimSort est supérieure à 1 si N > 64. Dans ce cas, nous pouvons conclure expérimentalement qu'il y'a une variation significatif et les boites à moustaches(Boxplots) sont utiles pour donner une idée sur la dispersion des données. Nous ne pouvons pas mettre tous les Boites dans ce chapitre. Pour cela, nous proposons uniquement de tracer les boites à moustaches (Boxplots) de l'algorithme TimSort avec un nombre d'éléments allant de 128 jusqu'à 4096. La Figure 3.18 montre une représentation graphique des différents distributions avec des boites à moustache en considérant 1000 réplications et 47 permutations pour chaque N>64. Le Tableau 3.7 présentent les valeurs de la moyenne des temps d'exécution, minimum, Q1, de la médiane (Q2), Q3, maximum et IQR des 47 permutations pour N>64 pour l'algorithme TimSort. Nous remarquons que les valeurs ne sont pas centralisé par la médiane.

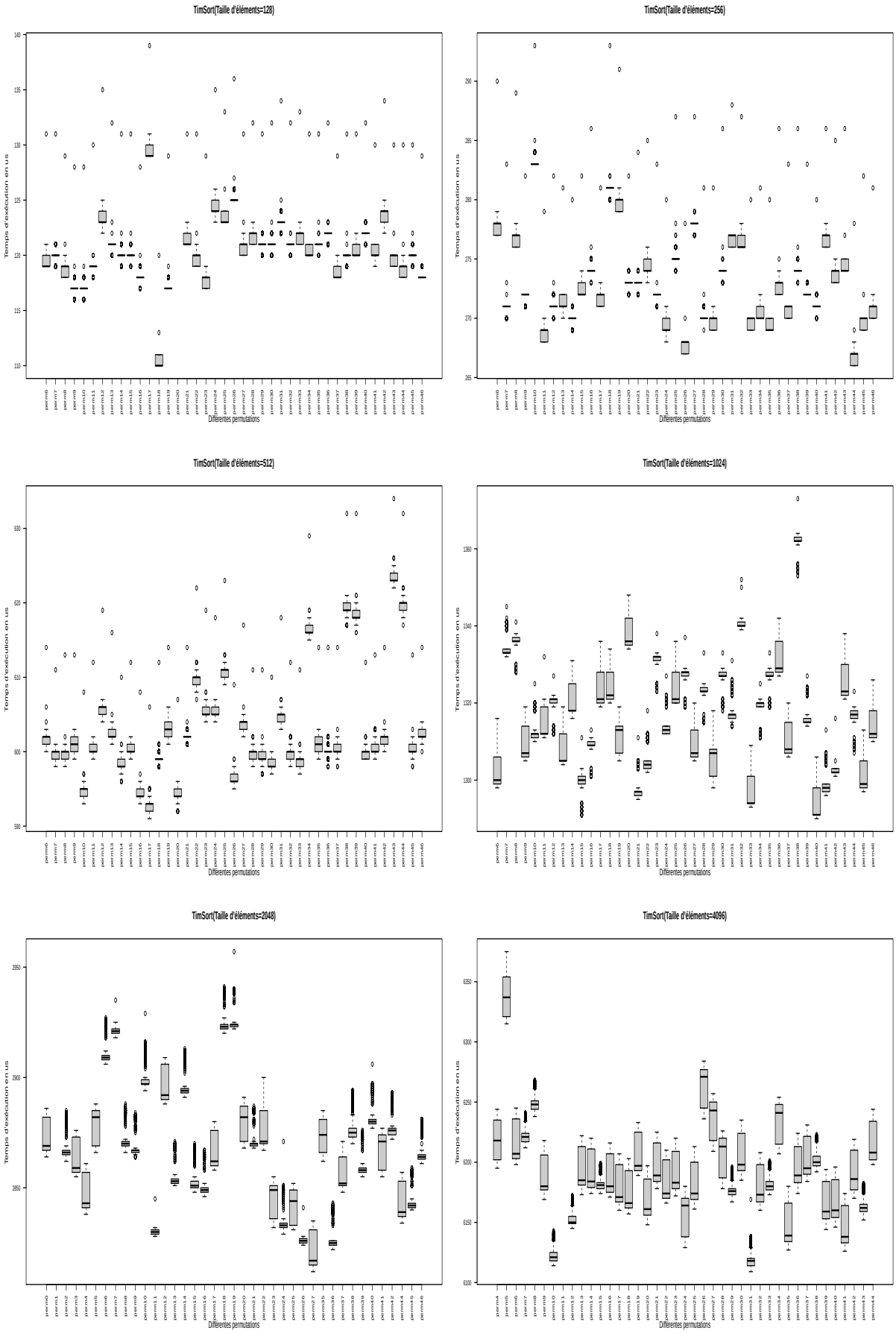


FIGURE 3.18 – Boxplots de l'algorithme TimSort sur le processeur ARM

Tableau 3.7 – Critères de performance pour l'algorithme TimSort N>64

	128							256							512							1024							2048							4096						
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	17	17	17.2		17	20	0	33	35	35	36	35	36	0	68	68	68	68.5	69	71	1	128	138	138	137.24	138	138	0	2864	2867	2869	2875.02	2882	2886	15	624	658	658	657.32	658	659	0
perm1	23	23	23	23.38	24	27	1	46	47	47	47.5	48	49	1	93	94	94	94	94	97	0	179	188	188	187.68	188	189	0	2846	2848	2849	2850.42	2851	2868	3	745	780	781	796.1	798	801	18
perm2	20	20	20	20.12	20	23	0	39	41	41	42	41	43	0	81	81	81	82	82	84	1	154	163	163	162.5	163	163	0	2862	2865	2866	2868.04	2867	2885	2	948	982	982	980.9	983	985	1
perm3	24	25	25	25.02	25	30	0	48	50	50	50	50	52	0	97	98	98	98	98	102	0	192	193	193	193.12	193	195	0	2855	2857	2859	2866.08	2873	2876	16	6195	6202	6218	6218.14	6235	6244	33
perm4	31	31	31	31.1	31	35	0	60	62	62	62	62	66	0	121	122	122	122.5	123	126	1	236	236	236	236.38	236	248	0	2838	2841	2843	2849.04	2857	2861	16	6195	6202	6218	6218.14	6235	6244	33
perm5	20	20	20	20.22	20	24	0	39	41	41	41	41	43	0	81	81	81	81.5	82	85	1	154	163	163	161.9	163	164	0	2866	2869	2882	2877.74	2885	2888	16	6315	6321	6357	6337.34	6354	6375	33
perm6	119	119	119	119.58	120	131	1	277	277	278	277.82	278	290	1	600	601	602	602.26	602	614	1	1298	1299	1300	1301.74	1306	1316	7	2906	2908	2909	2911.52	2910	2927	2	6198	6203	6207	6215.84	6236	6245	33
perm7	119	120	120	120.22	120	131	0	270	271	271	271.1	271	283	0	598	599	600	599.88	600	611	1	1332	1333	1333	1334.64	1334	1345	1	2918	2920	2921	2921.04	2922	2935	2	6212	6217	6221	6221.94	6224	6241	7
perm8	118	118	119	118.8	119	129	1	276	276	277	276.88	277	289	1	598	599	600	599.88	600	613	1	1328	1336	1336	1336.34	1337	1341	1	2866	2869	2870	2871.42	2871	2888	2	6238	6244	6248	6246.94	6251	6268	7
perm9	116	117	117	117.3	117	128	0	271	272	272	272.02	272	282	0	599	600	601	601	602	613	2	1305	1306	1307	1309.64	1314	1319	8	2864	2866	2867	2867.5	2867	2884	1	6169	6177	6180	6185.44	6206	6218	29
perm10	116	117	117	117.24	117	128	0	283	283	283	283.4	283	293	0	593	594	595	594.92	595	608	1	1310	1311	1312	1312.32	1312	1325	1	2894	2897	2897	2900.3	2899	2929	2	6114	6118	6121	6122.88	6125	6143	7
perm11	118	119	119	119.2	119	130	0	268	268	269	268.76	269	279	1	599	600	600	600.44	601	612	1	1311	1312	1312	1314.9	1319	1332	7	2828	2829	2830	2830.2	2831	2845	2	623	659	659	656.94	659	660	0
perm12	122	123	123	123.54	124	135	1	270	271	271	271.42	271	282	0	604	605	606	605.82	606	619	1	1312	1320	1321	1319.98	1321	1327	1	2888	2890	2892	2896.94	2906	2909	16	6145	6149	6150	6152.5	6155	6172	6
perm13	120	121	121	121.24	121	132	0	270	271	271	271.66	272	281	1	601	602	602	602.74	603	616	1	1304	1305	1305	1306.9	1312	1319	7	2851	2852	2853	2855.64	2854	2871	2	6173	6181	6185	6187.8	6213	6222	32
perm14	119	120	120	120.24	120	131	0	269	270	270	270.14	270	280	0	596	598	598	598.94	599	610	1	1316	1318	1318	1320.68	1325	1331	7	2891	2893	2894	2898.3	2895	2913	2	6174	6179	6184	6189.7	6211	6220	32
perm15	119	120	120	120.26	120	131	0	272	272	272	272.54	273	282	1	599	600	600	600.4	601	612	1	1291	1299	1300	1299.94	1301	1311	2	2848	2850	2851	2853.86	2853	2870	3	6174	6178	6181	6181.68	6183	6199	5
perm16	117	118	118	118.4	118	128	0	273	274	274	274.3	274	286	0	593	594	594	594.82	595	608	1	1301	1309	1309	1307.5	1310	1313	1	2846	2848	2848	2850.84	2850	2867	2	6171	6175	6180	6188.34	6207	6216	32
perm17	129	129	129	129.72	130	139	1	271	271	272	271.9	272	281	1	591	592	593	593.24	593	606	1	1319	1320	1321	1324.26	1328	1336	8	2858	2860	2862	2866.84	2876	2880	16	6160	6167	6171	6180.92	6198	6207	31
perm18	110	110	110	110.68	111	120	1	280	281	281	281.22	281	283	0	598	599	599	599.28	599	612	0	1320	1321	1322	1323.72	1328	1334	7	2920	2922	2923	2923.56	2924	2941	2	6157	6162	6166	6172.88	6193	6203	31
perm19	117	117	117	117.42	117	129	0	279	279	280	279.64	280	291	1	601	602	603	603.22	604	614	2	1305	1307	1313	1310.02	1314	1319	7	2922	2923	2924	2924.74	2924	2957	1	6189	6193	6197	6206.34	6225	6233	32
perm20	120	121	121	118	121	131	0	272	273	273	273.04	273	282	0	592	594	594	594.42	595	607	1	1334	1335	1336	1337.9	1342	1348	7	2868	2871	2882	2879.38	2887	2891	16	6148	6156	6161	6166	6186	6197	30
perm21	121	121	121	121.72	122	131	1	272	273	273	273.04	273	284	0	601	602	602	602.42	602	614	0	1295	1296	1297	1297.36	1297	1311	1	2868	2869	2870	2871.34	2870	2887	1	6178	6184	6189	6194.4	6216	6225	32
perm22	119	119	120	119.92	120	131	1	273	274	274	274.64	275	285	1	607	609	610	610.1	610	622	1	1302	1303	1304	1305.34	1305	1318	2	2867	2870	2876.68	2885	2885	2900	15	6166	6170	6174	6180.9	6202	6210	32
perm23	117	117	118	117.78	118	129	1	271	272	272	272.14	272	283	0	604	605	605	605.74	606	619	1	1323	1331	1332	1331.9	1332	1338	1	2832	2836	2840	2844.98	2851	2855	15	6172	6178	6183	6193.3	6209	6220	31
perm24	123	124	124	124.58	125	135	1	268	269	269	269.64	270	280	1	604	605	605	605.78	606	618	1	1312	1312	1313	1314.7	1314	1327	2	2829	2832	2833	2835.3	2834	2871	2	6129	6138	6164	6153.32	6170	6180	32
perm25	123	123	123	123.52	124	133	1	274	275	275	275.38	275	287	0	609	610	611	610.98	611	623	1	1320	1320	1321	1322.86	1328	1336	8	2831	2833	2844	2840.9	2849	2852	16	6161	6169	6174	6182.26	6200	6213	31
perm26	125	125	125	125.54	125	136	0	267	267	268	267.9	268	278	1	595	596	596	596.66	597	609	1	1319	1327	1328	1327.22	1328	1337	1	2824	2825	2826	2825.98	2827	2841	2	6236	6245	6271	6262.98	6277	6284	32
perm27	120	120	121	121	121	131	1	277	278	278	278.06	278	287	0	602	603	604	604.1	604	617	1	1305	1306	1307	1309.16	1313	1320	7	2812	2815	2817	2823	2831	2835	16	6209	6218	6243	6231.06	6250	6257	32
perm28	121	121	121	121.84	122	132	1	277	278	279	278.38	279	287	0	598	599	600	599.94	600	611	1	1315	1323	1323	1322.84	1324	1333	1	259	260	272	268.76	275	277	15	6178	6187	6213	6202.6	6220	6226	33
perm29	120	121	121	121.16	121	131	0	269	269	270	269.96	270	281	1	597	599	599	599.56	600	611	1	1298	1301	1307	1305.64	1308	1318	7	361	361	373	368.02	376	376	15	6167	6173	6176	6177.1	6178	6196	5
perm30	120	121	121	121.22449	121	132	0	273	274	274	274.3	274	286	0	597	598	598	598.68	599	610	1	1318	1327	1326.82	1328	1333	1	310	310	319	318.78	325	327	15	6185	6193	6198	6202.86	6224	6235	31
perm31	122	123	123	123.4	123	134	0	276	276	277	276.54	277	288	1	603	604	605	605.28	605	618	1	1314	1316	1317	1317.44	1317	1331	1	374	379	379	379.28	379	385	0	6109	6114	6118	6118.36	6120	6169	6
perm32	120	121	121	120.98	121	132	0	276	276	276	276.56	277	287	1	598	599	600	600.34	600	612	1	1339	1340	1340	1341.36	1341	1352	1	470	477	483	481.64	486	496	9	6160	6167	6173	6175.48	6198	6308	31
perm33	121	121	122	121.7	122	133	1	269	269	270	269.94	270	280	1	597	598	599	599.1	599	611	1	1293	1294	1294	1297.24	1301	1309	7	308	311	323	319.12	326	326	15	6173	6177	6180	6181.52	6184	6201	7
perm34	120	120	120	120.58	121	131	1	270	270	270	269.82	271	281	1	615	616	616	616.62	617	629	1	1311	1319	1320	1319.04	1320	1325	1	1047	1296.75	15.46.5	2021.5	1796.25	20.46	499.5	6207	6215	6241	6232.58	6248	6254	33
perm35	120	121	121	121.28	121	131	0	269	269	269	269.58	270	280	1	599	600	601	601.08	602	614	2	1319	1327	1327	1327.24	1328	1333	1	2862	2866	2874	2874.14	2881	2885	15	6127	6134	6139	6152.4	6166	6180	32
perm36	121	122	122	122.2	122	132	0	272	272	273	273.26	273	286	1	598	600	600	600.28	600	614	0	1327	1328	1329	1331.64	1336	1342	8	2822	2824	2825	2827.28	2826	2843	2	6174	6183	6189	6194.16	6213	6224	30
perm37	118	118	118	118.64	119	129	1	270	270	271	270.92	271	283	1	598	600	600	600.44	601	614	1	1306	1307	1308	1310.66	1315	1320	8	2848	2851	2852	2857.1	2864	2871	13	6184	6190	6195	6205.88	6221	6231	31.25
perm38	119	120	120	120.24	120	131	0	273	274	274	274.54	274	286	0	617	619	619	619.34	620	632	1	1353	1362	1362	1362.38	1363	1373	1	2870	2873	2875	2876.26	2877	2894	4	6192	6197	6200	6203.36	6205	6223	8
perm39	120	120	120	120.7	121	131	1	272	272	272	272.36	272	283	0	616	618	618	618.28	619	632	1	1314	1315	1315	1316.4	1316	1327	1	2855	2857	2858	2860.7	2859	2876	2	6144	6153	6159	6160.7	6184	6194	31
perm40	121	122	122	122.14	122	132	0	270	271	271	271.32	271	280	0	599	599	600	600.22	600	612	1	1290	1291	1291	1294.04	1298	1306	7	2877	2879	2880.84	2881	2881	2896	2	6146	6154	6160	6164.04	6185.25	6196	31.25
perm41	119	120	120	120.56	121	130	1	276	276	277	276.82	277	286	1	599	600	600	600.76	601	613	1	1296	1297	1298	1299.92	1299	1313	2	2855	2858	2871	2865.48	2874	2877	16	6126	6133	6138	6144.9	6164	6174	31
perm42	122	123	124	123.96	124	134	1	273	273	274	273.92	274	285	1	600	601	602	601.82	602	614	1	1301	1302	1302	1302.52	1303	1316	1	2872	2874	2876	2876.54	2877	2893	3	6170	6177	6186	6199.52	6210	6219	33
perm43	119	119	120	119.96	120	130	1	274	274	274	274.56	275	286	1	622	623	623	623.68	624	634	1	1321	1322	1323	1325.28	1330	1338	8	2869	2871	2872	2876.54	2843.84	2874	2901	3	6152	6159	6162	6163.26	6165	6182	6
perm44	118	118	119	118.94	119	130	1	266	266	267	266.98	267	278	1	617	619	620	620.08	620	632	1	1307	1316	1317	1316.3	1318	1323	2	2834	2837	2839	2843.3	2853	2857	16	6198	6202	6208	6220.68	6234	6244	32
perm45	119	120	120	120.28	120	130	0	269	269	270	270.04	270	282	1	598	600	600	600.34	601	613	1	1297	1298	1299	1300.4	1305	1313	7	2840	2841	2842	2864.86	2843	2859	2	525	557	557	556.1	558	558	1
perm46	118	118	118	118.36	118	129	0	270	270	271	271	271	281	1	600	602	602	602.38	603	614	1	1310	1311	1312	1312.86	1318	1326	7	2861	2863	2864	2865	2861	2881	2	722	759	760	757.28	760	761	1

Tableau 3.8 – Pourcentage d'outliers pour TimSort sur ARM

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
TimSort	128	6.33	3.23
	256	4.3	2.44
	512	3.17	1.58
	1024	7.22	3.2
	2048	5.6	5
	4096	2.66	1.98

Par la suite, La figure 3.18 illustre un nombre important de données aberrante dûe aux erreurs de transmission ou bruit généré par SE. Pour cela, le tableau 3.8 montre le pourcentage des outliers pour l'algorithme TimSort sur le processeur ARM de FPGA. Nous constatons que la valeur de la pourcentage d'outliers de moustache supérieur est inférieur à %5.c'est un nombre négligeable.

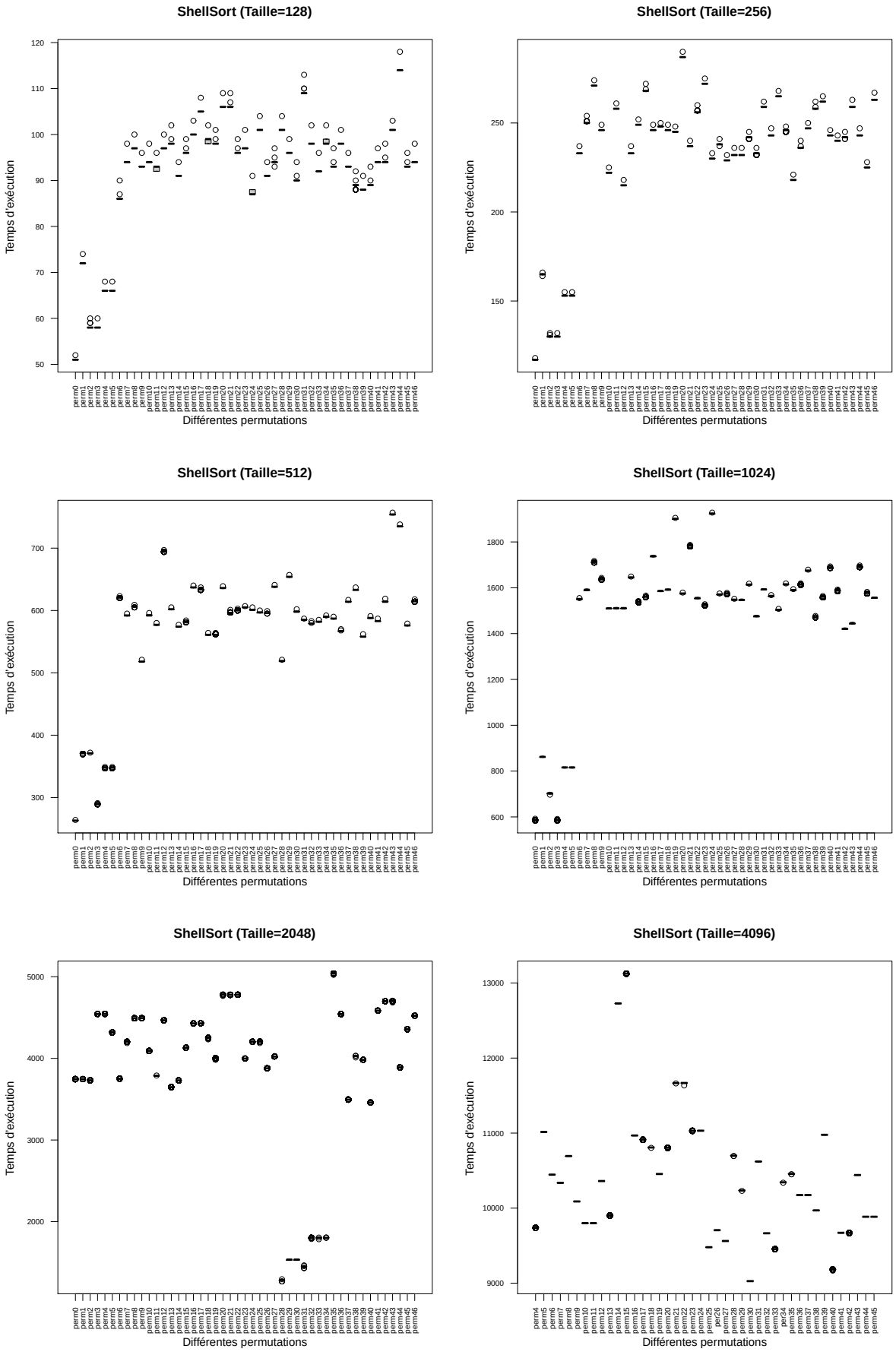


FIGURE 3.19 – Boxplots de l'algorithme ShellSort sur le processeur ARM

Tableau 3.9 – Boxplots de l'algorithme ShellSort sur le processeur ARM

	128						256						512						1024						2048						4096					
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	51	51	51	51,001	51	52	0	117	117	117	117,001	117	118	0	263	263	263	263,001	263	264	0	584	587	587	586,771	587	592	0	3744	3745	3745	3745,058	3745	3750	0	9650	9800	9738	9737,536	9801	9900	1
perm1	72	72	72	72,002	72	74	0	164	165	165	165	165	166	0	369	371	371	370,967	371	371	0	860	861	862	862,011	863	865	2	3745	3745	3745	3745,156	3745	3750	0	10600	10680	11015	11014,973	10682	11018	2
perm2	58	58	58	58,004	58	60	0	130	130	130	130,003	130	132	0	371	371	371	371,001	371	372	0	697	701	703	702,361	703	705	2	3730	3731	3731	3731,005	3731	3735	0	10000	10100	10446	10445,985	10102	10449	2
perm3	58	58	58	58,002	58	60	0	130	130	130	130,002	130	132	0	289	290	290	289,99	290	291	0	584	587	587	586,917	587	591	0	4541	4543	4543	4543,113	4543	4546	0	9800	9808	10337	10336,677	9900	10340	2
perm4	66	66	66	66,002	66	68	0	153	153	153	153,002	153	155	0	347	348	348	347,976	348	349	0	813	814	816	815,142	816	818	2	4541	4543	4543	4543,113	4543	4546	0	9734	9737	9738	9737,536	9738	9741	1
perm5	66	66	66	66,002	66	68	0	153	153	153	153,002	153	155	0	347	348	348	347,963	348	349	0	813	814	816	815,2	816	818	2	4316	4318	4318	4317,794	4318	4323	0	11011	11014	11015	11014,973	11016	11018	2
perm6	86	86	86	86,005	86	90	0	233	233	233	233,004	233	237	0	620	621	621	620,931	621	623	0	1548	1549	1551	1550,204	1551	1555	2	3749	3750	3750	3750,129	3750	3754	0	10442	10445	10446	10445,985	10447	10449	2
perm7	94	94	94	94,004	94	98	0	250	250	250	250,01	250	254	0	592	592	592	592,003	592	595	0	1588	1588	1591	1590,073	1591	1595	3	4187	4204	4204	4203,921	4204	4208	0	10333	10336	10337	10336,677	10338	10340	2
perm8	97	97	97	97,003	97	100	0	271	271	271	271,003	271	274	0	605	606	606	605,997	606	609	0	1710	1712	1713	1712,525	1713	1717	1	4490	4492	4492	4491,954	4492	4498	0	10690	10693	10694	10694,268	10695	10698	2
perm9	93	93	93	93,003	93	96	0	246	246	246	246,003	246	249	0	518	518	518	518,003	518	521	0	1635	1638	1639	1638,124	1639	1643	1	4491	4493	4493	4492,834	4493	4496	0	10085	10088	10089	10089,254	10090	10093	2
perm10	94	94	94	94,004	94	98	0	222	222	222	222,003	222	225	0	592	592	592	592,304	593	596	1	1507	1508	1510	1509,157	1510	1513	2	4089	4091	4091	4090,863	4091	4095	0	9796	9799	9800	9799,857	9801	9803	2
perm11	92	92	93	92,746	93	96	1	258	258	258	258,003	258	261	0	577	577	577	577,003	577	580	0	1508	1509	1511	1510,174	1511	1514	2	3785	3786	3787	3786,657	3787	3790	1	9796	9799	9800	9799,857	9801	9803	2
perm12	97	97	97	97,003	97	100	0	215	215	215	215,003	215	218	0	694	695	695	694,972	695	697	0	1508	1509	1511	1510,135	1511	1514	2	4464	4466	4466	4466,069	4466	4470	0	10358	10360	10361	10361,026	10362	10364	2
perm13	98	98	98	98,005	98	102	0	233	233	233	233,004	233	237	0	602	602	602	602,003	602	605	0	1642	1643	1645	1644,179	1645	1649	2	3646	3647	3647	3647,219	3647	3652	0	9896	9900	9901	9900,478	9901	9904	1
perm14	91	91	91	91,003	91	94	0	249	249	249	249,003	249	252	0	574	574	574	574,003	574	577	0	1536	1538	1538	1537,727	1538	1542	0	3729	3730	3730	3730,011	3730	3735	0	12724	12727	12728	12728,044	12729	12732	2
perm15	96	96	96	96,004	96	99	0	268	268	268	268,005	268	272	0	581	582	582	581,989	582	584	0	1559	1561	1562	1561,961	1562	1566	1	4129	4130	4130	4130,043	4130	4135	0	13122	13125	13126	13125,471	13126	13128	1
perm16	100	100	100	100,003	100	103	0	246	246	246	246,003	246	249	0	637	637	637	637,003	637	640	0	1734	1735	1738	1736,911	1738	1741	3	4428	4428	4428	4428,187	4428	4433	0	10964	10967	10968	10967,995	10969	10972	2
perm17	105	105	105	105,003	105	108	0	248	248	248	248,002	248	250	0	633	634	634	633,947	634	637	0	1583	1583	1586	1584,919	1586	1589	3	4428	4428	4428	4428,186	4428	4433	0	10908	10912	10913	10912,499	10913	10915	1
perm18	98	98	99	98,634	99	102	1	246	246	246	246,003	246	249	0	561	561	561	561,003	561	564	0	1589	1590	1592	1591,172	1592	1595	2	4234	4250	4250	4250,218	4250	4255	0	10803	10807	10808	10808,085	10809	10811	2
perm19	98	98	98	98,004	98	101	0	245	245	245	245,003	245	248	0	561	561	561	561,146	561	564	0	1898	1899	1901	1900,159	1901	1905	2	3985	4002	4002	4001,815	4002	4006	0	10451	10454	10455	10455,081	10456	10458	2
perm20	106	106	106	106,003	106	109	0	287	287	287	287,003	287	290	0	636	636	636	636,003	636	639	0	1573	1573	1575	1574,35	1575	1579	2	4765	4779	4779	4778,833	4779	4782	0	10800	10804	10805	10804,479	10805	10808	1
perm21	106	106	106	106,004	106	109	0	237	237	237	237,003	237	240	0	596	596	596	596,169	596	601	0	1781	1783	1784	1783,374	1784	1787	1	4777	4778	4778	4777,993	4778	4783	0	11662	11666	11667	11667,163	11668	11670	2
perm22	96	96	96	96,004	96	99	0	256	256	256	256,01	256	260	0	600	601	601	600,847	601	603	0	1552	1552	1555	1553,848	1555	1558	3	4778	4779	4779	4779,029	4779	4783	0	11635	11667	11668	11667,764	11669	11671	2
perm23	97	97	97	97,004	97	101	0	272	272	272	272,003	272	275	0	605	605	605	605,002	605	607	0	1522	1524	1525	1524,483	1525	1528	1	3997	3997	3997	3997,137	3997	4002	0	11026	11030	11030	11030,416	11031	11034	1
perm24	87	87	87	87,351	88	91	1	230	230	230	230,003	230	233	0	601	601	601	601,273	602	605	1	1921	1922	1924	1923,341	1924	1928	2	4203	4204	4204	4204,07	4204	4209	0	11028	11031	11032	11032,048	11033	11035	2
perm25	101	101	101	101,003	101	104	0	237	238	238	238,002	238	241	0	596	596	597	596,653	597	600	1	1568	1569	1571	1570,192	1571	1575	2	4187	4204	4204	4203,886	4204	4209	0	9475	9478	9479	9478,999	9480	9482	2
perm26	91	91	91	91,003	91	94	0	229	229	229	229,003	229	232	0	595	596	596	596	596	599	0	1572	1574	1575	1574,509	1575	1578	1	3877	3878	3878	3878,1	3878	3883	0	9702	9705	9706	9706,226	9707	9710	2
perm27	93	94	94	94,003	94	97	0	232	232	232	232,004	232	236	0	638	638	638	638,003	638	641	0	1545	1546	1548	1547,182	1548	1552	2	4022	4022	4022	4022,186	4022	4028	0	9558	9561	9562	9561,89	9563	9565	2
perm28	101	101	101	101,003	101	104	0	232	232	232	232,004	232	236	0	519	519	519	519,002	519	521	0	1544	1545	1547	1546,737	1548	1551	3	1268	1283	1285	1284,49	1285	1295	2	10694	10698	10699	10698,663	10700	10702	2
perm29	96	96	96	96,003	96	99	0	241	242	242	241,995	242	245	0	654	654	654	654,003	654	657	0	1611	1612	1614	1613,228	1614	1618	2	1526	1528	1536	1533,274	1536	1538	8	10230	10234	10235	10234,712	10236	10238	2
perm30	90	90	90	90,005	90	94	0	232	233	233	232,99	233	236	0	598	598	598	598,345	599	602	1	1473	1473	1476	1475,02	1476	1480	3	1526	1528	1536	1538,492	1536	1542	8	9023	9027	9028	9028,896	9029	9030	2
perm31	109	109	109	109,006	109	113	0	259	259	259	259,003	259	262	0	585	585	585	585,002	585	587	0	1590	1591	1593	1592,263	1593	1596	2	1432	1448	1449	1448,361	1449	1465	1	10617	10620	10621	10621,3	10622	10625	2
perm32	98	98	98	98,004	98	102	0	243	243	243	243,004	243	247	0	579	579	579	579,005	579	583	0	1562	1562	1564	1563,325	1564	1568	2	1788	1802	1802	1801,804	1802	1804	0	9661	9664	9665	9664,851	9666	9668	2
perm33	92	92	92	92,004	92	96	0	265	265	265	265,003	265	268	0	582	582	582	582,003	582	585	0	1501	1502	1504	1503,145	1504	1508	2	1785	1801	1802	1801,863	1802	1805	1	9449	9453	9454	9453,471	9454	9457	1
perm34	98	98	98	98,341	99	102	1	245	246	246	245,942	246	248	0	589	589	590	589,73	590	592	1	1612	1613	1615	1614,195	1615	1619	2	1801	1801	1802	1801,844	1802	1805	1	10339	10343	10344	10344,036	10345	10348	2
perm35	93	93	93	93,005	93	97	0	218	218	218	218,003	218	221	0	587	587	587	587,003	587	590	0	1587	1588	1590	1589,208	1590	1594	2	5023	5038	5038	5037,786	5038	5040	0	10451	10455	10456	10455,656	10457	10459	2
perm36	98	98	98	98,003	98	101	0	236	236	236	236,005	236	240	0	567	567	567	567,004	567	570	0	1612	1615	1615	1614,554	1615	1619	0	4541	4542	4542	4541,998	4542	4547	0	10170	10173	10174	10174,155	10175	10177	2
perm37	93	93	93	93,003	93	96	0	247	247	247	247,003	247	250	0	614	614	614	614,003	614	617	0	1672	1673	1675	1674,177	1675	1679	2	3493	3494	3494	3494,069	3494	3498	0	10171	10173	10174	10174,821	10176	10178	2
perm38	88	89	89	88,92	89	92	0	258	258	258	258,005	258	262	0	633	633	633	633,004	633	637	0	1470	1472	1473	1472,509	1473	1477	1	4011	4027	4027	4027,221	4028	4032	1	9966	9969	9970	9970,212	9971	9973	2
perm39	88	88	88	88,003	88	91	0	262	262	262	262,003	262	265	0	558	558	558	558,004	558	562	0	1558	1560	1561	1560,429	1561	1564	1	3979	3981	3981	3981,141	3981	3986	0	10973	10976	10977	10976,605	10978	10979	2
perm40	89	89	89	89,005	89	93	0	243	243	243	243,003	243	246	0	588	588	588	588,003	588	591	0	1686	1688	1689	1688,53	1689	1693	1	3458	3459	3459	3459,05	3459	3464	0	9167	9181	9182	9181,504	9185	9187	1
perm41	94	94	94	94,003	94	97	0	240	240	240	240,003	240	243	0	583	583	583	583,004	583	587	0	1585	1588	1589	1588,142	1589	1592	1	4582	4584	4584	4584,169	4584	4589	0	9667	9670	9671	9670,612	9672	9674	2
perm42	94	94	94	94,005	94	98	0	241	242	242	242,001	242	245	0	614	614	614	614,316	615	619	1	1418	1419	1421	1420,667	1422	1425	3	4698	4700	4700	4699,828	4700	4704	0	9665	9669	9670	9669,535	9670	9673	1
perm43	101	101	101	101,002	101	103	0	259	259	259	259,004	259	263	0	754	754	754	754,003	754	757	0	1442	1442	1445	1443,859	1445	1449	3	4684	4701	4701	4700,851	4701	4706	0	10437	10440	10441	10440,717	10442	10444	2
perm44	114	114	114	114,004	114	118	0	243	243	243	243,004	243	247	0	735	735	735	735,003	735	738	0	1690	1692	1693	1692,498	1693	1697	1	3887	3889	3889	3888,953	3889	3893	0	9881	9884	9885	9884,915	9886	9888	2
perm45	93	93	93	93,004	93	96	0	225	225	225	225,003	225	228	0	576	576	576	576,003	576	579	0	1575	1577	1578	1577,463	1578	1582	1	4355	4356	4356	4356,068	4356	4361	0	9882	9884	9885	9885,325	9886	9888	2
perm46	94	94	94	94,004	94	98	0	263	263	263	263,004	263	267	0	614	615	615	614,835	615	618	0	1554	1554	1557	1555,967	1557	1560	3	4520	4522	4522	4522,169	4522	4526	0	9700	9705	9706	9706,5	9707	9710	2

Tableau 3.10 – Pourcentage d'outliers pour ShellSort sur ARM

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
ShellSoort	128	0.3	0.13
	256	0.3	0.13
	512	2.41	0.7
	1024	3.74	0.2
	2048	19.8	11.6
	4096	2.2	0.8

Le tableau 3.9 et la Figure 3.19 montrent que les boites de moustaches (boxplots) pour l'algorithme ShellSort sont presque des lignes. D'une part, les boites de moustaches ne sont pas symétrique par rapport la médiane et la moyenne. D'autre part, le tableau A.4 présente le pourcentage supérieur d'outliers pour l'algorithme shellSort est négligeable pour N=128,256,512,1024 et 4096 mais nous remarquons que le nombre des valeurs aberrantes est importante pour N=2048. Dans cette partie, nous ne pouvons pas mettre toutes les graphes grâce aux nombres importants des données (voir annexe A). Afin de comparer la mesure expérimentale de la complexité en temps des algorithmes de tri, nous avons calculé la moyenne des temps d'exécution de chaque algorithme de tri en fonction du taille de tableau pour donner une idée sur la variation temporelle de l'algorithme en fonction de N.

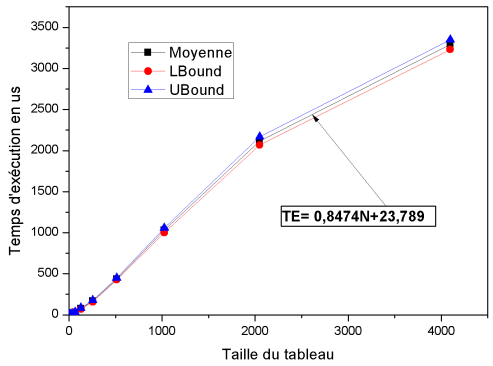


FIGURE 3.20 – ShellSort

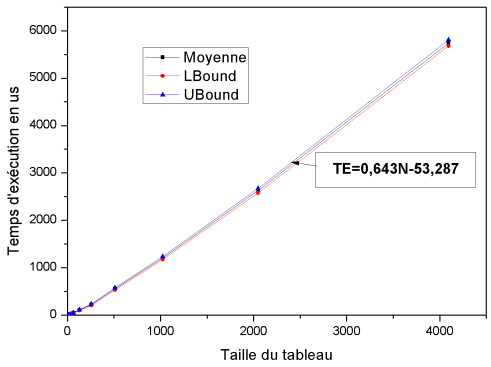


FIGURE 3.21 – TimSort

La Figure 3.20 présente la variation temporelle de l'algorithme ShellSort, LBound et UBound en fonction de N. Nous observons que la courbe du temps est presque linéaire. De plus, les Figure 3.21 montre que la fonction de l'algorithme TimSort est linéaire en termes de N.

3.5 Conclusion

Dans ce chapitre, nous avons effectué plusieurs analyses temporelles et énergétiques sur les algorithmes de tri afin de valider le choix du meilleur algorithme sur le processeur ARM Cortex A9. Pour cela, nous avons eu recours à deux modes d'exécution *Standalone* et Linux embarqué. Les résultats expérimentaux utilisant la plateforme Zedboard montrent

que le temps moyen de l'algorithme de tri ShellSort pour le mode *Standalone* est plus rapide que les autres si le nombre des données est supérieur à 64. Par ailleurs, nous avons calculé la consommation d'énergie des différents algorithmes pour décider à la fin qu'il est le meilleur algorithme utilisé sur le processeur ARM Cortex A9. En se basant sur toutes ces analyses, nous allons appliquer dans le chapitre 3 plusieurs optimisations matérielles pour ces huit algorithmes dans le but d'améliorer les performances en termes du temps, écart type et de la consommation.

Chapitre 4

Conception et synthèse haut niveau des algorithmes de tri

Sommaire

4.1 Introduction 71
4.2 Évolution des méthodologies de conception haut niveau . . . 71
4.2.1 Avantages du HLS 72
4.2.2 Les problèmes du HLS 73
4.3 Génération de l'IP matériel par HLS 74
4.3.1 Synthèse et optimisation en RTL 75
4.3.2 Développement des accélérateurs matériels avec Vivadeo HLS . 78
4.3.3 Intégration des IPs dans un environnement SW/HW avec HLS 82
4.4 Evaluation des performances des algorithmes de tri dans un environnement SW/HW 83
4.5 Conclusion 93

4.1 Introduction

Au cours des dernières années, Les outils de synthèse de haut niveau HLS *(High-Level Synthesis)* [77, 76, 110, 57, 83, 4] ont évolué pour augmenter la productivité des conceptions basées sur FPGA. Malgrès les défaillances rencontrées dans la première génération, cet outil a répondu aux attentes des concepteurs de matériel ; différentes raisons ont encouragé les chercheurs à les améliorer pour produire des conceptions matérielles plus efficaces. Parmi ces raisons, nous pouvons citer :

- Augmentation croissante de la capacité de silicium,
- Utilisation des accélérateurs matériels et des SoCs pour les conceptions récentes,
- Réduction du temps de mise sur le marché,
- Amélioration de la productivité de la conception en réutilisant les conceptions comportementales au lieu des conceptions de type RTL (Register-Transfer-Level),
- Séparation de l'algorithme de l'architecture pour permettre une vaste exploration des alternatives d'implémentation [76]

Dans ce chapitre, nous présentons une étude sur les algorithmes de tri en utilisant l'outil HLS. En effet, il existe quelques algorithmes complexes qui rendent la programmation manuelle difficile et possèdent un temps d'exécution important pendant le processus de validation. Dans ce cas, l'outil HLS est une solution efficace pour diminuer le temps de mise sur le marché "time to market". Pour cela, beaucoups de projets académiques et commerciaux actives dans ce domainene cessent d'améliorer la qualité de conception [111, 112]. Cependant, malgré prolifération des études de conception HLS [113, 114], ceux ci ne prennent pas en considèrent les limitations présentées dans ces solutions. Malheureusement, la programmation des algorithmes avec l'outil HLS n'est pas assez simple à compiler en RTL car il faut tout d'abord raffiner l'algorithme en utilisant une façon particulière pour faciliter le parallélisme et l'opimisation au niveau des instructions. Dans ce contexte, nous choisissons les algorithmes de tri comme une application cible afin de les optimiser et de les comparer en utilisant une méthode de conception haut niveau sur la plateforme Zedboard [54]. A partir de cette étude, nous permettons de calculer les performances des algorithmes en termes de temps d'exécution, Écart type, consommation d'énergie et des ressources sur FPGA. Nous concluons ce chapitre par une étude comparative entre les différents algorithmes de tri afin d'identifier le meilleur parmi les huit.

4.2 Évolution des méthodologies de conception haut niveau

Une synthèse HLS se base sur la programmation logicielle de l'application en considérant un ensemble de contraintes spécifiées par le concepteur afin de générer un module matériel qui répond à ces contraintes comme montre la Figure 4.1. L'ensemble des contraintes permettent de fixer la plateforme matérielle, sa technologie et l'interface entrées/sortie. Il est possible aussi de réduire le temps de conception, le temps de latence ou la consommation électrique et les ressources matérielles utilisées. Ainsi, la synthèse HLS a un apport principal qui permet de générer plusieurs architectures spécifique à une plateforme cible et un environnement de développement précis à l'aide d'une seule description fonctionnelle. En effet, la synthèse HLS permet de créer des composants adaptés aux contraintes

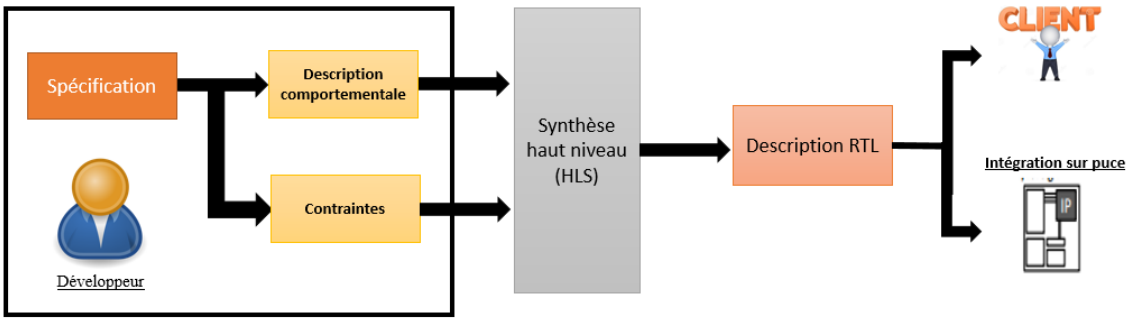


FIGURE 4.1 – Flot de conception du HLS

proposées par le développeur. Les accélérateurs matériels générés (IP) sont optimisés et très peu compréhensibles. De ce fait, le fournisseur considère ces accélérateurs comme des boites noire (c à d il ne connaît pas leurs implémentations matérielles) pour les utiliser dans plusieurs applications. Différents outils de synthèse HLS sont accessible dans le marché. On cite à titre d'exemple Vivadeo HLS fourni par XILINX, Synphony-C-Compiler propre à Synopsis, le Catapult-C de Mentor Graphics. L'outil de synthèse HLS effectue une séquence de tâches. Tout d'abord, la spécification de conception est écrite au niveau algorithmique par un langage de programmation de haut niveau basé sur C (que ce soit ANSI C ou C + +) [Gajski et al.2010]. À ce niveau, l'accent est mis sur les calculs effectués par un composant individuel et la manière dont il mappe des séquences d'entrées à des séquences de sorties. La deuxième étape consiste à compiler la spécification algorithmique de haut niveau en une représentation intermédiaire sous la forme de divers graphiques de flux (flux de données et sde contrôle (CDFG)). Les phases suivantes consistent en l'ordonnancement des opérations, l'allocation des ressources et la liaison des ressources. L'allocation de ressources détermine le type et le nombre de ressources matérielles (additionneurs, multiplicateurs, registres) qui doivent être utilisées pour implémenter la conception. Ensuite, dans la liaison de ressources, chaque opération est assignée aux composants matériels alloués. Une fois les décisions d'allocation, de planification et de liaison prises, l'objectif est de générer une architecture RTL. La figure 4.2 illustre les différentes phases HLS.

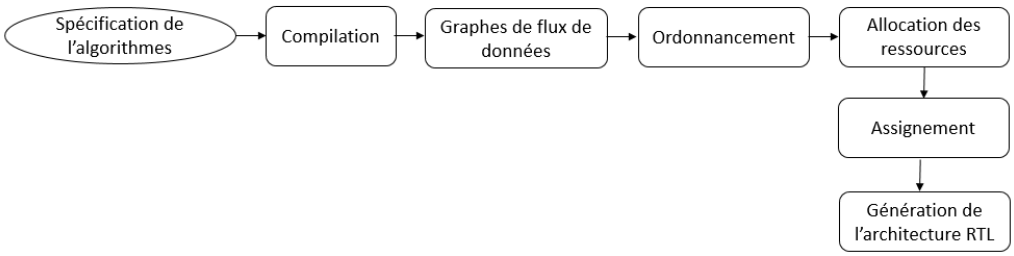


FIGURE 4.2 – Les étapes de HLS

4.2.1 Avantages du HLS

Avec une synthèse de haut niveau, l'algorithme peut être facilement transféré du logiciel à une implémentation matérielle. Actuellement, beaucoup de compilateurs de génération peuvent travailler avec le code C standard, compiler de C à RTL, et s'appuyer sur les

outils spécifiques au fournisseur de FPGA pour faire la synthèse sur un FPGA. Cependant, il suffit rarement de compiler du code logiciel pour FPGA. L'algorithme du logiciel a souvent été optimisé pour l'implémentation sur un processeur à base de CPU. Il est généralement nécessaire de restructurer l'algorithme pour qu'il correspond mieux au matériel. Cependant, au lieu d'effectuer cette étape manuellement dans la traduction de C à RTL, cette restructuration peut être effectuée directement avec HLS sur la source de haut niveau, ce qui est plus facile et moins sujette aux erreurs. L'un des principaux avantages de travailler directement dans un langage de haut niveau est que le chemin de contrôle est souvent implicite dans la représentation du langage. Les outils HLS analysent la structure de l'algorithme (boucles, branches, etc.) pour extraire et construire automatiquement le chemin de contrôle. Par contre, l'implémentation manuelle dans RTL nécessite de coder explicitement le chemin de contrôle ainsi que le chemin de données. Pour les algorithmes complexes, la conception du chemin de contrôle peut nécessiter autant d'efforts que la conception du chemin de données.

Pour chacune des conceptions générées, la simulation ou la vérification est nettement plus rapide au sein des outils de haut niveau. Ceci est dû à la vérification à lieu à un niveau supérieur [31].

Dans les environnements de coprocesseur, où le calcul est partagé entre un FPGA et un CPU, HLS permet de compiler le même code source en tant que matériel ou logiciel (par exemple [29, 35]). Cela permet d'échanger rapidement différents modules entre l'implémentation matérielle et logicielle, ce qui facilite la recherche de l'optimal. Pour un code structuré de manière appropriée, les outils HLS modernes peuvent générer des conceptions aussi efficaces que la RTL codée à la main en termes de ressources et de vitesse de traitement [6, 25, 37].

Les outils de synthèse de haut niveau (HLS) peuvent fournir des avantages significatifs pour l'implémentation des algorithmes de tri sur les FPGAs. La représentation de niveau supérieur (généralement basée sur C) permet d'exprimer plus facilement les algorithmes, réduisant ainsi considérablement les temps de développement. Le niveau supérieur facilite également l'exploration de l'espace de conception, ce qui facilite l'optimisation du compromis entre les ressources et la vitesse de traitement.

4.2.2 Les problèmes du HLS

Avec la plupart des outils HLS, l'algorithme doit être implémenté avec une façon particulière pour permettre aux outils de synthèse d'identifier et d'exploiter le parallélisme au niveau architectural. Cela nécessite une restructuration du code. Même avec une telle restructuration du code, le matériel résultant ne peut pas être bien optimiser et souffre toujours des mauvais performance. La programmation matérielle bas niveau et haut niveau peut souvent mener à une utilisation inefficace du matériel. De plus, les algorithmes basés fortement sur les pointeurs et l'arithmétique du pointeur ne se synthétisent pas bien avec le matériel [37]. Dans le logiciel, un pointeur est une adresse d'une variable dans une mémoire. Dans ce cas, il y a deux problèmes majeurs dans le contexte d'une implémentation FPGA. Premièrement, la mémoire sur puce est distribuée, avec de nombreux blocs indépendants et relativement petits qui possèdent chacun son propre espace d'adressage. Deuxièmement, dans le matériel plusieurs variables sont stockées dans des registres. Un autre problème pour HLS est la récursivité. Elle est une autre technique logicielle qui

ne se génère pas bien au matériel. Les fonctions ou procédures du logiciel stockent les variables locales sur la pile, de sorte que chaque appel dispose d'un espace séparé pour le stockage de la variable. Sur un FPGA, une fonction est implémentée comme un bloc de matériel, avec un appel de procédure multiplexant ce matériel pour chaque invocation. Par contre, il n'y a pas la notion de pile dans HLS. Les variables locales sont généralement stockées dans des registres partagés avec chaque appel de la procédure. La réutilisation de registres par des appels successifs limite l'utilisation des fonctions récursives à la récursion de queue. En général, les algorithmes récursifs doivent être restructurés pour utiliser une itération équivalente.

En effet, la conception du système est difficile en raison de la nécessité de concevoir la synchronisation. La conception de logiciel mono-thread n'est pas un problème, car tout est implicitement séquentiel. Par contre, avec plusieurs threads, il est nécessaire d'assurer la synchronisation aux points clés. Donc, il y a beaucoup plus de tâches à synchroniser. Alors que HLS fait raisonnablement bien la programmation et le pipelining des séquences d'opérations, et la synchronisation de la distribution du calcul sur plusieurs processeurs parallèles, il se débat avec une synchronisation plus complexe. Le code RTL généré par les outils HLS n'est pas particulièrement lisible par un humain, ce qui le rend très difficile à modifier au niveau hardware. Globalement, le but de l'utilisation de HLS est d'éviter la programmation RTL. Cependant, il est toujours nécessaire de vérifier la sortie RTL [31, 32]. Si la vérification RTL indique une erreur, il est donc très difficile de préciser ou se trouve le problème dans HLS. En logiciel, toutes les structures de données sont stockées dans une seule mémoire monolithique.

4.3 Génération de l'IP matériel par HLS

Le processus de développement d'une architecture matérielle par le HLS passe par plusieurs étapes dont la plus compliquée est l'étape de génération de l'IP matériel. Dans ce contexte, nous utilisons l'outil Vivado HLS développé par Xilinx afin d'effectuer la synthèse de l'accélérateur matériel à partir d'un langage de description haut niveau. Il sera retenu comme outil HLS de pointe pour la synthèse matérielle de bas niveau. Vivado HLS est un outil HLS à la pointe de la technologie créé en 2011, Xilinx a acquis l'outil AutoPilot HLS développé par AutœSL, dans le cadre de Vivado Design Suite. Vivado HLS compile les langages d'entrée C en RTL, qui peuvent ensuite être synthétisés et implémentés sur la logique programmable d'un FPGA Xilinx. Cet outil est étroitement intégré avec le reste des outils de conception Xilinx et fournit un support complet et des fonctionnalités pour créer une implémentation optimale pour l'algorithme C.

Le flot de conception d'un design matériel avec Vivadeo HLS commence par une compilation suivi d'une exécution et débogage du code C. Par la suite, dans la deuxième étape, on effectue la synthèse de l'algorithme C pour le convertir au niveau RTL. Optionnellement, on peut utiliser les directives suivantes :

– PIPELINE,
– ARRAY_PARTITION,
– DATAFLOW,etc

La Figure 4.3 montre les différents directives dont le choix dépend de la taille et type de données, de la structure de programme(l'enchainement et type des instructions utilisées), du type d'interface choisie, du type d'optimisation souhaité.

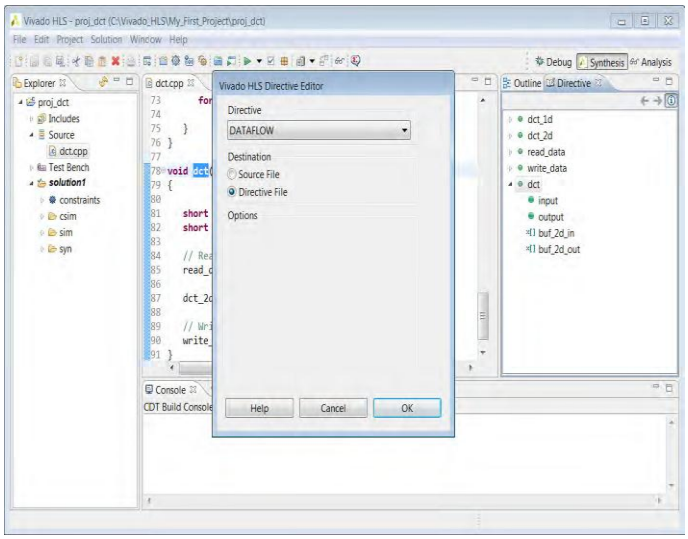


FIGURE 4.3 – Fenêtre de choix et de configuration de directives

Après avoir achevé la synthèse de l'algorithme C, un rapport de synthèse est généré d'une façon automatique afin d'analyser le design matériel en terme du temps d'exécution et de surface FPGA. La dernière étape consiste à enregistrer l'IP matériel généré sous un format spécifique. La Figure 4.4 présente une vue d'ensemble des fichiers d'entrée et de sortie avec Vivado HLS. Le fichier d'entrée peut contenir :

- Fonction C rédigée en langage C, C++ ou system C.
- Des contraintes qui comprennent la période et l'incertitude de l'horloge.
- Les directives sont facultatives et orientent le processus de synthèse pour une telle implémentation spécifique.
- Un test bench ou des fichiers associés.

Pour les fichiers de sortie, ils comportent :

- Une implémentation RTL décrite en langage (HDL).
- Les rapports de synthèse.

4.3.1 Synthèse et optimisation en RTL

Nous souhaitons développer des algorithmes de tri sur un FPGA Xilinx de la famille Zynq XC7z020clg484 qui est disponible dans notre équipe car elle n'a pas trop cher. Nous considérons dans ce travail une fréquence maximale 100MHz. A partir des résultats, nous pouvons construire une nouvelle solution pour notre projet en utilisant plusieurs contraintes et différentes directives d'optimisation. Par la suite, nous synthétisons cette solution. Dans ce contexte, nous répétons cette étape jusqu'à obtenir les performances désirées. A titre d'information, l'utilisation de plusieurs solutions permet de poursuivre le développement tout en conservant les résultats précédents. Dans cette partie, nous présentons les différentes optimisations appliquées aux algorithmes de tri définis dans la section précédente en utilisant des directives HLS avec la taille des données 32/64 bits. Ensuite, nous expliquons notre architecture d'exécution. Enfin, nous proposons différentes données en entrée en utilisant la distance de Lehmer pour prendre une décision finale. Dans la littérature, il existe plusieurs types d'optimisation qui s'applique sur le code C pour obtenir une implémentation hardware efficace. Nous citons par exemple :

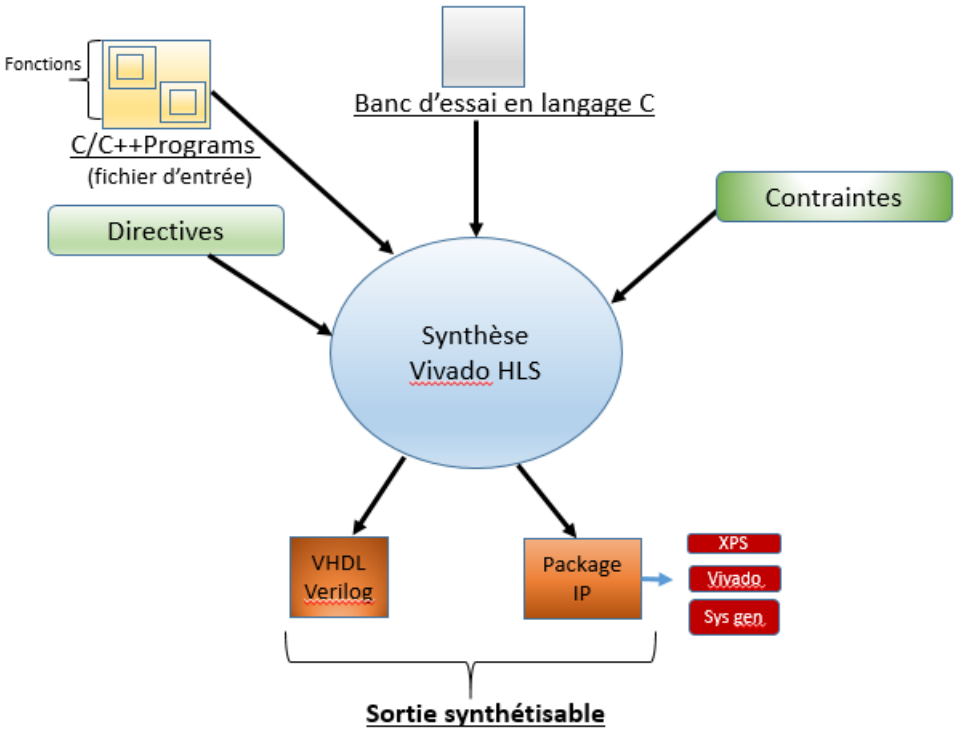


FIGURE 4.4 – Flot de conception du Vivado HLS

Optimisation des boucles

Parmi ces optimisations, nous pouvons détailler :

- **UNROLL :** Cette directive forme une liste d'itérations qui se déroulent d'une façon indépendante. Pour cela, nous pouvons fixer le nombre de répétitions de la boucle à l'aide d'un facteur de déroulement comme montre la figure 4.5.

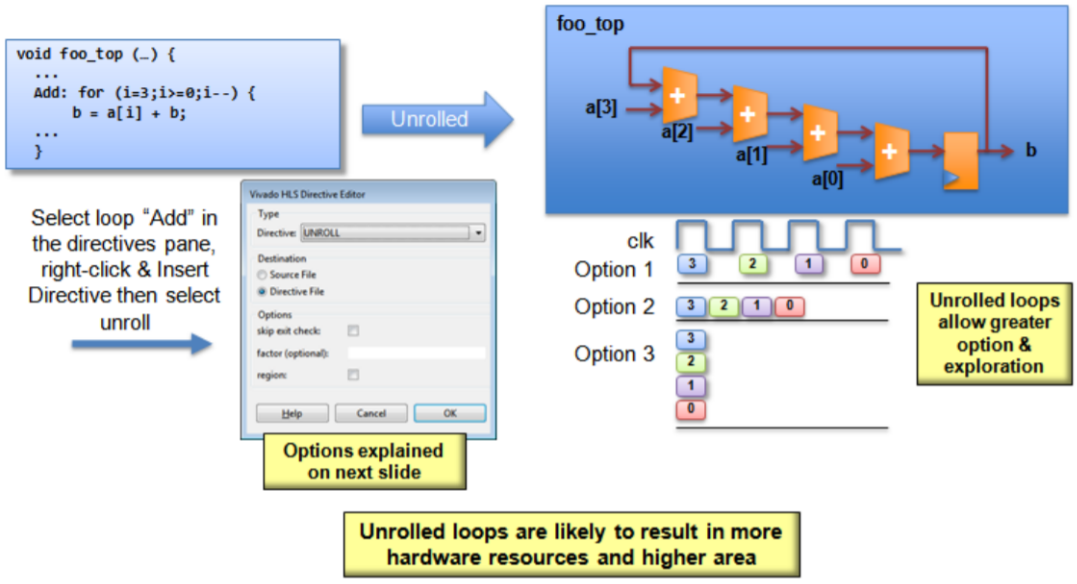


FIGURE 4.5 – Exemple de boucle optimisée avec le directive UNROLL

- **MERGE :** Si on a plusieurs boucles successives, on remarque qu'il y'a parfois des cycles d'horloge inutiles qui sont créés. Pour cela, on peut fusionner les boucles dans un seul boucle comme indiqué dans la Figure 4.6.

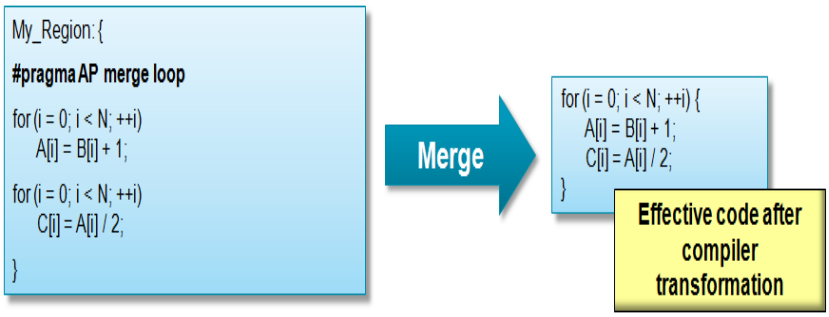


FIGURE 4.6 – Exemple de boucle optimisée avec le directive MERGE

- **PIPELINE :** Cette directive permet aux boucles de nature séquentielle de fonctionner simultanément au niveau RTL avec un temps de décalage de tel sorte de permettre à la prochaine tâche d'être exécuter avant l'achèvement de la tâche en cours (Figure 4.7).

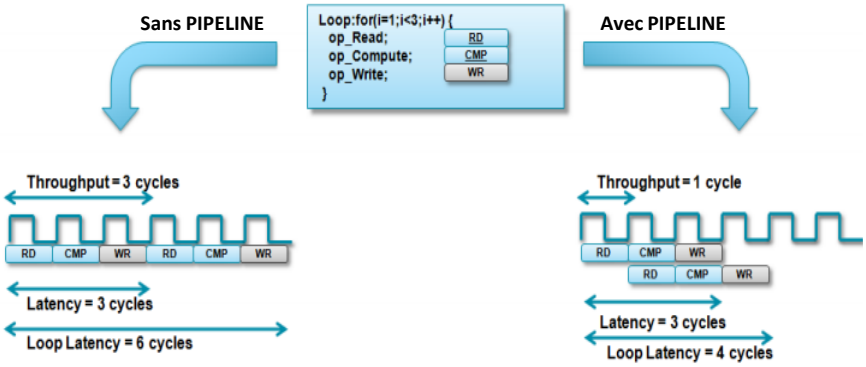


FIGURE 4.7 – Exemple de boucle optimisée en PIPELINE

Optimisation des tableaux

Les tableaux programmés en langage C sont généralement stockés dans les mémoires. Les optimisations effectuées sur les tables constituent un facteur important pour améliorer la surface et les performances.

- **ALLOCATION :** Cette directive permet de limiter le nombre de ressources matérielles utilisées pour implémenter des fonctions, boucles ou opérations spécifiques.
- **RESOURCES :** Cette directive est essentielle pour la programmation du tableau et opération arithmétique dans le niveau RTL. Si on n'utilise pas cette directive alors Vivado HLS détermine la ressource.
- **ARRAY PARTITION :** Il divise le tableau en des vecteurs de taille plus petits.

Optimisation des fonctions

- **INLINE :** Cette directive permet l'optimisation au niveau hiérarchie de la fonction. En effet, ce directive regroupe l'ensemble des opérations rédigées en C qui sont traduites en portes logiques. Par la suite, cette directive permet de simplifier les opérations inutiles du code RTL afin d'augmenter le débit calculé pour l'IP (Figure 4.8). Cette figure montre que Inlining permet une optimisation à travers de la fonction hiérarchie. De plus, L'utilisation d'un nombre important d'INLINE dans le découplage RTL peut ralentir l'exécution.

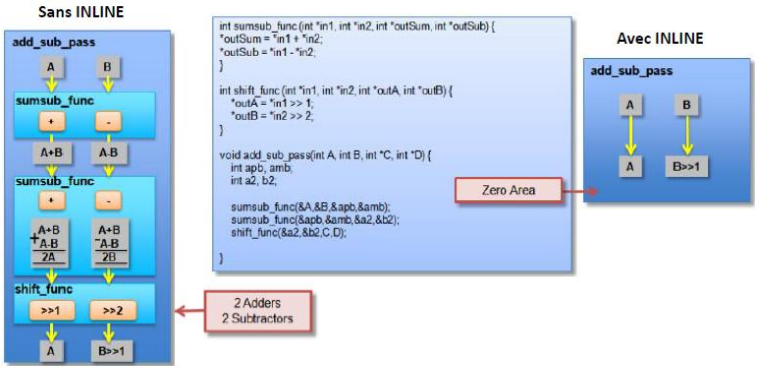


FIGURE 4.8 – Exemple de fonction optimisée avec le directive INLINE

4.3.2 Développement des accélérateurs matériels avec Vivadeo HLS

Nous envisageons dans cette partie d'implémenter différents algorithmes de tri en utilisant l'outil de synthèse HLS pour trouver le meilleur algorithme qui sera utilisé dans plusieurs domaines. Nous avons considéré le mode *Standalone* pour développer des accélérateurs matériels des algorithmes qui permettent d'améliorer les performances.

Développement des algorithmes de tri avec différentes optimisations

Afin d'avoir une implémentation matérielle efficace, nous avons appliqué les étapes d'optimisation suivantes au code C pour chaque algorithme de tri :

- Loop unrolling : Les éléments de tableau sont stockés dans des structures de mémoire BRAM, caractérisées par des ports physiques doubles. Les ports physiques doubles peuvent être configurés comme : deux ports de lecture, deux ports d'écriture ou un port pour chaque opération. Nous pouvons dérouler la boucle par le facteur = 2 comme indique la figure 4.9.
- Loop pipelining : Dans notre conception, les itérations de la boucle sont pipelinées avec une seule différence de cycle d'horloge entre les deux en appliquant Loop pipelining avec une Initiation Interval (II) = 1 qui correspond au nombre de cycles d'horloge entre les instants de début des itérations de boucle consécutives afin de réduire le temps de latence d'exécution.
- Input/output Interface : Pour le transfert des données, les ports d'entrée/sortie sont configurés pour utiliser le protocole AXI-Stream avec des signaux de communication

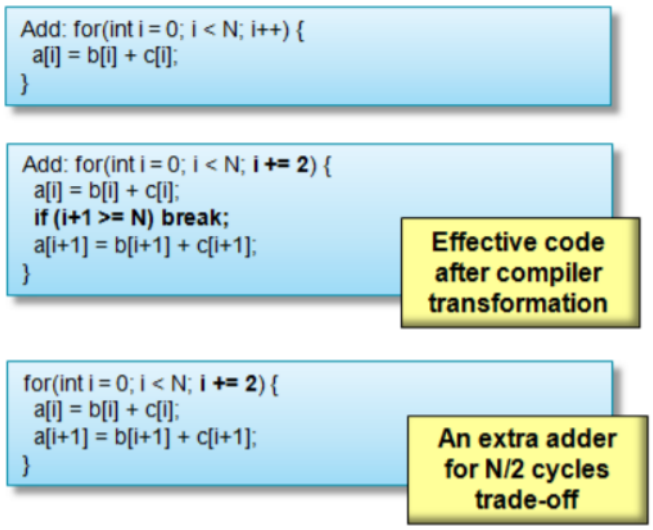


FIGURE 4.9 – Exemple de fonction optimisée avec le directive UNROLL avec factor=2

minimum (DATA, VALID et READY). Alors que le protocole AXI-Lite est utilisé à des fins de configuration de conception comme par exemple, pour identifier l'état actuel du système (démarrage, prêt, occupé).

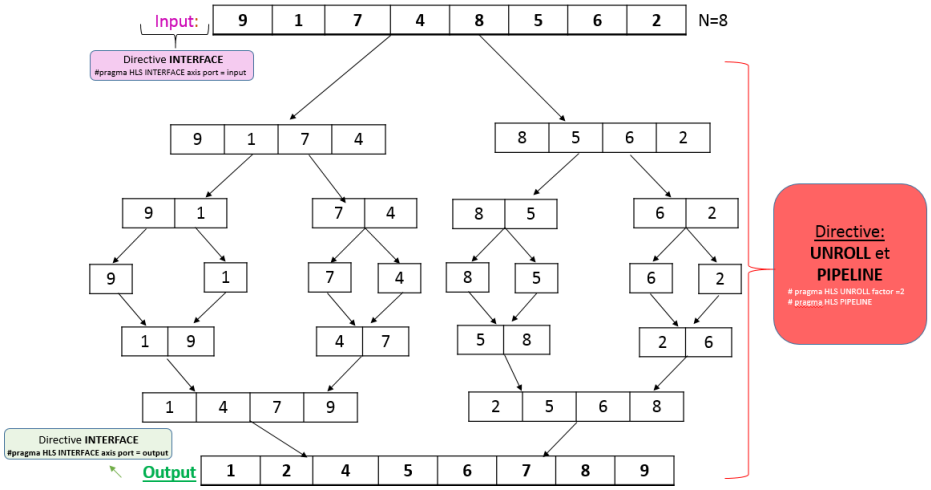


FIGURE 4.10 – Démarche de l'algorithme MergeSort

Les figures 4.10 et 4.11 présentent les étapes de l'algorithme hardware optimisé MergeSort. Pour créer une interface AXI4-Stream, nous avons appliqué le directive **INTERFACE** pour le tableau d'entrée "Input" et le tableau de sortie "Output" afin de spécifier l'interface AXIS avec les signaux TDATA, TVALID et TREADY. Nous avons cherché à améliorer le débit maximal achevé par l'IP matériel. Pour cette raison, les deux options d'optimisation PIPELINE et UNROLL sont rajoutées dans l'algorithme.

Architecture matérielle des algorithmes de tri

Aujourd'hui, l'architecture hétérogène est très prometteuse pour l'extraction haute performance en combinant l'accélérateur matériel reconfigurable FPGA avec l'architecture

```
#define size N
struct out_axis
{
 int data[size]
 int TLast[size]
};

Function Mergesort(int *input, struct out_axis *
    result)
{
 # pragma HLS INTERFACE s_axilite port=return
 # pragma HLS INTERFACE axis port=input
 # pragma HLS INTERFACE axis port=result
 int array[size];
 int i, z;
 mergesort_label0:for(int i=0; i<size; i++)
 {
  #pragma HLS UNROLL factor=2
  #pragma HLS PIPELINE
  array[i]=input[i];
 }
 mergesort_label1:for(int c=1; c<=size-1; c=2*c)
 {
  #pragma HLS PIPELINE
  mergesort_label2:for(int l=0; l<size-1; l=l+2*c
      )
  {
   #pragma HLS UNROLL factor=2
   #pragma HLS PIPELINE
   int mid=l+c-1;
   int r=min(l+2*c-1, size-1);
   merge(array, l,mid, r);
  }
 }
 mergesort_label3:for(z=0; z<size; z++)
 {
  #pragma HLS UNROLL factor=2
  #pragma HLS PIPELINE
  if(z<size-1)
  {
   result->data[z]=array[z];
   result->TLast[size-1]=0;
  }
  else if (z==size-1)
  {
   result->data[size-1]=array[size-1];
   result->TLast[size-1] = 1;
  }
 }
}
```

FIGURE 4.11 – Implémentation Hardware optimisée de l'algorithme MergeSort

classique. L'utilisation de cette architecture entraîne une amélioration significative de la performance et de l'efficacité énergétique. Dans la littérature, différentes architectures hétérogènes telles que Intel HARP, Mi-crosoft Catapult et Xilinx Zynq sont utilisées. L'architecture Xilinx promet un parallélisme massif pour procéder à l'amélioration de l'accélération matérielle par la technologie FPGA. En fait, la technologie FPGA peut offrir de meilleures performances (consommation d'énergie, temps ...), jusqu'à 10 fois plus rapide que le CPU. L'outil de synthèse haut niveau Vivado HLS permet de créer l'IP via l'interface Axi Stream programmé en VHDL. Cependant, la figure 4.12 montre comment l'IP HLS (tri HLS) est exporté dans la conception en utilisant l'outil vivado. Tout d'abord, les données en entrée sont stockées dans la carte SD. Pour lire et écrire ces données, la bibliothèque file fat system (ff.h) est utilisé. Ces données sont transférés du processeur (ZYNQ) au IP (Algorithme de tri HLS) via la communication AXI-DMA. Ensuite, Bitstream est généré et l'outil SDK lancé pour calculer le temps de traitement sur l'IP avec la bibliothèque xtime_l.h. Une fois les données triées, le résultat est réécrit via le chemin inverse. La figure 4.13 montre que l'accès direct à la mémoire AXI (AXI DMA) offre un accès direct à large bande passante entre la mémoire et les périphériques cibles de type

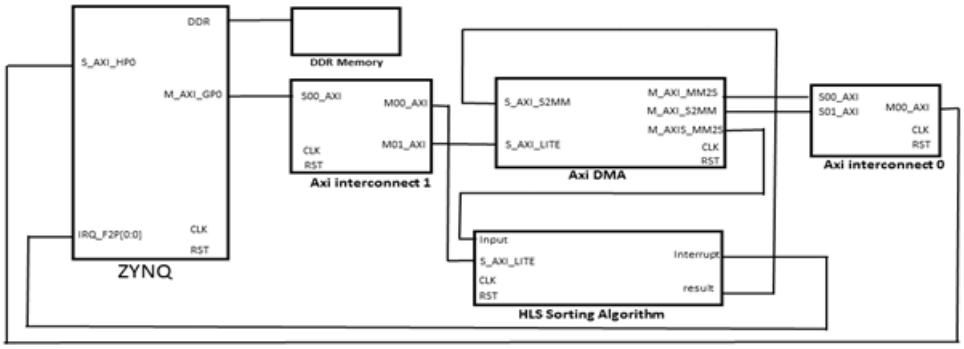


FIGURE 4.12 – Architecture Hardware des algorithmes de tri

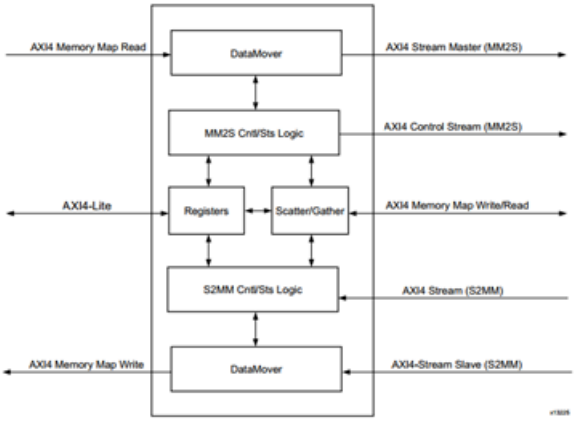


FIGURE 4.13 – Bloc DMA

AXI4-Stream. D'une part, nous considérons que l'option de moteur de collecte de dispersion est désactivée dans l'interface AXI DMA. Nous utilisons AXI4 sur le second type d'interface de flux car ce type est très flexible, productif et disponible. Il est conçu pour les données de streaming à haute vitesse.

Choix des données

Dans cette partie, nous avons proposé différentes données d'entrées, qui sont stockées dans plusieurs systèmes de fichiers. Tout d'abord, nous avons présenté dans la Figure 4.14 la gestion des fichiers sur une carte SD (Secure Digital Memory Card) tout en conservant une forte portabilité et aspect pratique du FPGA. Les cartes SD sont facilement ineffaçables avec les FPGA et sont largement utilisées dans les supports de stockage portables. Aujourd'hui, plusieurs études utilisant le contrôleur de carte SD avec FPGA jouent un rôle important dans différents domaines. Ils reposent ainsi sur l'utilisation d'une interface API (Application Programming Interface), d'un bus AHB (Advanced High Performance Bus), etc. Ils sont dédiés à la réalisation de communications ultra-rapides entre la carte SD et les systèmes supérieurs. Toute communication est synchrone avec une horloge fournie par l'hôte (FPGA). La conception et l'implémentation du système de fichiers sur une carte SD offre trois moyens d'innovation IP principaux :

– L'intégration et la combinaison du contrôleur de carte SD et du système de fichiers, donne un système hautement intégré et pratique.
– L'utilisation de la gestion des fichiers facilite le traitement. En outre, il améliore l'effi-

cacité globale des systèmes.
– La conception numérique offre une haute performance et permet une meilleure portabilité puisqu'elle est indépendante de la plateforme.

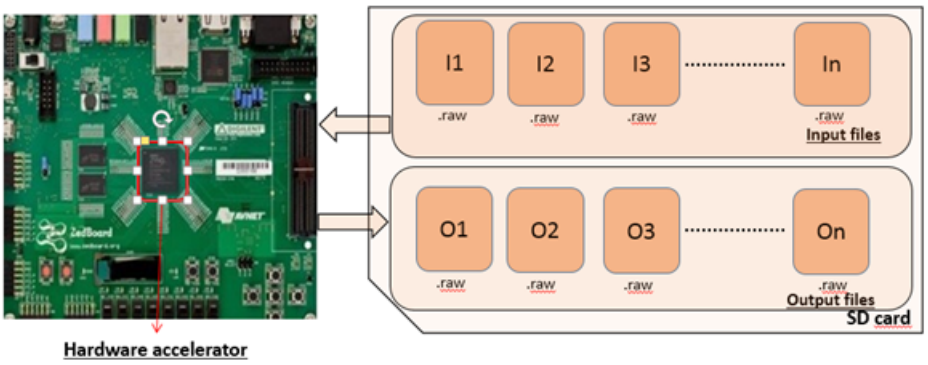


FIGURE 4.14 – Communication entre la carte SD et FPGA

Dans ce chapitre, nous avons implémenté les différents algorithmes en utilisant de nombreuses données codées sur 4 et 8 octets. Nous avons considéré dans le chapitre 3 seulement des données codées sur 32 bits car la configuration du processeur ARM Cortex A9 ne supporte pas des données codées sur 64 bits. Par contre, FPGA peut transférer des données codées jusqu'à 64 bits. De plus, plusieurs études sur les algorithmes de tri utilisent la notion de permutations comme entrée.

4.3.3 Intégration des IPs dans un environnement SW/HW avec HLS

L'outil HLS permet la génération des interfaces stream incorporant l'accélérateur matériel (IP). En effet, une fois la synthèse haut niveau est terminée, un fichier compressé (.Zip) générés contenant tous les composants matériels. Ce fichier est exporté dans le support des packages du Vivado afin de générer notre design SW/HW. À ce niveau, l'outil Vivado constitue un environnement complet pour la conception d'une architecture SW/HW finalisée. Ce dernier permet la mise en place d'un processeur embarqué connecté à un ou plusieurs accélérateurs matériels à travers des bus d'interconnections AXI spécifiques au plateforme sélectionnée. Par la suite, L'étape de synthèse et du placement et routage est nécessaire pour le déclenchement du fichier binaire (.bit). Pour la partie logiciel, l'outil SDK (Software Development Kit) prend en charge la compilation du code C et puis l'exécution du fichier(.elf) dans la carte FPGA. Pour notre application cible, nous avons utilisé la plateforme Zedboard à base d'un processeur hardware ARM Cortex A9 et des périphériques mémoires (DDR). En outre, la communication entre le processeur et le module matériel est assurée avec des bus d'interconnexion AXI4 de type Stream. La Figure V.13 illustre un exemple de schématique décrivant l'interfaçage entre le processeur et l'IP matériel généré par HLS en utilisant un AXI4-Stream et un DMA (Direct Memory Acess) pour un accès directe à la mémoire externe DDR. L'avantage majeur de l'outil Vivado HLS est qu'il offre la possibilité de créer une interface Lite ou Stream connectée directement avec l'accélérateur matériel. Lors de l'étape d'exportation RTL, Cette interface est ajoutée à la conception et dispose d'une entrée esclave et une sortie maitre contrôlée par des signaux de synchronisation qui assurent le début et la fin du transfert. Le bus AX4-Stream est

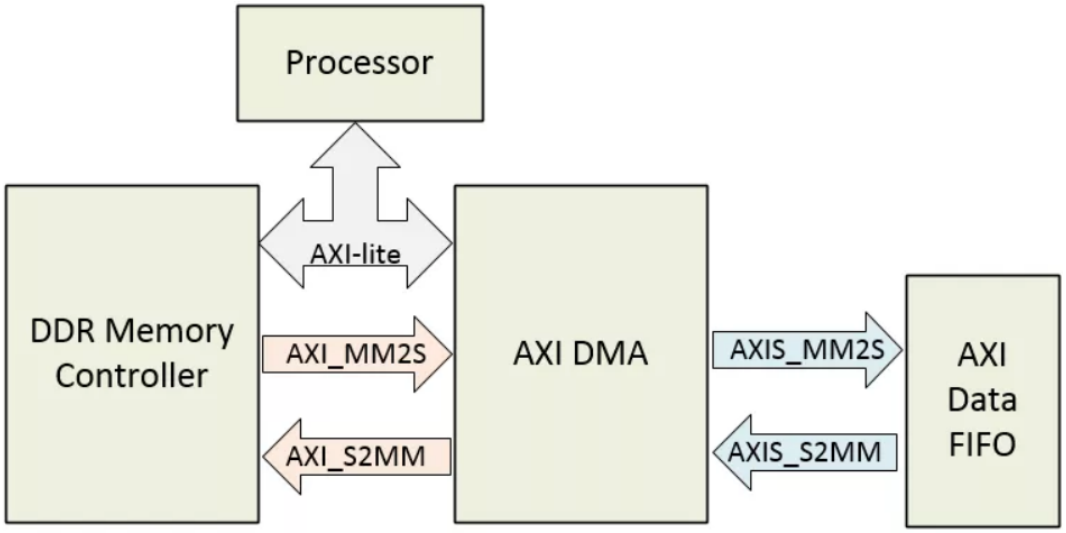


FIGURE 4.15 – Exemple de design SW/HW conçu avec Vivadeo

capable de transférer une quantité importante de données au même temps et améliore les performances en termes de temps d'exécution. Pour créer une interface AXI4-Stream, le directive INTERFACE est sélectionné afin de spécifier l'interface AXIS avec les signaux TDATA, TVALID and TREADY. Ces signaux sont responsable à contrôler la statut de transfert entre les ports maitre "Master" et esclave "Slave".

4.4 Evaluation des performances des algorithmes de tri dans un environnement SW/HW

Dans cette section, nous présentons les différents résultats obtenus de temps d'exécution, d'écart type et des ressources d'utilisation pour les algorithmes de tri sur l'architecture hétérogène CPU/FPGA. Nous comparons ces résultats pour l'implémentation matériel optimisées des algorithmes de tri en utilisant plusieurs permutations et vecteurs. nous avons fixé la taille du tableau de 8 à 4096 entiers codés sur 4 et 8 octets. Comme mentionné précédemment, nous avons limité la taille à 4096 éléments car les meilleurs algorithmes de tri sont principalement utilisés pour le système d'aide à la décision en temps réel pour les applications avioniques. Dans ce cas, Les algorithmes d'aide à la décision permettent de trier au maximum 4096 actions issues des blocs de calcul précédents. Pour mettre en évidence l'influence de l'accélération matérielle en terme de temps, écart type et surface, nous avons proposé une conception SW/HW pour les modules des algorithmes de tri. Ce concept exploite l'architecture de la plateforme ZC 702 qui intègre un processeur ARM Cortex A9 et de la PL. En effet, la lecture et l'écriture des données se fait à travers le processeur ARM alors que le module matériel est chargé dans l'FPGA. Après avoir déterminer toutes les parties du système, les bus internes de communication prend en charge le transfert des informations entre le processeur et l'FPGA. En se référant au travaux [115], nous avons prouvé que les performances du bus AXI4-Lite sont limitées vu qu'il assure le transfert séquentiel des données à 32 bits. Ceci engendre un temps de communication énorme et rends le transfert un peu lent. Pour cette raison, nous avons choisi de limiter l'étude sur le bus AXI4-Stream. Dans ce cas, il suffit de fixer la taille maximale du paquet

transmis (32 bits ou 64) en fonction du nombre total des entrées/sorties communiqués. Ensuite, les entrées sont transférées sous forme d'un flux de données vers la mémoire DDR en utilisant un accès DMA. Afin d'estimer les performances de nos travaux, nous avons adopté l'outil Vivadeo pour la création la synthèse de la du système (Processeur + accélérateur matériel). Par ailleurs, la compilation et le chargement du fichier exécutable (.elf) est assurée par l'outil SDK en mode Standalone. Premièrement, nous avons comparé le temps d'exécution et l'écart type entre plusieurs algorithmes de tri sur architecture hétérogène HW/SW.

Tableau 4.1 – Temps d'exécution en microseconde des algorithmes de tri avec des données codées sur 4 et 8 octets pour N<=64

		BubbleSort			InsertionSort			SelectionSort			ShellSort			QuickSort			HeapSort			MergeSort			TimSort		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Données codées sur 32 bits	8	11.244	17.302	23.181	11.036	17.106	22.98	10.78	16.84	22.29	11.816	17.88	23.768	12.55	18.68	24.6	11.903	17.99	23.8	12.84	18.93	24.847	12.12	18.12	24
	16	15.934	22.009	27.897	14.11	20.206	26.102	13.914	20	25.88	16.1	22.25	28.1	17.16	23.22	29.11	15.77	21.85	27.73	17.52	23.6	29.5	16	22.07	27.95
	32	32.317	38.763	44.647	24.02	30.15	36.03	23.85	29.9	35.25	27.12	33.25	39.14	28.3	34.3	40.2	24.97	31.1	37.02	28.36	34.48	40.36	27.51	31.18	37
	64	97.11275	103.255	109.136	59.44	65.56	77.185	59.13	65.25	71.13	54.12	60.24	66.12	59.6	62.8	68.6	46.66	52.78	58.66	52.33	58.465	64.34	44.8	50.89	56.7
Données codées sur 64 bits	8	11.312	17.447	23.385	11.042	17.15	22.95	10.94	17.01	22.88	11.92	18	23.87	12.56	18.67	24.57	11.87	17.96	23.85	12.79	18.89	24.78	12.12	18.13	24.01
	16	16.32	22.419	28.341	14.38	20.48	26.37	14.36	20.5	26.48	16.2	22.5	28.5	17.4	23.5	29.38	16.048	22.16	28.08	17.91	24.02	29.91	16.38	22.43	28.29
	32	33.433	39.53	45.416	24.73	30.82	36.69	24.55	30.31	36.5	27.8	33.9	39.78	28.9	34.9	40.8	25.673	31.76	37.636	28.95	35.05	40.9	25.74	31.8	40.86
	64	98.482	104.75	110.45	60.79	66.89	72.03	60.5	66.61	72.48	55.49	61.57	67.47	58	64.14	70.02	48.05	54.15	60.027	53.75	59.86	65.74	46.18	52.28	58.14

Le tableau 4.1 présente le temps d'exécution des algorithmes de tri avec des données codées sur 4 et 8 octets pour N<64. Pour cela, nous constatons que le temps d'exécution des algorithmes de tri pour N<64 est presque le même pour transférer des données codées sur 64 bits ou 32 bits en mode streaming ce qui est logique car le processeur ARM supporte seulement 32 bits. De plus, nous remarquons que à partir de la figure 4.16 et la figure 4.17 que l'algorithme SelectionSort est 1.01x-1.23x plus rapide que les autres algorithmes de tri si N<64.

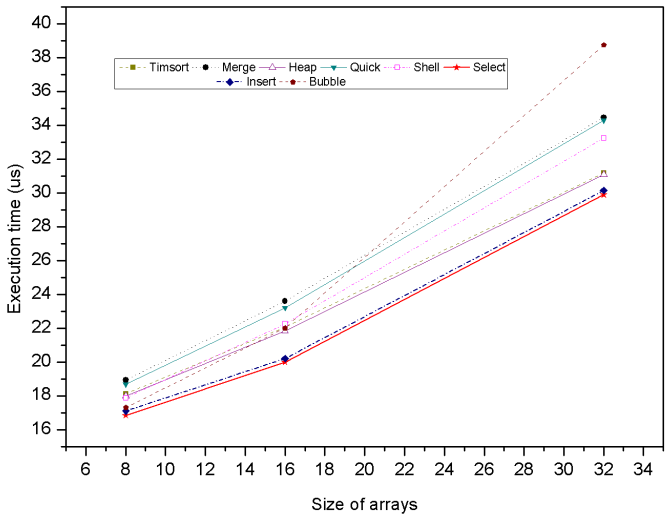


FIGURE 4.16 – Comparaison entre les algorithmes de tri en termes de temps d'exécution Si N<64 avec des données codées sur 32 bits

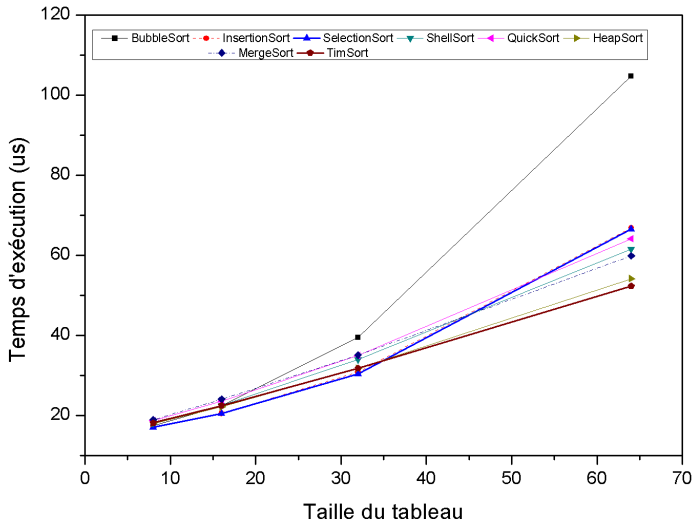


FIGURE 4.17 – Comparaison entre les algorithmes de tri en termes de temps d'exécution avec des données codées sur 64 bits Si N<64

Tableau 4.2 – Temps d'exécution en microseconde des algorithmes de tri avec des données codées sur 4 et 8 octets pour N>64

	N	BubbleSort			InsertionSort			SelectionSort			ShellSort			QuickSort			HeapSort			MergeSort			TimSort		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
Données codées sur 32 bits	128	348.937	358.255	360.94	190.94	196.96	202.84	191	197.2	203	120.2	126.3	132.1	131.7	137.9	143.7	96.535	102.665	108.54	105.71	111.88	117.76	88.39	94.5	100.3
	256	1344.71	1350.84	1356.72	708.1	714.21	720.039	701	707.2	713	281.8	293.55	293.8	342.7	351.2	357.07	222.47	228.57	234.45	223.1	231.3	237.25	182.9	189	197.9
	512	5304.07	5310.11	5315.92	2711.18	2717.29	2723.17	2698.4	2711.4	2717.1	662.16	668.2	674.1	984.1	1001.3	1007	459.92	466.05	471.93	476.4	782.5	788.4	387	393.17	399
	1024	21095.1	21101.2	21107.01	10629.9	10635.7	10645.4	10649	10655	10661.1	1607.6	1625.1	1803	2881	3121	3127	1001.8	1015.5	1024.6	1025.4	1031.5	1037.4	826.75	832.85	837.2
	2048	32767	83852.9	84177.8	31733.1	42209.7	42215.5	32767	42288	42294	4021.6	4027.67	4033.48	4232	10660	10665	2221.04	2227.1	2233	2182.1	2211.5	2217.3	1763.4	1769.5	1775
	4096	32767	333077	336274.5	31744.8	168307	168313	32767	168531	168536	9750.4	9756.4	9762	20630	101507	101512	4810.5	4848.5	4854.2	4728.9	4734.9	4740.7	3750.1	3756	3761.9
Données codées sur 64 bits	128	353.24	359.35	365.236	193.64	199.75	205.63	193.8	199.9	205.8	123	129.1	135	134.6	140.6	146.5	99.36	105.47	111.35	108.4	114.5	120	91.22	97.3	103.21
	256	1350.31	1356.42	1362.3	710.56	717.07	802.5	704.7	718.2	724.1	292.9	299	304.9	345.5	351.7	357.5	222.47	228.57	234.45	228.1	234.45	240.12	188.5	194.5	200.4
	512	5315.43	5321.45	5327.25	2722.62	2728.8	2716.5	2722.5	2728.6	2735.6	670.4	676.6	682.5	482.5	489	494.8	471.27	477.38	484.32	484.7	490.8	496.7	398.3	404.5	410.3
	1024	20926.15	21243.01	21140.12	10652.9	10656.11	10664.8	10672	10678.2	10684	1642.5	1648.7	1654.6	2893	3132.8	3138.7	1035.47	1041.58	1047.7	1048.1	1054.2	1060.1	849.6	855.7	861.8
	2048	32767	84217.69	84223.49	31734.6	41802.5	42264.7	32767	41525.3	46392	4067	4073	4078.8	4254.9	10682	10688	2244.7	2272.5	2278.43	2250.9	2257	2262.9	1809	1815	1821
	4096	32767	336356	336361.8	30728	168554	168578	32767	168618	168624	9814	9820.5	9826.3	67153	38599	38684	4930	4936.1	4941.95	4812.4	4819.5	4825.3	3837	3843.9	3849.7

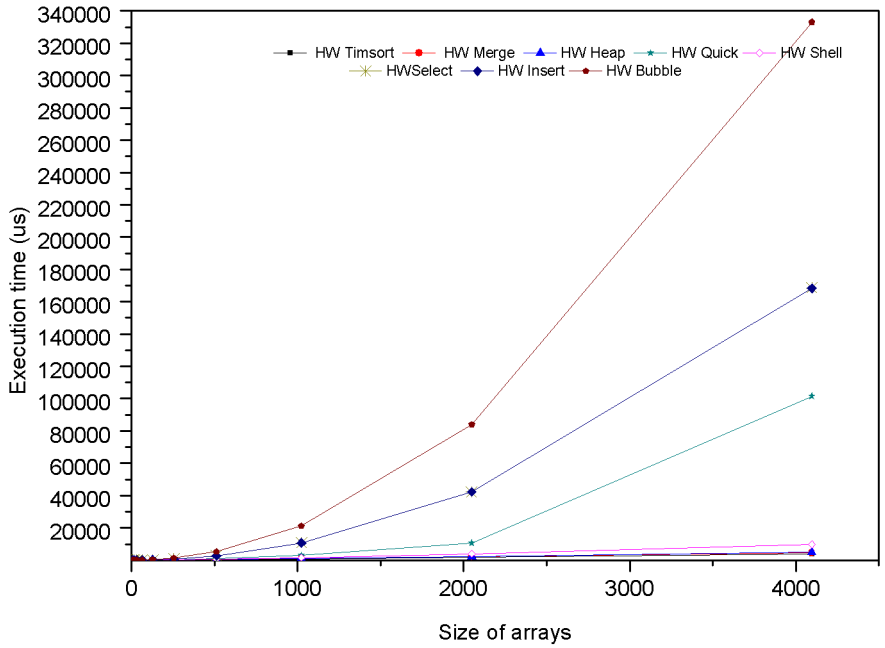


FIGURE 4.18 – Comparaison entre les algorithmes de tri en termes de temps d'exécution avec des données codées sur 32 bits

Par la suite, nous calculons le moyenne de temps d'exécution des algorithmes de tri pour différentes tailles de tableaux allant de 8 à 4096 éléments et pour 47 permutations qui sont générés en utilisant la méthode de Lehmer et codés en 4 et 8 octets. Les tableaux 4.1 et 4.2 présentent les temps d'exécution minimum, moyenne et maximum pour chaque algorithme avec des données codées sur 32 bits et 64 bits pour N<64 et N>=64. La Figure 4.18 montre le temps d'exécution des algorithmes de tri dans le cas moyenne en considérant des données codées sur 4 octets. Nous remarquons que si N<64 alors les courbes ne sont pas visible comme montre la figure 4.18. Donc, la figure 4.16 et la figure 4.17 montre un zoom sur la partie où N< 64 pour bien interpréter les courbes. Par la suite, nous comparons la moyennes des temps d'exécution des algorithmes de tri pour N >=64 dans les deux cas : données codées sur 32 bits et données codées sur 64 bits. Les résultats obtenus dans le tableau 4.2 et les figures 4.19 et 4.20 montrent que l'algorithme BubbleSort, InsertionSort, SelectionSort et QuickSort ont un temps d'exécution très élevé avec 32 et 64 bits. Par conséquent, nous comparons seulement les quatre autres algorithmes pour choisir les meilleurs algorithmes en termes d'exécution et d'écart-type. La figure 4.21 et la figure 4.19 présentent la moyenne des temps d'exécution pour les algorithmes ShellSort, HeapSort, MergeSort et TimSort pour N>=64. Nous prenons par exemple, lorsque N = 4096, le temps d'exécution était de 3756 us, 4734,9 us, 4848,5 us et 9756,4 us pour Timsort, Mergesort, Heapsort et Shellsort respectivement. Pour cela, nous observons que l'algorithme TimSort est 1.12x-1.21x, 1.03x-1.022 et 1.15x-1.61x plus rapide que MergeSort, HeapSort et ShellSort respectivement. Par la suite, nous calculons l'écart-type pour différents algorithmes de tri lorsque N<=64 et N>64. Les tableaux 4.3 et 4.4 montrent l'écart-type des algorithmes de tri qui est presque constant. Puisque nous n'avons pas pris en considération les algorithmes bubbleSort, InsertionSort, SelectionSort et QuickSort si N> 64, nous comparons les algorithmes de tri en termes de l'écart-type et

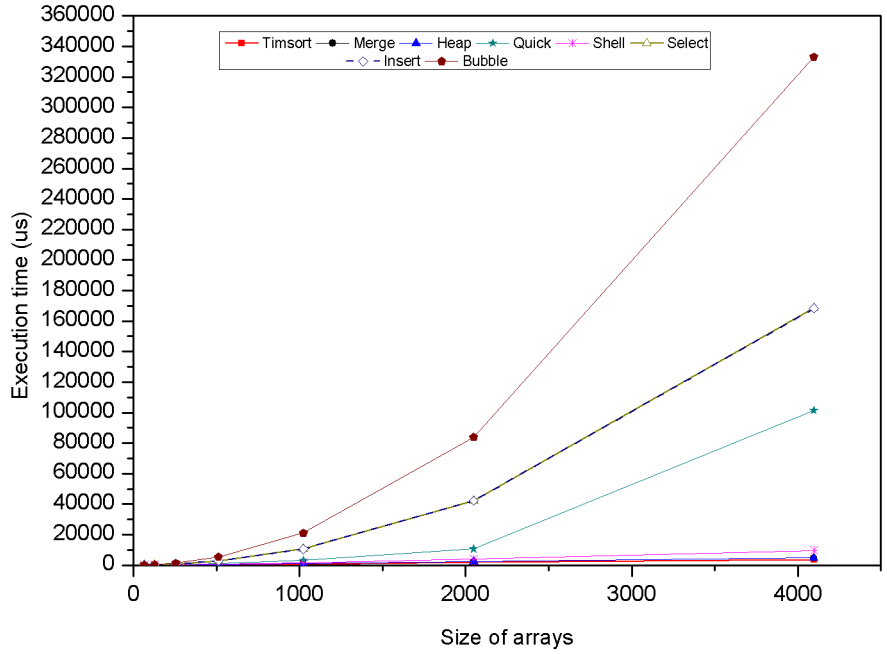


FIGURE 4.19 – Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 32 bits

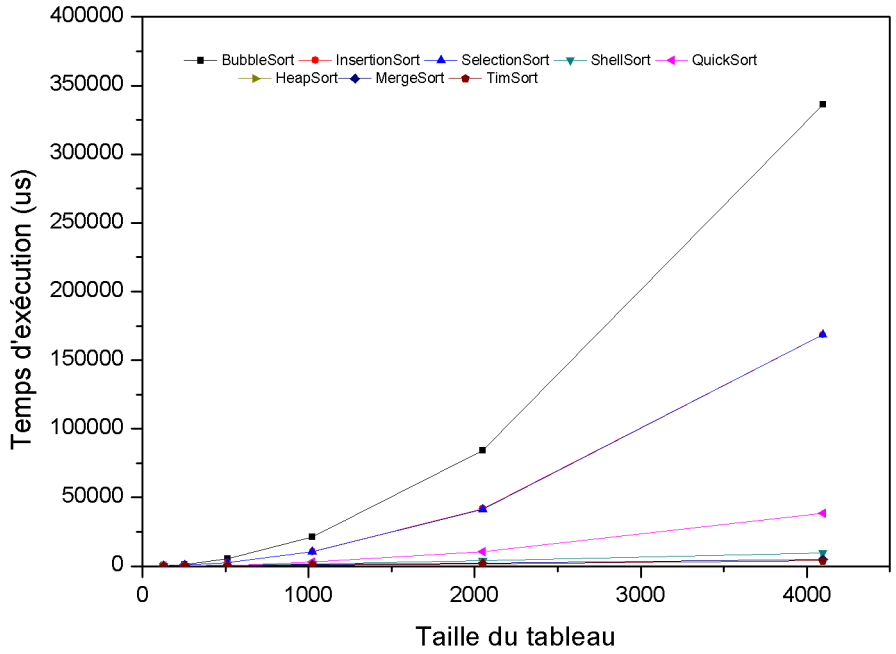


FIGURE 4.20 – Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 64 bits

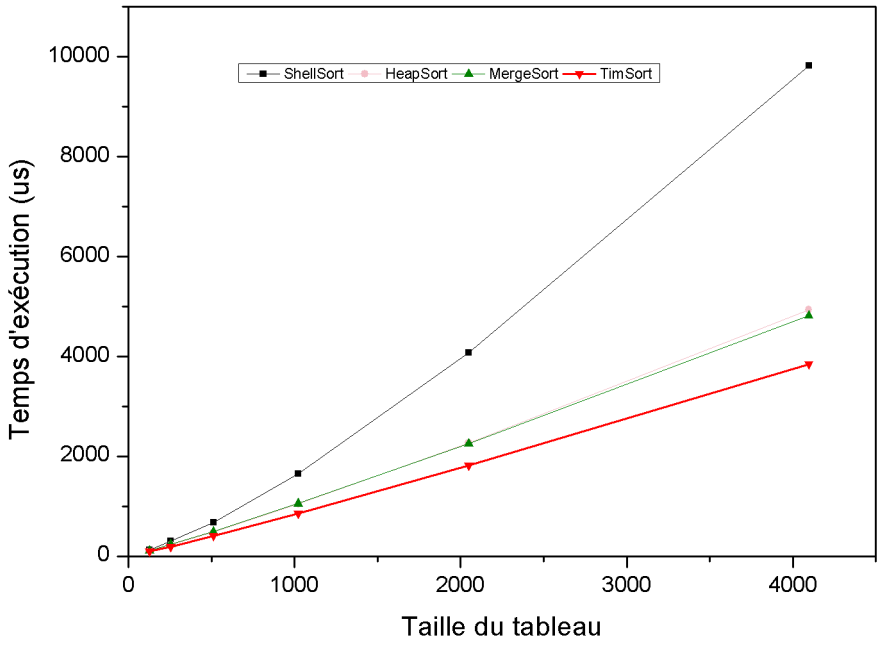


FIGURE 4.21 – Comparaison entre les algorithmes de tri si N >64 pour des données codées sur 64 bits

Tableau 4.3 – Écart type des algorithmes de tri (N<64)

	BubbleSort	InsertionSort	SelectionSort	QuickSort	HeapSort	MergeSort	ShellSort	TimSort
8	3.47244	3.474	3.485	3.504	3.486	3.491	3.4752	3.471
16	3.4789	3.481	3.507	3.4849	3.4777	3.483	3.4742	3.4712
32	3.474	3.474	3.476	3.4757	3.475	3.4749	3.4763	3.4731
64	3.475	3.4743	3.4762	3.4739	3.474	3.475	3.4747	3.4724

de temps d'exécution entre HeapSort, MergeSort, ShellSort et TimSort. Par conséquent, nous remarquons que la variation temporelle ne dépend pas de la taille de données.

En outre, nous remarquons que le temps d'exécution dans l'implémentation matérielle est réduit par rapport à l'exécution dans le processeur Intel en considérant la même fréquence. Par exemple, lorsque N = 2048, le temps d'exécution de l'implémentation matérielle était de 1815 us (50 MHz) et l'implémentation logicielle était de 243134 ns (1.33 GHz) pour Timsort. L'étude des performances des différentes approches SW/HW développées a montré que BubbleSort, InsertionSort et SelectionSort ont un temps d'exécution important pour un grand nombre d'éléments. En outre, nous remarquons que l'algorithme Merge-

Tableau 4.4 – Écart type pour les meilleures algorithmes de tri avec N >=64

	HeapSort	MergeSort	ShellSort	TimSort	HeapSort	MergeSort	ShellSort	TimSort
128	3.475	3.4752	3.4745	3.4736	3.486	3.491	3.4752	3.471
256	3.474	3.4737	3.4758	3.4731	3.4777	3.483	3.4742	3.4712
512	3.6695	3.707	3.472	3.4714	3.475	3.4749	3.4763	3.4731
1024	4.082	3.479	3.56	3.4714	3.474	3.475	3.4747	3.4724

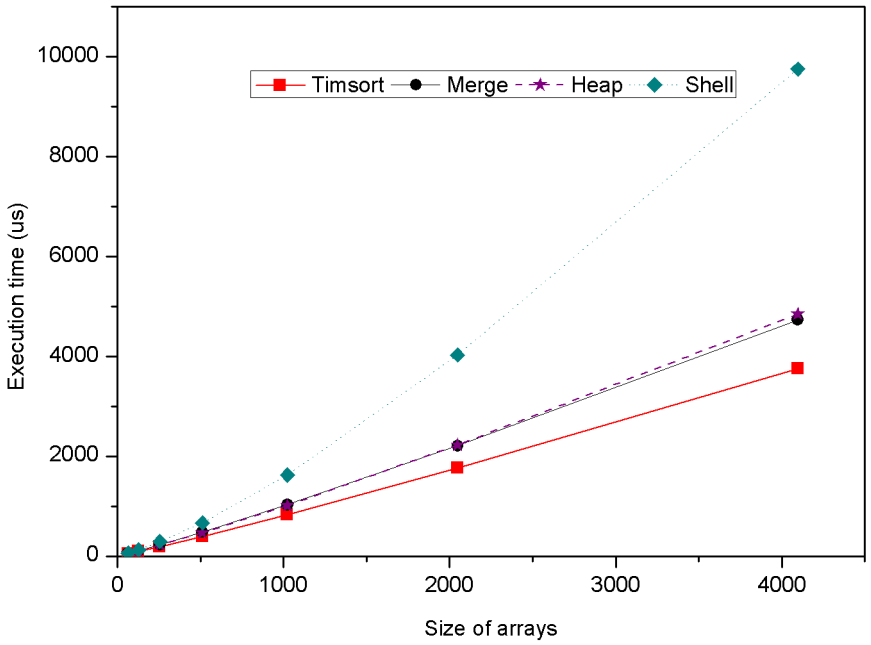


FIGURE 4.22 – Comparaison de temps d'exécution entre TimSort, HeapSort, ShellSort et MergeSort pour N>=64 et avec des données codées sur 32 bits

Sort est le meilleur algorithme dans la partie logicielle et TimSort dans la partie Hardware lorsque N>= 64 ; Sinon, nous notons que InsertionSort est très rapide par rapport aux autres algorithmes fonctionnant sur le processeur et SelectionSort est plus rapide s'exécutant sur une plateforme matérielle.

Tableau 4.5 – Ressources d'utilisation pour les algorithmes de tri

	BubbleSort			InsertionSort			SelectionSort			ShellSort			QuickSort			HeapSort			MergeSort			TimSort		
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM
8	242	121	1	219	131	0	336	203	1	356	224	0	733	541	2	524	444	0	955	877	0	5299	2480	35
16	176	136	1	226	139	0	333	214	1	355	232	0	750	552	2	526	459	1	985	883	0	5326	2498	35
32	189	136	1	187	115	0.5	351	225	1	314	208	0.5	776	564	2	541	474	1	856	809	1.5	5293	2482	35
64	199	149	1	201	123	0.5	363	236	1	319	216	0.5	814	608	2	556	489	1	871	815	1.5	5320	2499	35.5
128	215	162	1	212	131	0.5	373	247	1	337	224	0.5	820	588	2	583	504	1	885	821	1.5	5487	2516	35.5
256	228	175	1	221	139	0.5	365	258	1	345	232	0.5	835	600	2	606	519	1	893	827	1.5	5468	2535	35
512	238	188	1	239	147	0.5	394	269	1	359	240	0.5	841	612	2	611	534	1	925	849	1.5	5494	2552	35.5
1024	245	201	1	248	155	1	407	280	1	365	248	1	860	624	3	651	551	1	936	855	3	5545	2569	36
2048	259	216	2	265	163	2	418	293	2	367	258	2	909	638	6	669	566	2	942	861	6	5668	2586	38
4096	291	229	4	274	171	4	434	304	4	384	266	4	976	682	15.5	689	581	4	954	867	12	5841	2602	44

Le tableau 4.5 montre l'utilisation des différentes ressources pour le tri à bulles, le tri par insertion, le tri par sélection, le tri par tas, le tri par fusion, ShellSort et TimSort. Nous montrons que TimSort consomme plus de ressources que d'autres algorithmes. TimSort a consommé 44% de plus pour Slice LUT dans l'implémentation matérielle et environ 9% pour Slice Register. Nous pouvons conclure que si un algorithme est meilleur en termes de temps et d'écart type alors il consomme plus de ressources sur FPGA.

4.5 Conclusion

Au début de ce chapitre, nous avons proposé une optimisation sur les implémentations Hardware des algorithmes de tri pour améliorer les performances en termes de temps d'exécution et d'écart type en considérant plusieurs tailles de données (8 jusqu'à 4096) codées sur 32 et 64 bits. Aussi, nous avons utilisés l'outil de synthèse haut niveau HLS pour générer la conception RTL. Cependant, nous avons utilisé un algorithme de tri multi-critères qui contient plusieurs actions en ligne et différents critères en colonnes.

L'évaluation des performances sur la carte Zedboard a montré que l'algorithme de tri à bulles, le tri par insertion, tri par sélection et le tri rapide ont un temps d'exécution élevé. En outre, TimSort est 1.12x-1.21x plus rapide que MergeSort, 1.03x-1.22x plus rapide que HeapSort et 1.15x-1.61x plus rapide que ShellSort quand N> = 64 et en utilisant l'implémentation matérielle optimisée avec de nombreuses permutations / vecteurs. En revanche, lorsque N <64, le tri par sélection est 1,01x-1,23x plus rapide que les autres algorithmes de tri si N <64 ;

Chapitre 5

Conclusion générale

5.1 Synthèse

Dans cette thèse, nous avons développé une bibliothèque de modules IP pour les algorithmes de tri en utilisant l'approche de conception haut niveau. Ces IPs sont par la suite intégrés dans un environnement SW/HW dans le but de les comparer en tenant compte de la surface, du temps, d'écart type et de la consommation.

La complexité énorme de cette application rend la contrainte temps réel un défi difficile à surmonter. De ce fait, le chapitre 2 montre une étude de complexité de ces algorithmes qui est réalisée sur une machine dotée d'un système d'exploitation Linux ayant un processeur Intel I5 de fréquence 1.33 GHz. Nous avons aussi utilisé dans cette thèse un nombre important d'éléments varie entre 8 et 4096 et nous avons généré les différentes permutations à l'aide de la méthode de Lehmer. Les résultats montrent que l'algorithme MergeSort est plus rapide en termes de temps et de l'écart type si N>64. De plus, nous avons constaté que nous avons intérêt à réaliser des Boxplots pour interpréter les résultats avec 1000 réplications sur une même échelle.

Dans le chapitre 3, nous avons répété le même travail sous l'environnement *Standalone* en utilisant le processeur ARM Cortex A9. Les différentes analyses développées ont prouvé que le temps moyen de l'algorithme de tri ShellSort pour le mode Standalone est plus rapide que les autres si le nombre des données est supérieur à 64. Par ailleurs, nous avons calculé l'écart type et la consommation d'énergie des différents algorithmes pour visualiser et comparer les différentes permutations des différents algorithmes. Nous avons tracé aussi dans ce chapitre les Boxplots pour bien visualiser les variations temporelles.

En se basant sur toutes ces analyses, différents accélérateurs matériels ont été proposés dans le chapitre 4 avec une méthodologie de conception haut niveau pour l'implémentation des algorithmes de tri. Cette méthodologie de conception adopte la synthèse de haut niveau avec l'outil Vivadeo HLS pour la génération automatique du niveau RTL. Ce flot de conception permet une synthèse rapide des modules matériels. Nous avons aussi considéré plusieurs optimisations (pipeline, parallélisme..) pour améliorer les performances architecturales des IPs matériels développés.

Par la suite, nous avons besoin de ces IPs afin de concevoir des architectures SW/HW pour les algorithmes de tri avec la méthodologie haut niveau. La plateforme cible adoptée pour ces implémentations est la carte Zedboard à base du processeur ARM Cortex A9

de fréquence 667 MHz. En effet, le processeur ARM prend en charge l'exécution de la partie logicielle de l'implémentation proposée. Pour la partie matérielle, elle est chargée dans la logique programmable de l'FPGA. La communication entre les deux modules est assurée par le bus interne d'interconnexion de type AXI4-stream. Ces derniers permettent la transmission d'une quantité illimitée de données vers la mémoire externe DDR à travers des accès DMA. Pour gérer la lecture/écriture des données triées en sortie, nous avons eu recours à la bibliothèque "File Fat System" de SDK afin d'accéder en mode *Standalone*.

Par la suite, nous avons développé différentes implémentations hardware optimisées des algorithmes de tri pour améliorer les performances en termes de temps d'exécution et écart type en considérant plusieurs taille de données (8 jusqu'à 4096) codées sur 32 et 64 bits. Nous avons aussi utilisé l'outil de synthèse haut niveau HLS pour générer la conception RTL. Cependant, nous avons utilisé un algorithme de tri multi-critères qui contient plusieurs actions en ligne et différents critères en colonnes. L'évaluation des performances sur la carte Zedboard a montré que l'algorithme de tri à bulles, le tri par insertion, tri par sélection et le tri rapide ont un temps d'exécution élevé. En outre, TimSort est 1.12x-1.21x plus rapide que MergeSort, 1.03x-1.22x plus rapide que HeapSort et 1.15x-1.61x plus rapide que ShellSort quand N> = 64 et en utilisant l'implémentation matérielle optimisée avec de nombreuses permutations / vecteurs. En revanche, lorsque N <64, le tri par sélection est 1,01x-1,23x plus rapide que les autres algorithmes de tri si N <64 ;

Par conséquent, nous avons remarqué que l'écart type est presque constant et la valeur de la variabilité est inférieure à 1. Dans ce cas, nous n'avons pas besoin des Boxplots pour l'interprétation des résultats. Pour cette raison, nous pouvons conclure que les variations temporelles ne dépendent pas de la taille des données.

5.2 Perspectives

Nous terminons cette conclusion en évoquant les différentes perspectives de recherche que nous envisageons d'aborder dans le futur.

5.2.1 Perspectives à court et à moyen terme

Implémentation hardware de l'algorithme d'aide à la décision

Nous envisageons le développement matériel de l'algorithme d'aide à la décision temps réel utilisé dans notre laboratoire de recherche sur FPGA pour améliorer les performances et aussi pour faciliter la prise de décision dans les systèmes embarqués. Pour cela, nous intégrons l'algorithme TimSort dans ces applications pour trier les critères de performances.

Parallélisation et implémentation hardware de l'algorithme de planification de plan de vol en 3D

L'enjeu de planification de vol à bord d'un hélicoptère en tenant compte des différents paramètres environnementaux constitue un facteur clé dans le secteur aéronautique afin d'assurer une mission en toute sécurité avec un coût réduit. Nous réfléchissons à paralléliser

et implémenter les algorithmes de planification de plan de vol en 3D proposés par notre laboratoire LAMIH sur des architectures hétérogènes CPU/FPGA qui offrent de hautes performances et une adaptabilité aux contraintes temps-réel de l'application.

5.2.2 Perspectives à long terme

Création d'un environnent de simulation temps réel dédié aux transports intelligents

Nous planifions de développer un modèle de simulation permettant les migrations hardware/software à la volée lorsque la charge des ressources est proche de la limite acceptable. Nous intéresserons dans cette partie à la phase de simulation temps réel dédiée aux transports intelligents. Tout d'abord la création d'un simulateur permet une simulation du comportement d'un ou plusieurs éléments/composants relatifs aux transport intelligent comme par exemple une fonction avionique (évitement d'obstacles), un hélicoptère (path planning). Ensuite, cette phase permet aussi la vérification et la validation des applications de transport. Par conséquent, ces applications se réfèrent à un traitement de données intensif et à un traitement d'un grand nombre de solutions en temps réel. Ce qui doit aider à la décision en temps réel souple. Finalement, nous validerons cette plateforme logicielle sur des cas concrets issus de l'avionique pour le calcul de plans de vol en 3D.

Annexe A

Annexe

Cette annexe est consacrée à la description détaillée des représentations graphiques avec les boites à moustaches pour les différents algorithmes de tri qui ne sont pas présenté dans le chapitre 2 et 3. D'une part, nous négligeant les données aberrantes dans cette partie. D'autre part, cette partie est divisée en deux : Une vue globale sur la représentation graphique et les résultats obtenus pour les algorithmes de tri en l'exécutant sur CPU et sur ARM.

A.1 Vue globale sur les représentations graphiques avec Boite à moustaches

Dans les représentations graphiques de données statistiques, la boîte à moustaches est un moyen rapide de figurer le profil essentiel d'une série statistique quantitative. Elle est une invention de TUKEY (1977) [116] pour représenter schématiquement une distribution. La boîte à moustaches utilise six valeurs qui résument des données : le minimum, la moyenne, les trois quartiles Q1, Q2 (médiane), Q3 et le maximum. Les quartiles Q1, Q2, Q3 sont les éléments essentiels de ce graphique.

Les étapes pour construire les quartiles et l'intervalle interquartile (IQR) sont :

– Calculer les valeurs des quartiles et IQR. Nous prenons un exemple simple pour comprendre comment trouver les valeurs de quartiles. Soit la série des 9 valeurs ordonnées : 1, 3, 4, 5, 6, 7, 9, 10, 15. La médiane Q2 partage la série en deux groupes d'effectifs égaux, ce qui donne : Q2=6. La médiane est représentée par la ligne dans la boîte. La médiane est une mesure courante du centrage des données. La moitié des observations sont inférieures ou égales et la moitié des observations lui sont supérieures ou égales. Le Quartile Q1 repartage le groupe du bas (5 valeurs inférieures) en deux groupes d'effectifs égaux, ce qui donne : Q1=4. Le Quartile Q3 repartage le groupe du haut (5 valeurs supérieures) en deux groupes d'effectifs égaux, ce qui donne : Q3=9. L'intervalle interquartile (InterQuartile Range) est utilisé comme indicateur de dispersion. Il correspond à 50% des effectifs situés dans la partie centrale de la distribution.

$$IQR = Q3 - Q1 \tag{A.1}$$

– Lecture d'une boîte à moustaches.

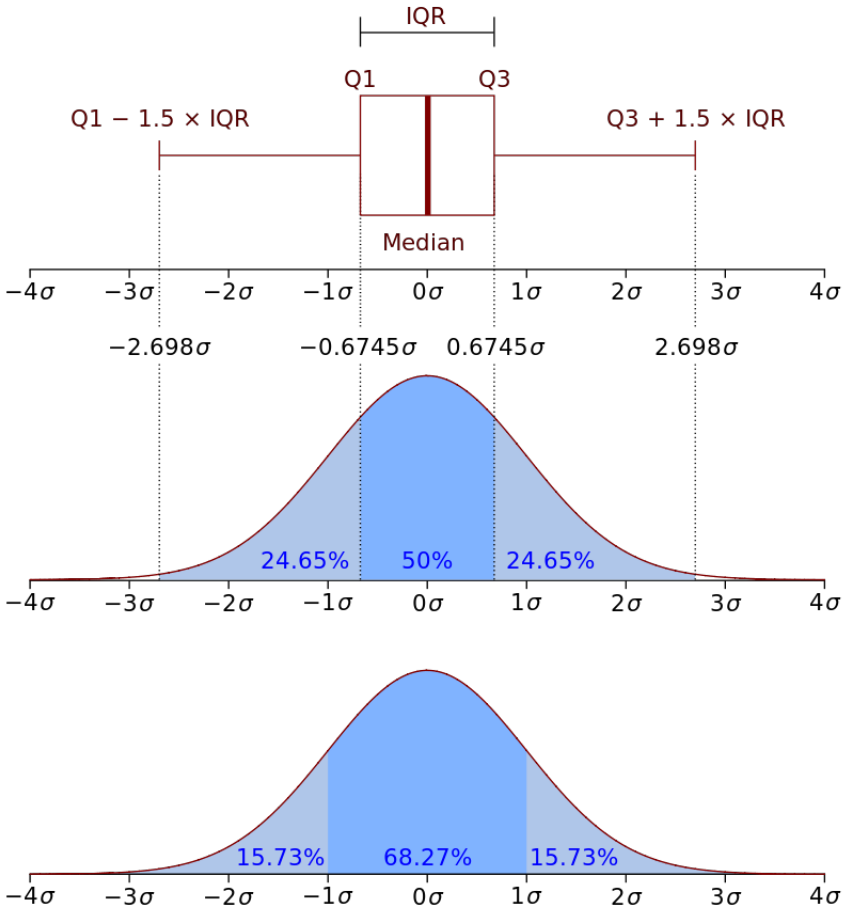


FIGURE A.1 – Informations nécessaires pour caractériser un Boxplots

– Délimitation des longueurs des moustaches (valeurs adjacentes) : L'extrémité de la moustache inférieure est la valeur minimum dans les données qui est supérieure à la valeur frontière basses exprimé par l'équation :

$$Q1 - 1,5 * (Q3 - Q1) \tag{A.2}$$

L'extrémité de la moustache supérieure est la valeur maximum dans les données qui est inférieure à la valeur frontière hautes exprimé par l'équation :

$$Q3 + 1,5 * (Q3 - Q1) \tag{A.3}$$

Par construction, nous avons 50% des observations dans la boite à moustache car Q1=25% et Q3=75%. Donc Q3-Q1 fait 50% des observations. Pour cela, si la dispersion de l'échantillon des observations a les "bonnes propriétés" alors 1.5*(Q3-Q1) représente donc un écart situé entre la médiane + ou - 2 à 3 σ. Ceci est illustré par la figure A.1. Donc, il ne faut pas y avoir plus d'environ 100-99% à 100-95%, soit 1% à 5% des observations en dehors de l'intervalle situé au niveau de la médiane +/- 2 à 3 σ. C'est précisément ce qu'approxime le 1,5*(Q3-Q1) et dans ce cas, le pourcentage d'outliers par valeurs supérieures varie entre 1% à 5% pour que la population suit une loi normale.

A.2 Résultats expérimentaux

Dans ce chapitre, nous envisageons de tracer des Boxplots pour les algorithmes MergeSort, TimSort, ShellSort et HeapSort sur le processeur CPU et sur le processeur ARM pour donner une idée sur la dispersion des données. Cette section a comme objectif d'analyser expérimentalement les Boxplots et les variations temporelle en fonction de N.

A.2.1 Exécution sur le processeur CPU

Dans cette partie, nous expliquons les résultats de la complexité indiquée dans le chapitre 2 sous forme des Boxplots pour bien visualiser et interpréter les résultats sur le processeur CPU.

MergeSort

TimSort

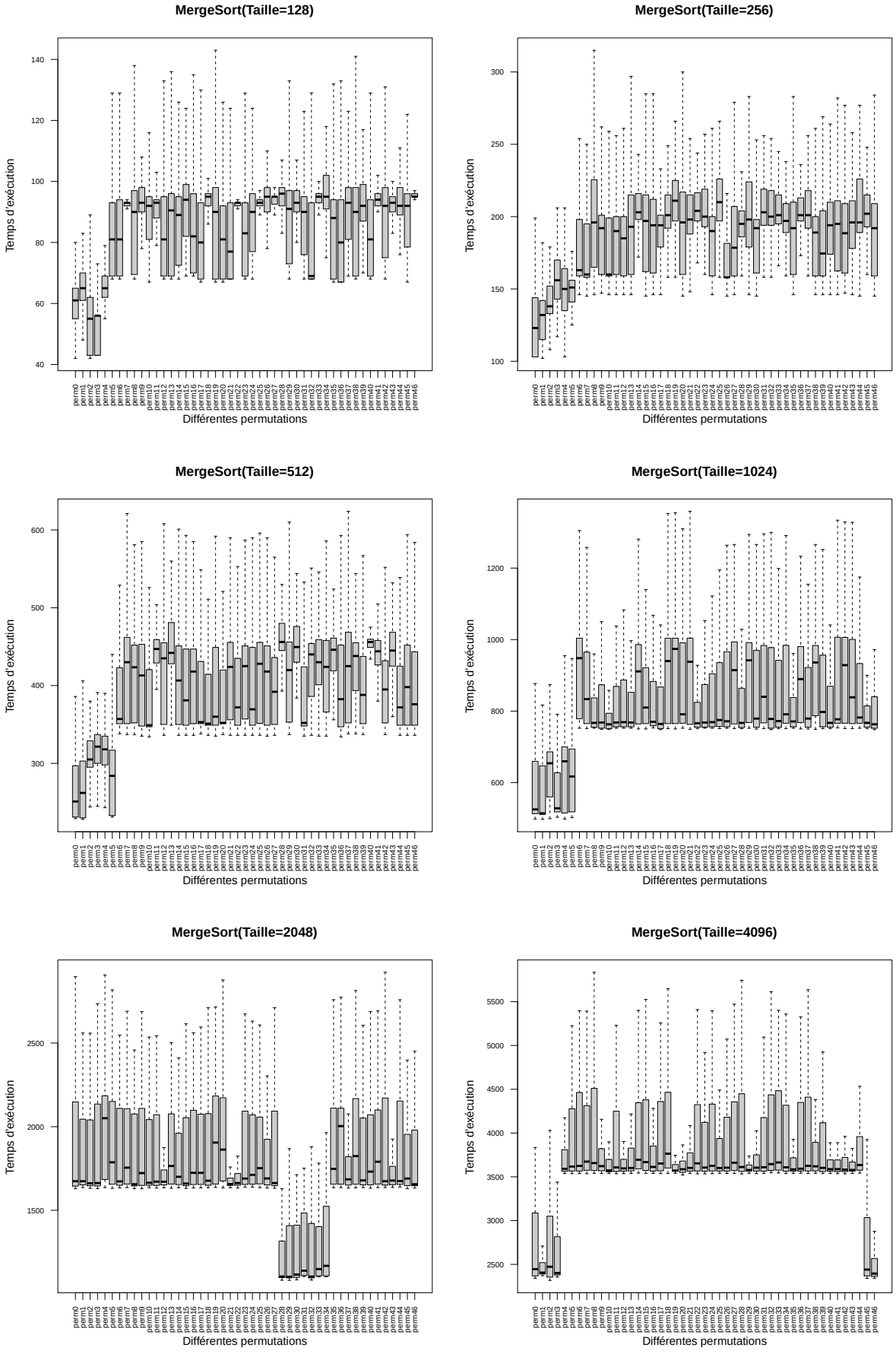


FIGURE A.2 – Boxplots de l'algorithme MergeSort sur CPU

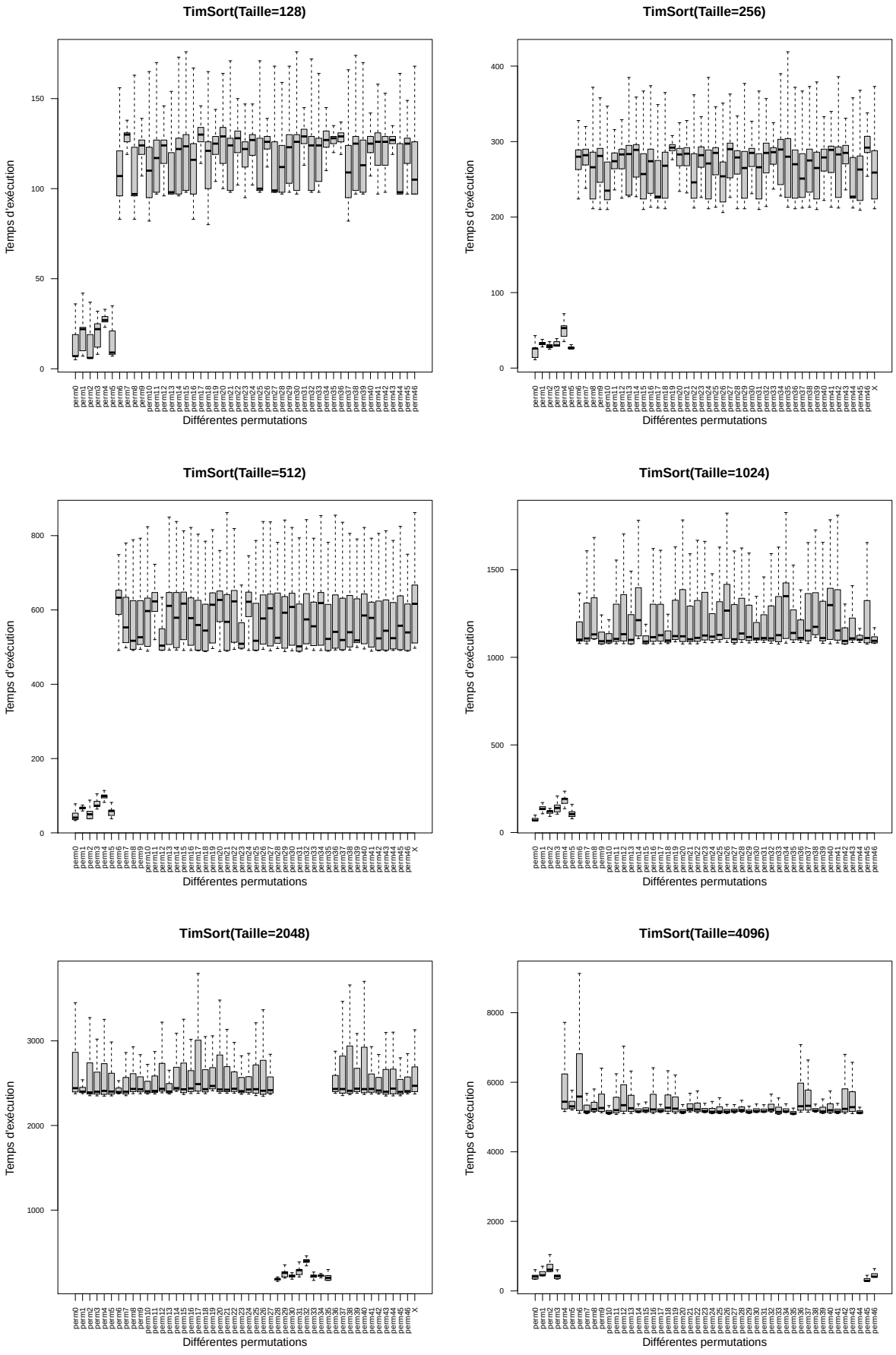


FIGURE A.3 – Boxplots de l'algorithme TimSort sur CPU

Tableau A.1 – Critères de performance pour l'algorithme TimSort sur CPU

	128					256					512					1024					2048					4096				
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	5	7	12.32	18.111	19	246	12	11	14	26	23.612	26	584	12	33	36	41	45.322	53	196	17	68	68	69	76.59	81	181	13	2375	2397	2411	2624.077	2863.75	3740	466.75	331	344	415	405.411	451	913	107
perm1	7	10	22	18.111	23	179	13	15	31	33	38.772	34	1002	3	48	65	67	71.048	69	1065	4	99	131	134	195.822	147	9391	16	2376	2391	2400	2511.316	2451	5423	60	433	447	451	497.033	555.5	852	108.5
perm2	6	6	6	12.042	19	246	13	13	27	29	29.105	31	539	4	38	38	50	55.505	58	576	20	92	110	122	121.877	123	1923	13	2352	2372	2391	2582.994	2737	8173	365	535	557	612	667.96	765	2007	208
perm3	8	12	22	20.88	25	667	13	17	29	30	32.895	35	109	6	64	71	73	153.899	85	28408	14	105	118	140	255.331	157	12393	39	2350	2371	2389	2562.393	2630.5	7086	2591.5	339	353	420	422.087	458	1266	105
perm4	11	26	27	52.039	29	12249	3	35	42	53	53.671	56	3667	14	67	94	99	134.327	102	12420	8	136	166	192	184.015	195	519	29	2347	2368	2409	2593.658	2729.5	6165	361.5	5157	5223.75	5440.5	5881.578	6238.25	19060	1004.5
perm5	7	8	9	14.307	21	97	13	13	26	26	26.173	28	817	2	38	47	59	56.596	61	745	14	78	91	105	112.74	119	2691	28	2350	2370	2401	2562.079	2616.25	7728	246.25	5200	5240	5307	5584.804	5454	12793	214
perm6	83	96	107	109.564	121	671	25	209	263	280	297.774	289	7744	26	491	588	633	693.657	653	8694	65	1079	1092	1101	1179.572	1202.25	3788	110.25	2373	2383	2395	2511.368	2443	10482	60	5108	5187	5589	6023.005	6818.75	16213	1631.75
perm7	98	126	130	302.232	131	24206	5	211	269	282	285.368	290	1829	21	498	512	553	574.799	634	780	122	1077	1092	1107.5	1202.573	1309.5	11321	217.5	2352	2372	2389	2531.416	2568	5691	190	5093	5118	5156	5378.356	5343.25	8280	225.25
perm8	83	96	97	110.622	123	485	27	211	224	266	263.167	286	2597	62	491	494	517	572.946	625	3224	131	1103	1106	1131	1227.929	1340	12243	234	2376	2400	2432	2593.393	2608.75	7111	2026.75	5136	5166	5226.5	5528.692	5424.25	10392	258.25
perm9	96	119	124	130.065	127	1480	8	210	246	281	277.672	291	1554	4.5	494	507	526.5	569.079	625	4439	118	1074	1077	1092	1155.039	1143.5	3905	66.5	2375	2398	2428	2586.229	2575.25	13266	177.25	5125	5161	5259.5	5579.395	5659.25	20372	498.25
perm10	82	95	110	112.189	123	301	28	210	223	235	251.728	273	831	50	489	502	597	580.013	632	914	130	1078	1082	1094	1141.614	1135.25	1731	50.25	2371	2385	2403.5	2545.807	2522.25	5492	137.25	5072	5095	5125	5282.31	5189.25	9208	94.25
perm11	97	98	117	116.806	127	208	29	208	264	274	279.288	285	1554	21	491	596	623	637.844	647	12690	51	1077	1091	1105	1188.388	1302.25	3834	211.25	2373	2390	2408	2576.997	2585.25	12323	195.25	5072	5119	5197.5	5481.662	5567	15267	448
perm12	96	114	124	152.11	127	17465	13	211	264	283	279.534	290	3286	26	491	492	504	536.812	549	1170	57	1078	1093	1132	1231.041	1357.25	4022	264.25	2386	2406	2423	2626.585	2733	5685	327	5097	5166	5340	5635.417	5926.75	14400	760.75
perm13	97	97	98	109.18	120	329	23	227	229	283.5	273.751	295	1501	66	492	507	611	503.4	647	3018	140	1074	1077	1099.5	1171.008	1242.75	1892	165.75	2372	2386	2401	2563.245	2495.25	9674	109.25	5104	5158	5253	5518.566	5624.5	18023	466.5
perm14	96	97	122	117.818	128	318	31	227	253	289	284.903	296	599	43	492	498.75	579	651.971	647	11451	148.25	1107	1122	1212	1296.337	1397.25	10233	275.25	2403	2420	2441.5	2608.514	2687.75	5597	267.75	5124	5198	5161	5326.321	5237.25	8145	99.25
perm15	98	99	123.5	126.865	130	2432	31	210	224	257	260.095	284	3338	60	491	530	617	700.377	648	11040	128	1076	1078	1090	1132.242	1122.25	3889	44.25	2366	2388	2427.5	2603.384	2733.5	9165	347.5	5126	5148.75	5187	5422.325	5364	14176	115.25
perm16	83	97	116	118.966	125	3774	28	213	231.75	274	375.01	290	3259	58.25	491	505	578	582.303	632.25	1625	127.25	1076	1091	1115	1203.041	1302.5	3240	211.5	2378	2401	2438	2607.212	2647.75	6460	246.75	5103	5144	5218	5525.107	5655	9149	511
perm17	100	126	130	137.387	134	3316	8	212	225	227	258.017	275	7257	50	490	492	559.5	572.083	626	921	134	1078	1092	1126	1246.15	1302.25	13670	210.25	2379	2406	2489	2680.766	3007.5	6074	601.5	5129	5151	5177.5	5332.724	5238	16718	87
perm18	80	100	121	118.518	126	515	26	211	225	268	263.609	286	1137	61	488	490	544.5	558.859	615.25	860	125.25	1078	1086	1096	1132.534	1151	2228	65	2376	2397	2429	2588.162	2659.25	9026	262.25	5093	5166	5265	5577.523	5604.5	30537	468.5
perm19	98	119	125	153.355	129	10294	10	226	288	292	328.734	296	16361	8	496	511	614	595.739	646	3439	135	1088	1102	1121	1217.383	1325	3813	223	2411	2433	2467	2653.936	2683	8236	250	5103	5149	5241	5558.033	5578	21350	429
perm20	100	114	129	129.066	134	589	20	211	268	283	329.471	291	2179	23	488	568.75	627	617.704	651	3408	82.25	1076	1090	1120	1240.176	1386.25	9725	296.25	2380	2400	2426	2630.348	2828.75	6003	428.75	5095	5115	5147.5	5353.546	5215.25	10850	100.25
perm21	98	99	124	126.73	128	1681	29	211	268	284	296.822	292	9927	24	480	491	568	612.251	641.25	12694	154.25	1077	1090	1103	1224.336	1292.25	12324	202.25	2380	2400	2423	2698.795	2693.75	16484	2934.75	5142	5177	5231	5528.831	5382.25	14438	205.25
perm22	97	130	128	136.953	132	5055	12	212	225	246	258.713	284	2563	59	494	513	623	643.022	632.25	12782	139.25	1077	1091	1111	1203.228	1324.25	3375	233.25	2381	2402	2454	2586.023	2633.5	7184	2211.5	5124	5154	5217	5463.569	5396.75	7632	242.75
perm23	95	112	122	163.329	126	21375	14	210	266	282	307.069	293	4779	27	495	497	509	550.07	565.25	1963	68.25	1084	1099	1124.5	1310.694	1370.5	164.25	271.5	2369	2387	2406	2569.015	2565.25	5450	178.25	5124	5142	5173	5363.923	5247	10362	105
perm24	99	118.75	127	126.898	130	322	11.25	211	224	271	433.325	289	27181	65	491	582	622	627.474	648	7102	66	1083	1097	1118	1206.594	1249.25	10765	152.25	2371	2392	2425	2627.229	2575.5	18730	1834.5	5094	5112	5150	5406.98	5247.5	23005	135.5
perm25	98	99	100	112.446	128	288	20	213	256	285	274.93	292	428	36	490	492	517	566.833	618	8726	126	1087	1101	1128	1203.535	1316.75	2702	215.75	2353	2382	2429	2655.176	2715.5	1314	3333.5	5097	5119	5156	5391.363	5292.75	19127	173.75
perm26	101	122	126	233.196	129	19148	7	206	220	254	251.958	273	1025	53	494	508	577	605.599	640.25	8107	132.25	1085	1107	1266.5	1354.896	1416.25	13094	309.25	2344	2368	2414.5	2588.355	2769.75	4356	401.75	5101	5124	5152.5	5333.917	5220	7626	96
perm27	98	98	99	114.237	126	764	28	226	252	290	397.287	298	12425	46	490	503	604.5	683.717	643	21987	140	1076	1084.75	1103	1185.115	1300.5	2919	215.75	2368	2384	2417	2551.526	2573.75	4730	189.75	5113	5143	5170	5348.784	5229.25	25048	86.25
perm28	97	98	112	115.346	124	1745	26	210	257	279	279.133	288	1287	31	486	510	525	578.743	645.25	2141	135.25	1085	1101	1136	1223.767	1337	3635	236	137	175	186	246.902	189	20149	14	5133	5160	5198.5	5401.335	5290.25	8097	130.25
perm29	98	103	123	123.992	130	1872	27	211	225	265	264.407	287	808	62	488	497	592.5	588.599	636	3024	139	1082	1096	1115.5	1193.028	1297.25	2721	201.25	197	211	258	260.294	271	447	60	5101	5123	5143	5287.413	5199.5	14382	76.5
perm30	97	99	126	122.34	130	343	31	212	267	285	281.211	291	1171	24	490	505	608	604.72	645	8693	140	1084	1099	1107	1175.787	1199	2554	100	164	210	224	224.268	230	806	20	5126	5144	5170.5	5337.732	5238.25	9546	94.25
perm31	98	125	129	140.074	133	2300	8	210	224	266	271.774	284	3600	60	487	489	502	615.201	616	8802	127	1085	1099	1110	1190.508	1242.5	1656	143.5	210	237.75	290	2922.076	299	3569	61.25	5130	5145	5163	5333.823	5229.25	23486	84.25
perm32	98	99	124	126.021	129	1107	30	214	258.75	285	297.511	298	16538	39.25	495	509	574.5	589.737	644	1773	135	1080	1094	1108	1191.882	1293.25	2698	199.25	272	383	396	439.262	414	8480	31	5140	5171	5217	5483.803	5369.25	16121	198.25
perm33	98	104	124	128.233	128	4163	24	210	270	286	428.709	292	16454	22	491	504	566	575.604	621	3162	117	1075	1089	1128.5	1269.247	1346.5	13257	257.5	169	206	223	220.438	231	522	25	5075	5099	5147	5358.514	5284	13550	185
perm34	97	123	127	176.227	132	14642	9	228	243	289.5	299.371	303	4389	60	491	505	618.5	647.109	647	16750	142	1081	1107.75	1349	1294.702	1425	1826	317.25	169	218	225	229.855	231	3083	13	5126	5142	5163	5308.597	5237	14332	95
perm35	106	125	128	157.222	129	8206	4	213	226	280	281.338	304	2629	78	489	491	522	555.858	615	951	124	1085	1100	1139	1212.847	1270.25	4032	170.25	169	171	199	205.427	226	582	55	5066	5086	5100	5223.18	5157.25	14842	71.25
perm36	100	126	129	133.184	131	1653	5	211	225	270	277.516	289	5791	64	492	497	541	583.591	640.25	8088	143.25	1085	1099	1110.5	1180.445	1213.25	1911	114.25	2380	2400	2452	2611.03	2592	12014	192	5140	5165	5310	5640.899	5975.5	9493	780.5
perm37	82	95	109	111.864	124	314	29	212	226	251	250.014	284	1406	58	491	493	519	567.375	631.25	1819	138.25	1079	1094	1152.5	1228.798	1304	9990	270	2353	2381	2451	2621.643	2820.5	5606	439.5	5139	5199	5323	5587.014	5775.25	9344	576.25
perm38	97	99	125	130.176	129	419	30	213	233	275	273.077	290	1395	57	492	500.75	540	572.925	639	2441	138.25	1114	1129	1174	1247.326	1369	2691	240	2365	2380.75	2410	2593.419	2939	4848	558.25	5138	5159.75	5184	5323.122	5245.25	7663	85.5
perm39	97	98	113	123.858	127	2307	29	210	224	265	270.468	286	2545	62	499	512	518	759.618	630	24765	118	1080	1094	1110	1215.191	1319	7050	225	2381	2401	2435	2605.392	2674.75	3948	273.75	5101	5124	5159.5	5330.01	5283	9036	159
perm40	97	120	125	150.744	129	5120	9	210	261	279	282.662	290	1768	29	495	509	585	587.105	643	1144	134	1077	1102	1298	1345.991	1394	9486	292	2367	2387	2450	2684.132	2923.5	17092	536.5	5104	5150.75	5194.5	5451.366	5379.75	11652	249
perm41	97	113	126	126.533	131	308	18	213	259	289	295.556	294	3795	35	489	500	578.5	572.993	621	2900	121	1082	1098	1153	1302.278	1386	10940	288	2372	2393	2452	2621.47	2606.5	21808	215.5	5096	5120	5157	5325.078	5229.25	10834	109.25
perm42	98	113	126	128.373	129	2172	16	212	226	283	415.399	293	19872	67	490	492	523.5	574.536	624	8752	132	1074	1077	1093.5	1147.888	1168.25	2360	91.25	2371	2387	2413	2584.419	2568	9086	181	5103	5146	5234	5601.875	5811	22910	665
perm43	103	125	127	205.612	129	12216	4	212	270.75	285	383.346	295	25520	24.25	489	492	544	570.908	627	1840	135	1084	1099	1108	1193.889	1224	3971	125	2349	2370	2389	2556.923	2661.25	3969	291.25	5099	5159	5284	5562.699	5726.25	18588	567.25
perm44	97	97	98	110.329	125	270	28	212	225	227	252.938	279	1045	54	491	495	524	567.295	620	3009	125	1085	1099	1100	1136.469	1125.25	1684	26.25	2350	2374	2436	2581.279	2665	3637	291	5092	5106	5126	5262.769	5180.25	13494	74.25
perm45	97	114	125	129.57	128	11155	14	209	222	263	259.734	281	760	59	491	493	557.5	667.429	638	7798	125	1075	1083	1110	1199.094	1324	3889	241	2351	2371	2397	2547.518	2542.25	12950	1734.25	5093	5116	5234	5601.875	5811	22910	665
perm46	97	105	113.806	126	293	29	226	285.75	292	368.254	307	11103	21.25	489	491	539.5	603.031	616	7798	125	1079	1082	1094	1135.109	1117	1907	35	2373	2389	2405.5	2562.073	2572.5	5124	1834.5	395	396	409	445.829	497.25	747	101.25

Tableau A.2 – Pourcentage d'outliers pour l'algorithme TimSort sur CPU

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
TimSort	128	7.74	4.71
	256	8.28	4.31
	512	3.2	2.48
	1024	4.47	4.45
	2048	10.98	10
	4096	10.95	10.95

La Figure A.3 et le tableau A.1 présentent les valeurs de la médiane et de la moyenne des différentes permutations avec différentes taille du tableau pour l'algorithme TimSort. Tout d'abord, nous observons un nombre important d'outlier. Pour cela, nous avons calculé dans ce cas les pourcentages d'outliers avec des moustaches supérieures pour l'algorithme TimSort avec des tailles de permutations différentes. Le tableau A.2 montre que les pourcentages des tailles des permutations 128/256/512 et 1024 sont entre 2% et 5%. D'autre part, la boite est à peu près symétrique par rapport à la médiane. Pour cela, l'analyse graphique indique que l'algorithme TimSort suit une loi normale pour N=128/256/512/1024. Par contre, nous remarquons que les pourcentages d'outliers par valeurs supérieures pour l'algorithme ShellSort augmentent à 10% pour N=2048 et 4096 et les boites à moustaches ne sont pas symétrique par rapport la médiane. Pour comparer cet algorithme en termes de la complexité en temps, la figure A.4 montre les variations temporelles de l'algorithme TimSort en fonction de la taille du donnée N. A partir de la figure A.4, nous remarquons

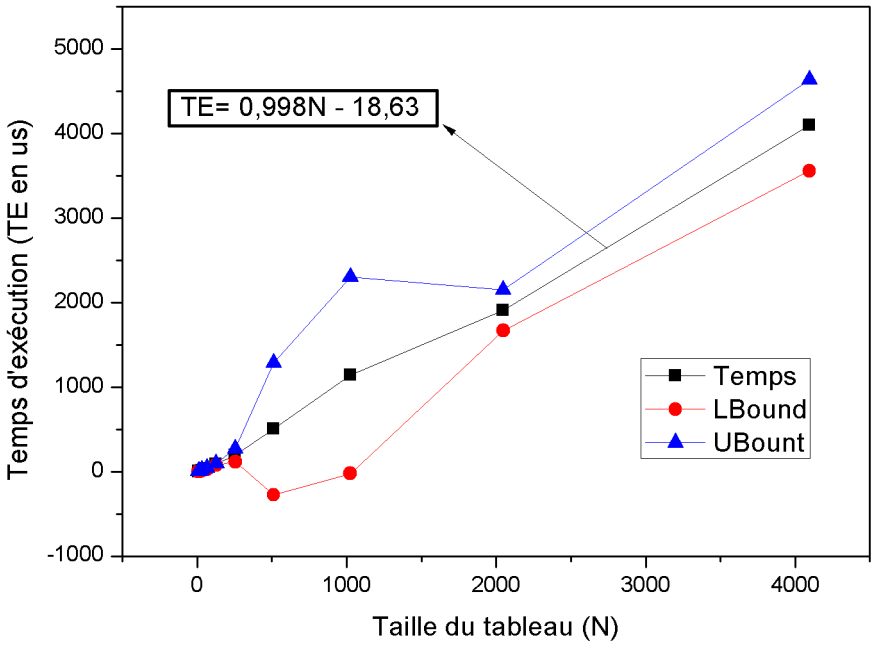


FIGURE A.4 – Variation temporelle de l'algorithme TimSort sur CPU

que la fonction est linéaire et le temps d'exécution augmente si le nombre de donnée augmente.

ShellSort

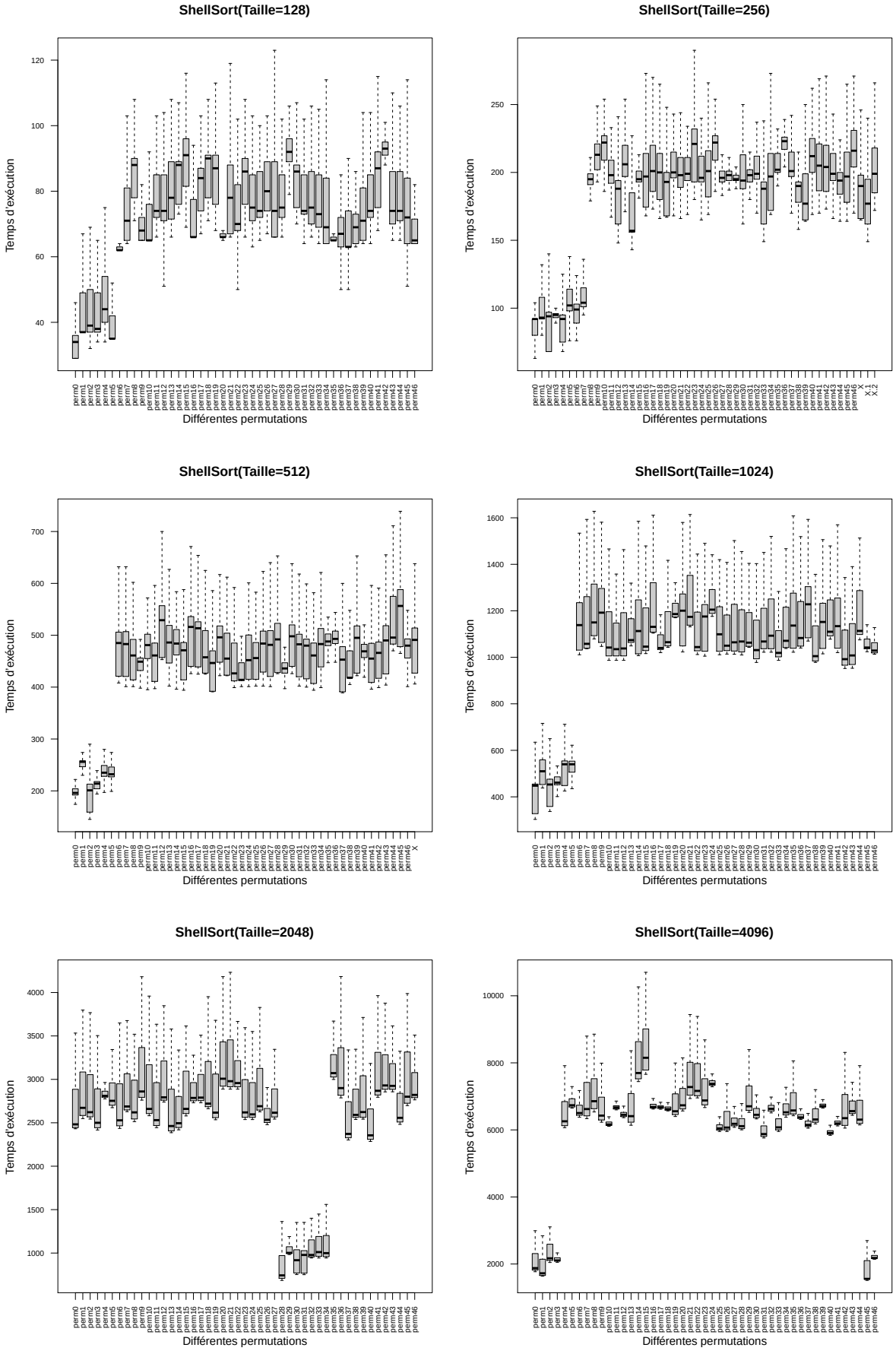


FIGURE A.5 – Boxplots de l'algorithme ShellSort sur CPU

Tableau A.3 – Critères de performances pour l'algorithme ShellSort sur CPU

	128					256					512					1024					2048					4096				
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	17	29	34	34,905	36	194	7	51	80	92	148,199	92	18334	12	135	192	196	205,252	204	3980	12	303	327,75	448	475,72	455	12768	127,25	2431	2444	2482	2669,816	2886	11818	442	1770	1818	1877	2044,103	2311,5	8142	493,5
perm1	37	37	37	42,79	49	256	12	80	92	93	102,971	108	994	16	198	246,75	256	308,391	258	6943	11,25	438	453	510	518,601	559	1632	106	2548	2584	2672	2872,038	3085	12495	501	1636	1658	1723,5	1878,888	2143,25	3368	485,25
perm2	32	37	39	98,999	50	19963	13	68	68	94	92,069	97	2772	29	145	159	201	195,186	213	815	54	337	358,75	453	449,681	476	3091	117,25	2343	2568	2622	2860,971	3054,25	27117	486,25	2045	2105	2170	2313,88	2590	5309	485
perm3	34	37	38	57,694	49	12138	12	71	93	95	203,096	96	17195	3	194	204	214	238,12	218	7655	14	339	432	461	566,923	486	13957	34	2417	2443	2500	2771,051	2891	22854	448	2051	2087	2106	2251,196	2187	10910	100
perm4	34	40	44	53,843	54	1880	14	68	75	92	90,551	95	2013	20	179	228	235	239,236	249	3046	21	425	448,75	540	616,903	554,25	7788	109,5	2775	2794	2810	2922,641	2864	5805	70	6066	6125,75	6256,5	6551,048	6845,25	19639	719,5
perm5	35	35	35	40,382	42	128	7	76	98	102	104,57	114	400	16	170	227	232	240,293	246	2274	19	425	506	540	691,274	553,25	20020	47,25	2674	2702	2755	2880,682	2959,25	3066	257,25	6650	6683	6736	6902,5	6736,5	9060	300,75
perm6	50	62	62	63,935	63	158	1	76	89	99	97,783	103	384	14	408	421	485	723,4	506	29455	85	1011	1031	1138,5	1172,06	1235	8523	204	2435	2465	2528,5	2725,773	29,49	10197	484	6377	6433,75	6502,5	6731,904	6736,5	9060	300,75
perm7	64	65	71	75,42	81	353	16	95	101	104	128,034	115	1494	14	401	420,75	483	621,029	507	28570	86,25	1037	1039	1058	1202,361	1261	10570	222	2028	2648	2688	2908,494	3064	15302	416	6335	6419	6622	6906,812	7415,75	9562	996,75
perm8	71	78	88	86,816	90	322	12	163	191	195	107,708	199	494	8	401	414	461	511,59	492	8726	78	1079	1093	1150	1239,518	1315	10706	222	2513	2544	2620	2907,633	2991	22380	447	6535	6633	6856	7114,086	7525,5	10326	892,5
perm9	65	65	68	72,606	72	663	7	193	202	213	345,354	221	14437	19	364	432	449	468,039	456	8497	24	1047	1064	1192	1198,984	1296	2493	232	2761	2793	2860,5	3161,767	3366,25	18773	573,25	6230	6296	6426,5	6705,219	6976	10230	680
perm10	65	65	65	73,357	76	925	11	174	209	222	228,149	227	3011	18	395	455	481	560,564	502	4602	47	987	1006	1042	1129,81	1196	5489	190	2581	2606	2659,5	3015,208	3168,5	15009	562,5	6106	6132	6235,76	6236	15639	104
perm11	66	72	74	126,393	85	16157	13	165	192	198	218,422	209	12719	17	397	411	461	466,038	485	2087	74	987	1006	1035	1106,149	1147,25	19220	141,25	2444	2468	2538,5	2747,031	2902,25	8663	494,25	6538	6623,75	6667,5	6754,655	6720,25	16224	96,5
perm12	51	71	71	81,585	85	345	14	148	162	188	196,83	194	11245	32	433	457	529	527,031	557	1521	100	987	1006	1038	1099,092	1191,5	1864	185,5	2738	2760	2793	3013,533	3211	9508	451	6395	6396	6450	6628,735	6523	19497	127
perm13	66	71,75	78	86,377	89	30133	17,25	171	197	206	216,816	220	6378	23	402	446,75	486	498,707	519	2821	72,25	1049	1064	1074	1148,082	1165,5	6135	101,5	2386	2406	2462,5	2712,314	2887	15012	481	6130	6229,75	6410	6604,588	7083,75	10919	854
perm14	73	76	88	113,77	89	7753	13	143	156	157	170,076	185	325	20	396	462	484	692,296	511	20586	49	1007	1015	1113	1221,263	1247	6642	232	2419	2445	2495	2660,373	2803	11773	358	7441	7524	7696,5	8004,758	8631,25	13364	1107,25
perm15	69	81,75	91	135,68	96	9063	14,25	163	193	195	231,92	201	8085	8	394	414	471	661,141	486	20029	72	1016	1029	1046	1123,517	1211,75	4160	182,75	2579	2606	2660	2852,207	3093,75	12253	487,75	7658	7785	8151	8497,738	9010,5	22247	1225,5
perm16	66	66	66	74,353	77,25	389	11,25	168	174,75	197	201,701	214	926	39,25	426	440	516	601,79	536	1367	96	1104	1108	1131	1332,931	1321	15749	213	2730	2733	2786	2977,264	2962	13071	209	6622	6650	6684	6903,346	6765,25	16632	1115,25
perm17	67	74	84	141,014	87	10535	13	173	186	201	204,514	220	1140	34	425	439	513,5	593,81	526	2107	87	1020	1034	1040	1108,322	1097	4929	63	2730	2755	2790	2938,009	3056,5	12510	301,5	6617	6643	6670	6776,82	6717	10081	74
perm18	71	78	90	125,878	91	8208	13	166	180	200	201,642	214	310	34	425	427	457,5	476,597	509	3360	82	1043	1049	1065	1175,042	1196,25	9291	147,25	2662	2685	2720,5	2972,269	3206	10605	521	6566	6591	6620	6892,745	6685	43549	94
perm19	68	76	87	86,22	91	1066	15	167	168	193	195,453	200	1483	32	391	392	446,5	441,818	470	693	78	1170	1173	1186	1236,527	1232,25	2542	59,25	2535	2559	2617	2797,068	3061,25	12140	502,25	6399	6457	6556	6857,393	7075	10599	618
perm20	65	66	66	70,323	67	168	1	167	196	200	219,694	215	3691	19	422	448	496	502,699	518	2122	70	1023	1049	1200,5	1250,576	1272,25	13043	229,25	2888	2922	3007,5	3196,686	3431,5	14202	509,5	6565	6624,75	6734	7014,291	7237,25	15715	612,5
perm21	66	67	78	82,543	88	492	21	166	189	198	199,686	211	391	22	422	423	455	474,355	504	3131	81	1131	1138	1174	1356,357	1352,25	18474	214,25	2887	2915	2979	3538,365	3454	28952	539	6936	7037	7278,5	7568,103	8017	16041	880
perm22	50	68	70	78,317	82	4032	14	165	194	199	207,118	211	3042	17	399	412	426,5	461,522	485	10540	73	1012	1028	1044	1113,857	1194,25	3972	166,25	2887	2914	2957,5	3152,719	3215,5	13731	301,5	6955	7019,75	7163	7660,374	7976	41746	956,25
perm23	66	76	86	152,779	90	20617	14	180	193	221	223,132	232	2512	30	401	413	414	433,249	447	1406	34	1005	1026	1175,5	1194,25	1227	9280	201	2537	2563	2619,5	2950,9	2997,25	23894	434,25	6667	6746	6878,5	7247,884	7527	31955	781
perm24	63	71	75	80,666	85	281	14	163	193	196	203,364	212	673	19	401	415	452	521,827	500,25	14305	85,25	1176	1189	1205	1271,383	1291,25	4318	102,25	2537	2562	2707	2808,345	2963	16300	401	7286	7329	7374	7628,538	7466,5	25409	137,5
perm25	65	72	74	104,56	86	9432	14	169	182	201	208,343	216	2505	34	402	415	456	461,559	486	2736	71	1011	1026	1099	1132,291	1217,25	3776	191,25	2826	2846	2901	2983,889	3126,25	15381	480,25	5953	5993,75	6043	6227,451	6154,25	12726	160,5
perm26	67	74	80	214,074	89	26874	15	174	209	222	303,74	227	16207	18	402	429,5	484	482,625	508	1512	78,5	1014	1029	1050	1126,194	1182	4260	153	2478	2503	2534,5	2698,671	2665,25	11204	162,25	5952	6000	6069,5	6386,176	6551,5	15113	551,5
perm27	66	66	74	79,295	89	289	23	163	193	196	226,369	201	6585	8	401	420,75	481,5	484,173	509	2794	88,25	1013	1027	1064,5	1117,429	1228	1502	201	2543	2567	2616	2832,808	2890	20680	323	6077	6121	6181	6485,086	6353	20676	232
perm28	66	72	75	162,066	85	8754	13	163	194	198	259,425	201	12280	7	428	428	492	498,888	523	2604	95	1010	1023	1068	1119,67	1230,25	1633	181,25	685	709	743	920,982	971	23265	262	6003	6037	6111,5	6352,238	6336,25	15120	299,25
perm29	73	89	92	154,491	96	13089	7	172	194	195	336,243	198	21338	4	362	427	436	466,215	447	6813	20	1043	1047	1063	1164,88	1192,25	6980	145,25	983	995	1002	1062,779	1073	14611	78	6523	6582	6704,5	7016,889	7310,25	11427	728,25
perm30	70	75	86	173,685	88	20195	13	162	188	194	206,268	213	3235	25	426	440	498	503,115	520	3803	80	978	993,75	1031	1089,968	1160	3480	166,25	732	770	917,5	949,509	1038	5180	268	6328	6367	6450	6704,279	6640,25	33190	273,25
perm31	64	73	74	78,91	85	319	12	164	193	198	200,592	202	877	9	402	420	482,5	624,993	502	24892	82	1022	1037	1068	1122,602	1211	1893	174	751	768	978	948,622	1028	8916	260	5760	5807	5876	6000,414	6120,25	8145	313,25
perm32	66	70	75	81,66	86	1139	16	167	195	199	205,047	212	897	17	400	416	480	472,518	492,25	1634	76,25	1022	1037	1093	1143,93	1251	1905	214	943	955,75	978	1053,219	1152	3332	196,25	6508	6562	6628	6819,177	6735,25	16426	171,25
perm33	64	69	73	77,633	85	194	16	149	162	188	192,169	193	3330	31	394	407	461	439,742	485	2051	78	987	1002	1019	1074,585	1115	3601	113	943	962	1011	1073,692	1191	1804	229	5954	6001	6075,5	6441,207	6330,25	31153	329,25
perm34	64	64	69	74,306	84	157	20	160	172	197	199,327	214	1367	42	399	439	483	484,594	513	2944	74	1038	1042	1071	1133,736	1215,25	3172	173,25	943	959	990	1084,339	1202	4092	243	6378	6446	6524,5	6775,831	6778,25	17236	332,25
perm35	65	65	65	69,435	66	262	1	176	200	202	325,746	214	12285	14	397	480	488	581,022	502,25	10342	22,25	1022	1039	1136,5	1254,564	1276	11272	237	2909	3027	3072	3309,68	3284,75	15900	257,75	6422	6474	6580	6894,956	7110,5	10770	636,5
perm36	50	63	67	70,38	72	201	9	174	217	223	225,979	226	1373	9	400	484	493	555,221	508	10508	24	1040	1048	1083	1152,116	1240	3342	192	2787	2817	2899,5	3190,125	3364	18525	547	6313	6344	6373	6619,054	6459	20781	115
perm37	50	63	63	73,122	74	897	11	168	197	201	208,212	215	1495	18	389	391	453	677,03	478	44557	87	1070	1084	1228	1221,023	1304	6529	220	2302	2327	2372	2542,525	2743	7573	416	6052	6098	6149	6473,123	6263,25	23579	165,25
perm38	63	64	69	71,821	73	212	9	146	178	190	244,795	193	18109	15	405	418	418	430,038	470	2769	52	978	984	1005	1079,657	1135,25	5480	151,25	2538	2562	2591	2750,692	2887,25	5408	325,25	6179	6228	6310,5	6580,909	6628	16469	400
perm39	64	65	71	73,489	81	198	16	164	165	177	188,101	199	2243	34	422	427	495	485,208	518	719	91	1015	1038	1152	1162,594	1232,25	3819	194,25	2542	2563	2623,5	2949,704	3041,25	19076	478,25	6051	6082	6711	6850,163	6771	16429	89
perm40	64	72	74	152,703	85	40157	13	169	200	212	275,421	225	21780	25	389	457	469	569,469	482,25	20934	25,25	1078	1092	1110	1176,505	1246,5	2610	154,5	2285	2309	2355,5	2480,973	2659,25	5001	350,25	5830	5869	5908	6013,442	5978	91,26	109
perm41	68	75	87	114,978	92	11203	17	170	186,75	205	210,183	221	1440	34,25	399	417	466	472,754	487	2234	70	952	966	992	1049,515	1117	2750	151	2856	2886	2930	3127,904	3282,5	9334	396,5	6054	6134	6350	6593,25	7056	8603	922
perm42	71	91	93	145,963	95	15944	4	173	186	204	208,678	220	1004	34	404	425,75	490	493,396	518,25	2874	92,5	953	960	1008	1083,331	1145	7295	176	2857	2885	2925	3081,84	3181,25	12804	290,25	6136	6187,75	6360	6911,696	6858,75	27060	371
perm43	65	70	74	82,422	86	1650	16	166	194	199	211,898	214	1324	20	404	425,75	490	493,396	518,25	2874	92,5	953	960	1008	1083,331	1145	7295	176	2857	2885	2925	3081,84	3181,25	12804	290,25	6136	6187,75	6360	6911,696	6858,75	27060	371
perm44	65	71	74	82,748	86	3922	15	164	184	194	197,484	200	1036	16	470	483	495,5	530,725	575	843	92	1075	1098	1113,5	1185,067	1287	2856	180	2482	2509	2557	2702,046	2830,25	7731	330,25	6148	6192	6306,5	6643,12	6881,75	19415	689,75
perm45	51	64	72	75,305	84	223	20	164	178	197	199,316	215	655	37	465	478	556,5	546,708	588	1309	110	1024	1037	1042	1084,559	1078,25	2343	41,25	2698	2725	2800,5	3068,783	3315	13453	590	1511	1552	1557	1800,343	2101,25	13188	549,25
perm46	64	65	70,285	71,25	289	7,25	170	204	216	302,867	231	9999	27	305	456	480	540,388	493	1254,5	37	1013	1019	1029	1067,537	1063	2177	44	2764	2789	2821	2980,413	3077	4586	288	2142	2167	2179	2338,226	2256,25	9251	89,25

Tableau A.4 – Pourcentages d'outliers pour l'algorithme ShellSort sur CPU

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
ShellSort	128	6.78	5.8
	256	9.22	6
	512	5.51	3.5
	1024	3.9	3.5
	2048	4	4
	4096	9.3	9.3

La Figure A.5 et le tableau A.3 montrent les différents critères de performances (médiane et moyenne) de plusieurs permutations avec différentes taille du tableau pour l'algorithme ShellSort. Par la suite, nous calculons le pourcentage d'outliers pour cet algorithme pour renseigner sur les éventuelle erreur de mesure.

Le tableau A.4 montre que les pourcentages des tailles des permutations 128/256/512/1024 et 2048 sont entre 3% et 6%. D'autre part, la boite est à peu près symétrique par rapport à la médiane pour N=128/256/512. Pour cela, l'analyse graphique indique que l'algorithme ShellSort suit une loi normale pour N=128/256/512/1024/2048. Par contre, nous remarquons que les pourcentages d'outliers augmentent à 9% pour N=4096 et la boite à moustache n'est pas symétrique par rapport la médiane. De plus, la figure A.6 montre une variation de temps en fonction de la taille du tableau N. nous pouvons déduire que la fonction shellsort est une fonction de puissance.

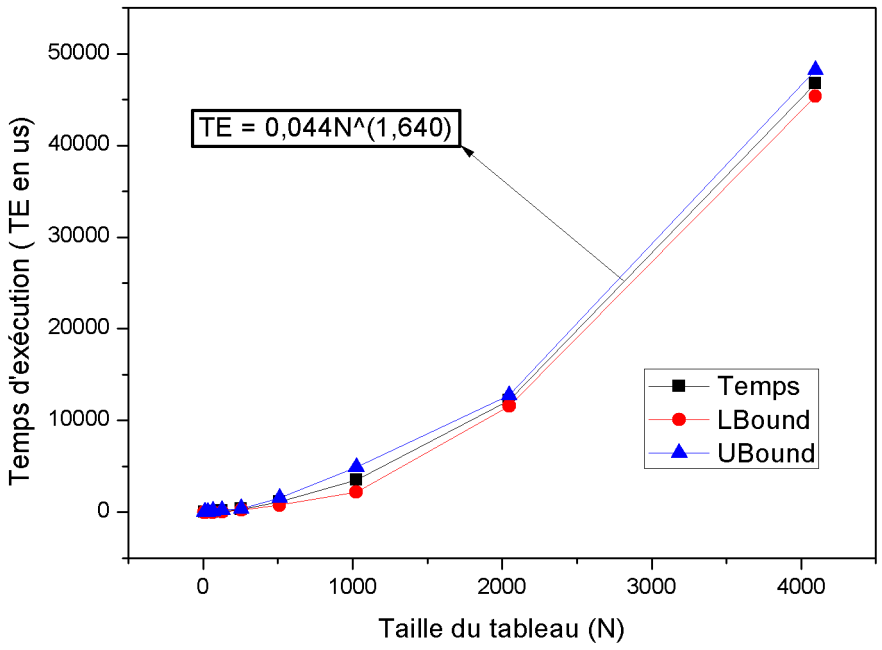


FIGURE A.6 – Variation temporelle de l'algorithme ShellSort sur CPU

HeapSort

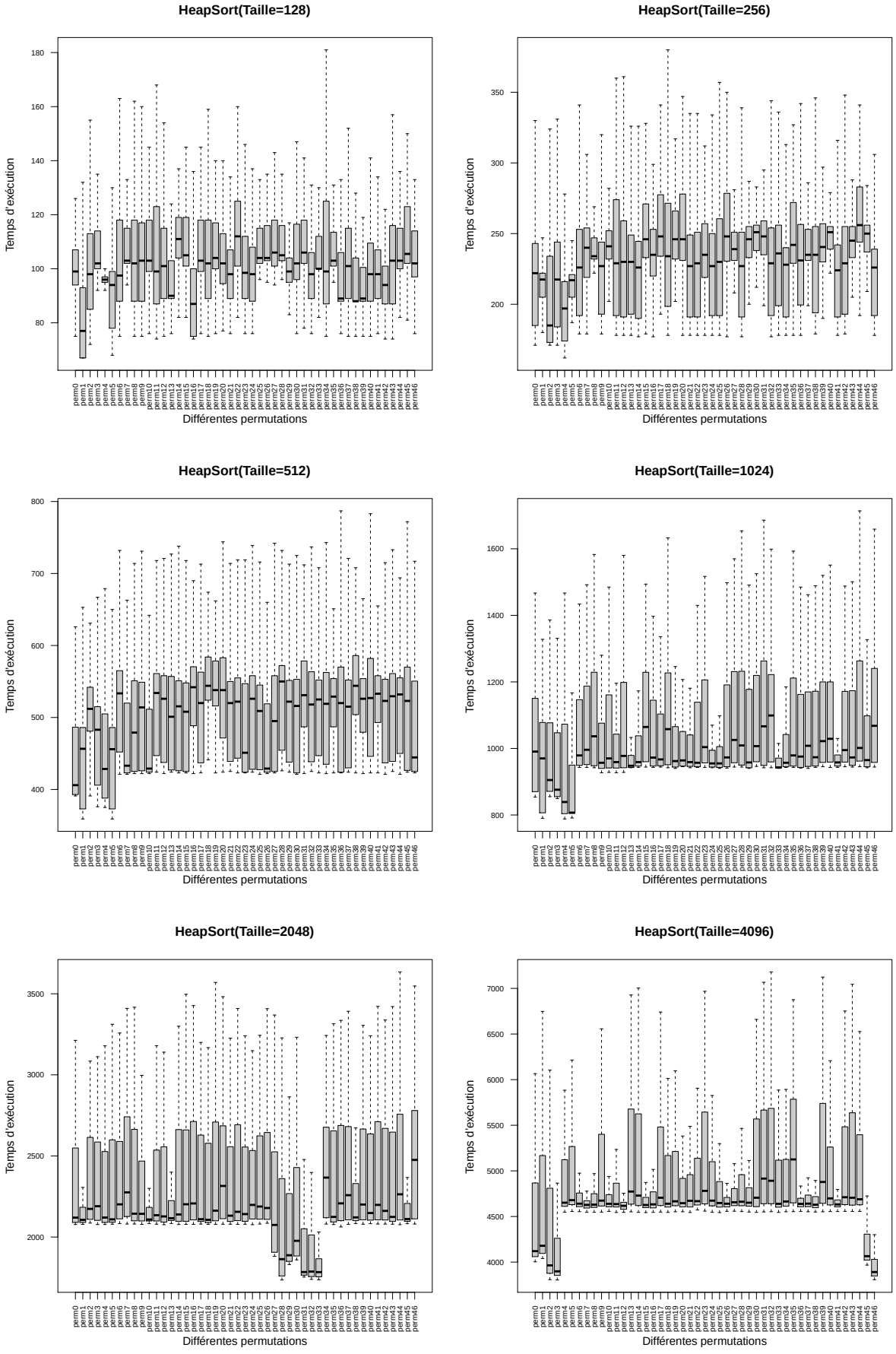


FIGURE A.7 – Boxplots de l'algorithme HeapSort sur CPU

Tableau A.5 – Critères de performances de l'algorithme HeapSort sur CPU

	128					256					512					1024					2048					4096				
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	74	94	99	119,74	107	3494	13	171	185	222	221,264	243	985	58	391	393	406	444,983	486,25	1197	93,25	854	870	990,5	1025,346	1150,25	4730	280,25	2077	2090	2119,5	2348,456	2549	10820	459	4003	4060	4120	4523,218	4866,75	19710	806,75
perm1	67	67	77	80,355	93	1190	26	161	205	217,5	446,695	222	19334	17	359	373	456,5	449,034	486	1935	113	790	806,75	970	990,5	1078	7135	271,25	2078	2090	2106	2235,608	2183,35	22320	93,25	4039	4095	4179	4612,82	5166,75	13713	1071,75
perm2	72	85	98	101,706	113	879	28	171	173	185	208,85	234	454	61	391	481	512	540,026	542	12618	61	855	871	905	981,839	1077	4320	206	2080	2109	2173,5	2458,62	2614,35	29098	505,25	3808	3877,75	3964,5	4315,501	4809,5	13592	931,75
perm3	76	100	102	233,024	114	3.4911	14	171	184	217,5	219,339	244	708	60	376	406	483	682,298	515	27246	109	849	855	876	954,342	1047	1434	192	2077	2103	2100	2352,133	2585	11268	482	3804	3854	3869	4221,743	4261,25	24069	407,25
perm4	73	95	96	144,564	97	1.61.67	2	162	174	197	218,485	216	14051	42	375	388	428,5	503,848	505	7461	117	788	804	839	1017,74	1073	20324	269	2077	2090	2120	2392,543	2525,5	3638	435,5	4547	4611,75	4651	5164,567	5120,5	35897	508,75
perm5	68	78	94	92,327	99	387	21	187	205	217	399,744	221	16287	16	359	373	456	462,228	486	3943	113	791	805	867	890,568	947	1335	142	2079	2091	2109	2362,152	2599	12043	508	4555	4630	4679	5103,188	5266,5	11824	636,5
perm6	75	88	97,5	105,091	118	2108	30	179	192	226	227,288	253	670	61	421	452,5	533,5	637,817	565	22252	112,5	943	949	979	1063,593	1146,25	3577	197,25	2083	2112	2201,5	2353,106	2589,25	3426	477,25	4553	4608	4640,5	5030,811	4757,25	46534	149,25
perm7	94	102	103	188,174	115	1.5732	13	179	219	240	246,055	254	4346	35	421	423	433	474,914	520	4560	97	943	951,75	996	1075,359	1187,25	2238	235,5	2080	2126,75	2275,5	2439,56	2741,25	3753	614,5	4553	4608	4640,5	4592,75	4627	4933,829	4673	401,89	80,25
perm8	75	88	102	116,925	118	3245	30	222	232	234	260,031	247	8297	15	422	425	479	509,591	551	10234	126	942	948	1036,5	1103,982	1229	4161	281	2080	2100	2144	2383,972	2664	13043	564	4546	4600	4630	5024,043	4749,5	16417	1419,5
perm9	75	88	103	103,775	117	2.21	29	179	193	227	228,005	244	484	51	422	426	514	505,019	549	964	123	927	940	957	1036,958	1076	3544	136	2078	2099	2143	2342,167	2468,25	10687	369,25	4548	4615	4674	5055,748	5401	16065	786
perm10	76	99	103	128,461	118	8588	19	201	232	241	318,802	252	22616	20	422	424	429	467,626	511,25	945	87,25	928	941,75	970	1051,027	1161	1526	219,25	2083	2099	2148	2329,378	2182,5	17798	83,5	4556	4606	4639	4834,215	4739,25	17174	133,25
perm11	74	87	99	103,249	123	293	36	178	191	229	236,393	274	479	82	424	447	534	524,678	561	1454	114	927	941	959	1031,088	1043	6459	102	2079	2099	2134,5	2313,626	2534,5	5547	435,5	4544	4604	4636	4932,754	4864,75	15860	260,75
perm12	75	89	101	105,394	115	3556	26	178	191	230	232,905	259	552	68	422	447,75	526	521,705	558	4419	120,25	928	942	977,5	1230,738	1198,25	17555	256,25	2076	2092	2127	2298,99	2555	3303	463	4545	4579	4617	4738,06	4656	8006	77
perm13	76	89	90	96,114	103	335	14	178	193	230	249,68	249	4540	56	424	427	501	544,371	557	24605	130	942	943	948	995,315	979	1513	36	2083	2102	2116	2278,037	2224,25	11639	122,25	4551	4635	4773,5	5135,214	5677,25	11359	1042,25
perm14	81	104	111	189,739	119	1.6181	15	177	190	226	223,81	244,25	479	54,25	423	426	515,5	509,468	551	1306	125	943	947	959	1032,059	1037,5	2228	990,5	2077	2097	2139	2407,535	2662,25	18767	565,25	4545	4620	4729	5168,442	5624,5	28803	1004,5
perm15	82	101	105	202,667	119	2.0366	18	179	233	246	261,496	271	4218	38	423	425	508	503,254	548	789	123	944	960	1064,5	1190,272	1229	1837,7	269	2078	2097	2232	2382,185	2660,25	10738	563,25	4543	4597	4626	4998,875	4709,25	32102	1112,25
perm16	74	75	87	95,334	100	1838	25	177	230	235	348,086	253	20307	33	422	488,75	542	549,606	570,25	5640	81,5	943	947	972,5	1050,706	1144	5360	197	2078	2105,75	2307	2397,162	2711,75	5053	606	4545	4594	4633	4816,07	4770	15211	176
perm17	76	99	103	114,289	118	563	19	193	234	248	362,36	277,25	14635	43,25	423	437	520	610,245	563	19469	126	944	948	967	1037,56	1103,25	2323	135,25	2080	2096	2110	2542,833	2828,25	31245	532,25	4549	4620	4705,5	5065,838	5481	14371	861
perm18	75	89	102	105,384	118	3.57	29	178	198,75	234	250,409	271,25	2082	72,5	422	524	544	626,097	584	20592	60	942	951	1058	1178,227	1226,5	17365	275,5	2078	2089	2106	2358,483	2574,5	22747	485,5	4545	4603,75	4640	4953,012	5167,5	14183	563,75
perm19	76	100	104	111,243	117	453	17	202	232	246	391,548	266	20353	34	422	516	538	542,637	578,25	814	62,25	940	944	962	1043,627	1065,25	8010	121,25	2078	2100	2162,5	2411,45	2709	9120	609	4552	4618	4666,5	4961,789	5211	10132	593
perm20	77	94,75	102	104,877	113	447	18,25	178	231	246	421,926	278	16311	47	424	471,75	538	534,545	583	1360	111,25	945	947	964	1021,564	1051	4383	104	2079	2113	2315	2441,954	2885,25	10084	572,25	4552	4605	4646	4952,265	4917,25	18159	312,25
perm21	76	89	98	100,051	107	502	18	178	191	227	233,254	249	3282	58	435	439	520	513,763	550	1109	111	942	946	959	1033,36	1040,25	2080	94,25	2079	2096	2131,5	2314,03	2556,5	8631	460,5	4544	4604	4672	4916,157	4958,25	12062	354,25
perm22	82	101	112	188,3	125	1.7723	24	178	191	229	232,54	251	4669	60	423	443,75	522	524,553	555	3739	111,25	943	945	975	1271,228	1138,5	16902	193,5	2078	2101	2155,5	2369,61	2691,5	5601	590,5	4571	4626	4668	4965,464	5136,5	8075	510,5
perm23	76	89	98,5	100,68	112	254	23	178	219	235	242,912	257	1468	38	423	424	451	498,281	547	1737	123	943	956,75	1004	1091,644	1206	6212	249,25	2079	2097	2141,5	2388,059	2555	20200	458	4550	4638	4781	5159,861	5644,25	16648	1006,25
perm24	76	88	98	101,152	108	3222	20	178	192	227	226,241	250	504	58	424	428	526	520,727	558	4036	130	942	944	955	997,564	994,25	1490	50,25	2076	2111	2138	2362,367	2552,75	11256	421,75	4546	4617	4673,5	5149,23	5098,5	29021	481,5
perm25	80	102	104	205,282	115	1.6714	13	178	192	230	240,247	260,25	1643	68,25	421	427,75	509	535,108	545	5334	117,25	940	943	955	1012,98	1005,25	3598	62,25	2080	2110	2188	2369,993	2624	19704	514	4543	4603	4647,5	4881,613	4881,75	67709	278,75
perm26	95	103	104	136,663	116	1.1651	13	177	230,75	248	286,417	278,25	8402	47,5	422	424	429	472,222	519	949	95	941	945	973	1067,933	1191	2253	246	2085	2109	2179	2464,493	2644,25	22699	535,25	4557	4605	4640	4830,363	4709	7465	104
perm27	94	101	106	112,462	118	639	17	177	231	239	243,863	251	583	20	423	425	495	522,063	558	6045	133	944	957	1025,5	1135,872	1231	1041,7	274	1881	1906	2074,5	2419,477	2525,25	22846	619,25	4558	4619	4656	4986,513	4805,25	30371	186,25
perm28	96	103	105	240,399	116	2.4201	13	177	191	227	230,514	251	601	60	436	454,75	550	532,861	572	1401	117,25	944	950	1009	1227,703	1232	2080,0	282	1735	1760	1863,5	2343,959	2800	29773	600	4554	4618	4662	4919,445	4957,75	6946	339,75
perm29	75	95	99	101,588	104	763	9	178	233	246	472,331	255	24557	22	424	438	522	515,574	551,25	1689	113,25	940	942	958	1138,036	1177,25	13383	235,25	1830	1850	1887,5	2419,016	2206	3879	416	4549	4613	4651	4950,489	4815,25	19084	202,25
perm30	76	96	102	106,111	116,25	347	20,25	209	238	251	265,382	256	9680	18	421	423	516	523,558	553	4185	130	946	960	1007	1087,074	1219,25	2012	259,25	1859	1883	1976,5	2156,997	2428,25	5891	545,25	4545	4608	4705,5	5057,317	5568	6660	960
perm31	76	102	106	148,683	118	1.2223	16	197	235	248	334,01	259	12421	24	422	487,5	531	530,67	578,25	813	90,75	942	949	1066,5	1329,062	1263	17604	314	1753	1765	1784	2034,259	2050	20689	285	4550	4640	4691,5	5190,959	5666,25	10044	1026,25
perm32	76	89	98	99,998	106	3.95	17	177	192	229	233,254	254	973	62	435	438,75	518	803,802	563,25	19016	124,5	942	959,75	1099	1114,928	1222	1649	282,25	1739	1756	1787	1915,036	2012	4078	246	4553	4640	4891,5	5169,312	5684,5	9114	1044,5
perm33	81	100	100	307,238	112	2.8265	12	178	199	236	398,726	256	22484	57	423	447,5	525	518,98	552	4196	104,5	940	941	943	982,475	971	1451	30	1736	1755	1783,5	1860,551	1866	2896	111	4545	4602	4639	5087,406	5117,25	26230	515,25
perm34	75	87	99	106,068	125	1041	38	178	191	228	227,311	240	1393	49	422	435	519	521,367	562,25	4907	127,25	943	945	957	1034,908	1042	1595	97	2080	2119,5	2366,5	2433,889	2677	5452	557,5	4552	4613	4664,5	5008,533	5124,25	28695	511,25
perm35	95	101	103	162,475	113,25	1.6168	12,25	178	220	242	257,829	272	3458	43	423	487	529	532,751	554	20883	67	943	947	979	1078,971	1211,25	1635	2664,25	2077	2093	2124	2318,669	2654,25	4059	561,25	4543	4640,75	5125,5	5262,113	5784,75	9676	1135
perm36	76	88	89	110,91	106	8183	18	178	190,75	231	240,689	256,25	2889	56,5	423	424	520	597,453	570	24614	146	941	943	975,5	1056,396	1162,25	1922	2219,25	2082	2101	2307,5	2412,322	2868	10941	587	4546	4607	4638	4844,601	4700	19392	93
perm37	75	89	101	104,854	115	931	26	179	231	235	246,971	253	850	22	424	430	515	505,307	552	721	122	940	946	1008	1071,021	1170	2080	224	2078	2106	2258	2395,061	2680,5	5226	574,5	4552	4608	4641	4837,093	4735	7053	127
perm38	75	88	88	95,556	104	418	16	179	194	235	246,593	255	5562	61	423	504	544	551,562	586	3701	82	943	947	973,5	1060,915	1172	3734	225	2083	2101	2121	2313,361	2330	10881	229	4549	4596	4629	4815,826	4717,5	14047	121,5
perm39	75	88	89	95,391	100,25	2.97	12,25	178	230	240,5	288,267	257	12306	27	422	479,75	526	524,998	554	3271	74,25	945	958	1022	1091,358	1200	3316	242	2079	2106	2230,5	2371,706	2665,5	7756	550,5	4552	4644	4676,5	5183,603	5730,25	7124	1095,25
perm40	75	88	98	98,908	100,25	225	21,25	206	239	251	442,781	255	28370	16	423	446	527	544,367	582	3896	136	943	959	1029	1087,247	1200	1582	241	2082	2104	2148	2393,546	2635,25	3241	531,25	4554	4627	4693,5	5010,696	5259	761,8	632
perm41	76	89	98	101,625	107	6.02	18	178	191	224	224,516	242	2015	51	423	493	533	537,074	558	6048	65	943	947	958	1160,363	980,25	25501	333,25	2080	2105	2161	2198	2376,873	2711	5946	605	4554	4605	4632	4820,97	4682	13895	77
perm42	74	87	94	96,261	101	1233	14	179	193	229	233,914	255	1457	62	421	437	523	515,882	553	4446	116	945	959	995	1081,088	1171,25	5390	212,25	2080	2105	2161	2355,572	2670	5390	565	4556	4630,75	4711	5255,869	5483	50336	852,25
perm43	74	87	103	103,057	116	1752	29	205	233	245	310,282	255	8358	22	435	439	529,5	707,398	561	24617	122	944	949	973	1065,67	1172,75	9920	223,75	2079	2096	2124	2323,37	2644	4118	548	4553	4624,75	4705,5	5095,986	5635,75	13529	1011
perm44	82	100	103	111,207	115	450	15	180	244	256	336,915	283	16303	39	421	450	532	545,988	555	11388	105	945	962	1001,5	1208,928	1263	18886	301	2076	2105	2283,5	2431,496	2757	6460	632	4556	4628,75	4689	5047,525	5396	10635	767,25
perm45	81	102	105,5	209,205	123	12236	21	207	237	250	283,557	256	15558	19	424	426,75	523	521,605	570	2732	143,25	942	945	965	1042,562	1007	10614	152	2082	2097	2111	2425,737	2205,5	3503,7	108,5	3967	4022,75	4065	4370,18	4305,75	256,70	283
perm46	76	97	102	141,295	114	1.6167	17	178	192	226	226,738	239	11182	47	423	425	444,5	499,869	550,25	3088	125,25	944	958	1068	1279,879	1240,25	15306	2824,25	2080	2112	2475,5	2579,057	2779,5	18784	667,5	3807	3848,75	3891	4117,652	4029,25	12088	180,5

Tableau A.6 – Pourcentage d'outliers pour HeapSort sur CPU

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
HeapSort	128	4.17	3,5
	256	4.4	3,8
	512	1.8	1,7
	1024	4.99	4,99
	2048	3.41	3,4
	4096	10.1	10

La figure A.7 et le tableau A.5 expriment les résultats obtenus pour la médiane et la moyenne des permutations avec différentes taille du tableau de l'algorithme HeapSort. Par la suite, nous observons un nombre important des valeurs aberrantes. Pour cela, nous calculons le pourcentage d'outliers pour cet algorithme.

Le tableau A.10 indique que les pourcentages des tailles des permutations 128/256/512/1024 et 2048 sont varié entre 1% et 5%. D'autre part, la boite est à peu près symétrique par rapport à la médiane pour N=128/256/512. Pour cela, l'analyse graphique indique que l'algorithme HeapSort suit une loi normale pour N=128/256/512/1024/2048. Par contre, nous remarquons que les pourcentages d'outliers augmentent à 10% pour N=4096 et la boite à moustache n'est pas symétrique par rapport la médiane. Donc, cet algorithme ne suit pas la loi normale pour N=4096.

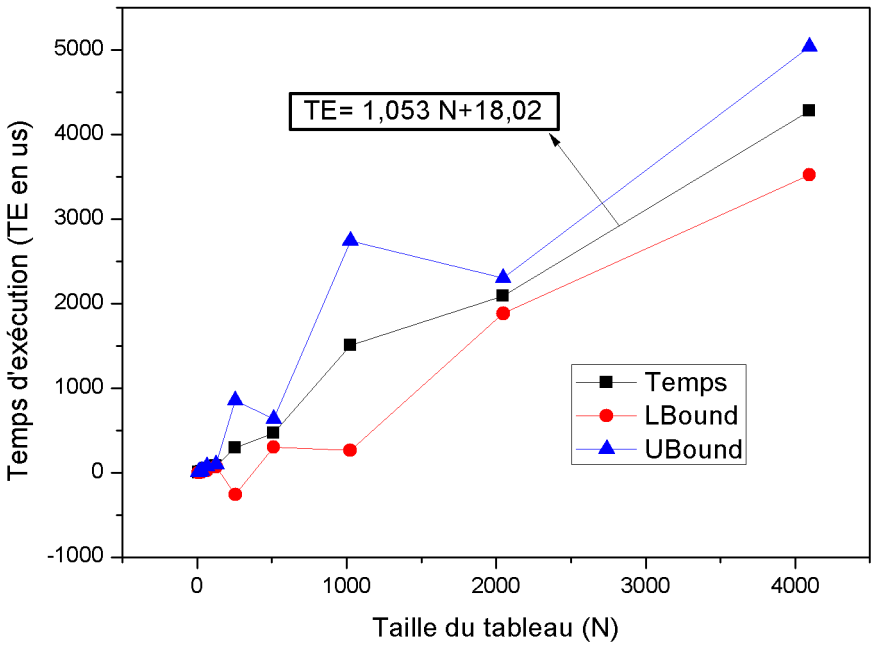


FIGURE A.8 – Variation temporelle de l'algorithme HeapSort sur CPU

La figure A.8 représente la variation du temps d'exécution pour l'algorithme HeapSort en fonction de la taille du tableau. Nous constatons que la fonction heapSort est une fonction linéaire c'est à dire si le on augmente le nombre N, le temps d'exécution augmente.

A.2.2 Exécution sur le processeur ARM de FPGA

Dans cette partie, nous expliquons les résultats de la complexité indiqué dans le chapitre 3 sous forme des Boxplots pour bien visualiser et interpréter les résultats sur le processeur ARM.

MergeSort

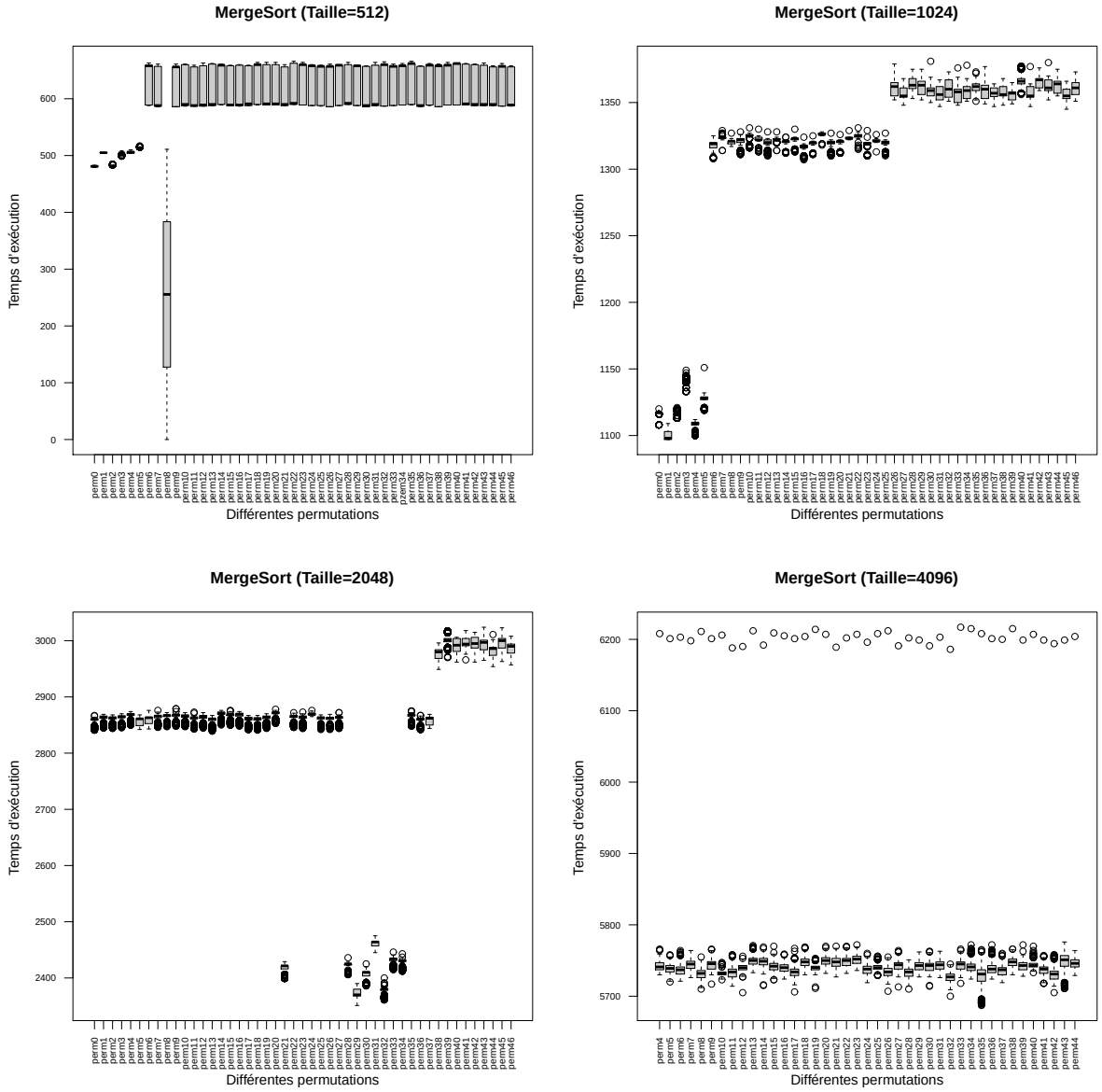


FIGURE A.9 – Boxplots de l'algorithme MergeSort sur le processeur ARM

Tableau A.7 – Critères de performances de l'algorithme MergeSort sur ARM

	512						1024						2048						4096					
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	479	480	481	481,015	482	483	2	1108	1117	1117	1116,594	1117	1120	0	2841	2859	2860	2857,97	2861	2867	2	5700	5650	5741	5720,52	5750	6100	100
perm1	505	505	505	505	505	505	0	1097	1097	1098	1100,252	1103	1109	6	2845	2862	2864	2861,873	2866	2869	4	5600	5500	5800	5823,2	5740	6105	240
perm2	482	482	482	482,236	482	484	0	1113	1116	1116	1116,294	1116	1121	0	2844	2861	2863	2860,106	2864	2868	3	5750	5800	5900	5985,2	5820	6800	20
perm3	499	501	501	500,866	501	503	0	1133	1141	1141	1140,945	1141	1149	0	2845	2863	2865	2862,018	2866	2870	3	5862	5750	5850	5500	5760	5780	30
perm4	504	504	506	505,305	507	510	3	1100	1108	1109	1108,875	1110	1112	2	2850	2867	2869	2866,133	2870	2874	3	5730	5737	5741	5742,096	5747	6208	10
perm5	514	514	514	514,218	514	517	0	1119	1127	1128	1127,942	1129	1151	2	2842	2849	2861	2857,58	2862	2868	13	5720	5734	5739	5738,649	5743	6201	9
perm6	588	589	658	626,765	659	663	70	1308	1316	1319	1318,226	1320	1325	4	2843	2852,75	2863	2859,459	2864	2876	11,25	5721	5731	5737	5737,267	5741	6203	10
perm7	586	587	588	621,278	657	661	70	1314	1323	1324	1323,942	1324	1329	1	2846	2863	2866	2862,448	2867	2876	4	5736	5739	5745	5744,866	5749	6198	10
perm8	0	127,75	255,5	255,5	383,25	511	255,5	1317	1319	1321	1320,191	1321	1327	2	2847	2865	2867	2864,609	2868	2872	3	5710	5726	5732	5731,523	5736	6211	10
perm9	586	586	656	624,348	656	661	70	1311	1320	1322	1321,778	1323	1328	3	2848	2866	2868	2865,638	2869	2879	3	5717	5738	5745,5	5745,192	5748,25	6201	10,25
perm10	587	588	590	622,608	660	661	72	1316	1325	1325	1325,336	1326	1331	1	2847	2864	2866	2863,407	2868	2872	4	5723	5730	5732	5732,541	5734	6206	4
perm11	587	587	588	620,202	656	659	69	1313	1322	1322	1322,65	1324	1330	2	2843	2861	2863	2860,587	2864	2873	3	5714	5727	5734	5733,362	5738	6188	11
perm12	587	587	589	618,456	658	663	71	1310	1319	1320	1320,233	1322	1328	3	2845	2862	2865	2863,313	2867	2872	5	5705	5737	5740	5739,958	5743	6190	6
perm13	588	588	590	623,971	661	662	73	1314	1322	1322	1322,33	1323	1328	1	2839	2858	2861	2858,434	2862	2867	4	5733	5745	5750	5749,754	5753	6212	8
perm14	589	590	659	625,053	659	661	69	1312	1320	1321	1320,68669	1321	1324	1	2850	2869	2871	2869	2873	2876	4	5715	5744	5749	5749,17	5753	6192	9
perm15	588	588	589	619,982	658	659	70	1313	1322	1323	1322,275	1323	1330	1	2850	2867	2868	2866,629	2870	2876	3	5723	5737	5742	5741,701	5746	6209	9
perm16	587	588	589	621,8	659	660	71	1307	1316	1317	1316,965	1318	1324	2	2848	2866	2869	2866,834	2871	2874	5	5724	5735	5740	5740,286	5744	6205	9
perm17	588	588	591	622,386	658	659	70	1311	1319	1320	1319,518	1320	1325	1	2842	2859	2861	2859,282	2864	2867	5	5706	5729	5734	5734,273	5738	6201	9
perm18	589	590	660	628,352	660	664	70	1318	1326	1326	1326,203	1327	1328	1	2841	2859	2863	2858,891	2863	2867	4	5730	5743	5748	5747,851	5752	6204	9
perm19	589	589	591	623,993	660	664	71	1310	1319	1320	1319,637	1321	1327	2	2844	2861,75	2864	2861,546	2866	2870	4,25	5711	5737	5740	5740,398	5742	6214	5
perm20	589	589	591	624,45	660	664	71	1312	1320	1321	1320,727	1321	1326	1	2853	2870	2872	2869,199	2873	2878	3	5732	5745	5750	5750,018	5754	6207	9
perm21	588	588	590	621,786	656	660	68	1322	1323	1323	1323,214	1324	1329	1	2399	2416	2421,5	2418,092	2422	2429	6	5727	5742	5748	5747,958	5753	6189	11
perm22	590	590	592	624,426	662,25	666	72,25	1315	1324	1325	1325,024	1326	1331	2	2846	2864	2865	2862,598	2867	2872	3	5732	5743	5750	5749,112	5753	6202	10
perm23	589	589	660	626,351	660	664	71	1310	1319	1319	1319,049	1320	1329	1	2844	2862	2864	2861,8	2866	2873	4	5736	5746	5752	5751,864	5756	6207	10
perm24	588	588	658	624,471	658	660	70	1313	1321	1321	1321,355	1322	1326	1	2865	2868	2869	2869,575	2871	2876	3	5719	5732	5738	5737,496	5742	6196	10
perm25	587	588	657	623,457	659	659	69	1310	1319	1320	1319,978	1321	1327	2	2842	2861	2863	2860,482	2864	2868	3	5729	5738	5740	5741	5743	6208	5
perm26	586	586	657	622,996	657	661	71	1352	1355	1362	1362,285	1365	1379	10	2842	2860	2862	2861,774	2864	2869	4	5707	5729	5734	5734,353	5739	6212	10
perm27	588	588	659	624,484	659	660	71	1348	1354	1355	1355,464	1361	1368	7	2844	2862	2865	2861,903	2865	2872	3	5713	5738	5744	5743,213	5747	6191	9
perm28	590	590	592	624,799	660	664	70	1353	1360,75	1363	1363,086	1367,5	1375	6,75	2406	2423	2425	2422,108	2425	2436	2	5710	5728	5734	5733,893	5738	6202	10
perm29	588	588	658	624,913	658	659	70	1352	1356	1363	1362,999	1366	1375	10	2351	2368	2370	2372,892	2380	2390	12	5727	5737	5743	5742,746	5747	6199	10
perm30	586	586	588	619,06	657	658	71	1350	1355	1359	1358,913	1361	1381	6	2386	2403,75	2410	2406,439	2411	2425	7,25	5714	5736	5743	5741,732	5746	6191	10
perm31	588	588	590	622,676	659	664	71	1347	1352	1356	1356,683	1362	1367	10	2445	2457	2464	2461,235	2465	2475	8	5726	5738	5744	5743,598	5748	6203	10
perm32	587	587	660	625,975	660	665	73	1351	1354	1360	1360,159	1367	1373	13	2361	2378	2380	2377,066	2381	2400	3	5700	5722	5727	5727,154	5731	6186	9
perm33	588	588	657	624,025	658	661	70	1348	1350	1358	1357,784	1360	1376	10	2415	2431	2434	2430,853	2435	2446	4	5718	5738	5745	5743,863	5748	6217	10
perm34	589	589	658	625,27	658	662	69	1351	1353	1359	1359,359	1362	1378	9	2412	2429	2431	2428,138	2432	2443	3	5724	5736	5741	5741,146	5745	6215	9
perm35	589	590	662	626,235	662	666	72	1351	1359	1362	1361,334	1364	1373	5	2848	2866	2868	2865,003	2869	2875	3	5687	5721	5731	5728,646	5737	6208	16
perm36	586	587	588	621,643	657	658	70	1349	1353	1360	1359,757	1363	1377	10	2842	2859	2861	2858,807	2862	2867	3	5724	5733	5738	5739,117	5743	6201	10
perm37	588	589	659	625,207	660	662	71	1347	1354,75	1357	1356,499	1361	1364	6,25	2844	2850	2862	2858,722	2864	2869	14	5719	5730	5737	5735,857	5740	6200	10
perm38	586	586	658	623,425	658	661	72	1348	1355	1356	1356,111	1362	1368	7	2949	2968,75	2980	2979,438	2984	2996	15,25	5730	5743	5748	5747,991	5752	6215	9
perm39	589	589	659	625,478	660	664	71	1349	1352	1357	1356,829	1358	1365	6	2970	2998	3001	3001,162	3004	3017	6	5728	5737	5743	5742,986	5747	6199	10
perm40	589	589	662	627,601	662	663	73	1356	1364	1366	1365,9	1368	1377	4	2962	2981	2992	2991,836	3004	3006	23	5733	5741	5743	5744,255	5746	6207	5
perm41	589	589	591	624,104	661	662	72	1347	1354	1355	1355,292	1362	1377	8	2966	2991	2994	2994,192	3004	3018	13	5718	5732	5738	5737,218	5741	6199	9
perm42	588	588	590	622,372	660	661	72	1359	1361	1367	1366,542	1368	1376	7	2962	2987	2995	2995,484	3006	3015	19	5705	5724	5731	5730,852	5735	6194	11
perm43	588	588	590	621,185	659	663	71	1352	1359	1361	1360,869	1367	1380	8	2965	2984	2997	2996,57	3001	3024	17	5711	5742	5751	5748,346	5757	6199	15
perm44	588	588	590	622,098	656	658	68	1355	1357	1364	1363,878	1366	1375	9	2954	2974	2986	2985,159	2988	3011	14	5729	5741	5746	5746,408	5751	6204	10
perm45	587	587	657	623,257	658	662	71	1345	1353	1355	1354,807	1360	1366	7	2963	2987	3000	2999,812	3004	3023	17	942	945	965	1042,562	1097	10614	152
perm46	587	588	589	621,826	656	658	68	1351	1356	1361	1360,859	1365	1373	9	2957	2978	2990	2989,001	2994	3008	16	944	958	1068	1279,879	1240,25	15266	282,25

Tableau A.8 – Pourcentage d'outliers pour MergeSort sur ARM

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
HeapSort	512	1	0.7
	1024	3.2	1.1
	2048	13.3	0.6
	4096	0.7	0.5

La figure A.9 et le tableau A.7 montrent les résultats obtenus pour la médiane et la moyenne des permutations avec différentes taille du tableau de l'algorithme MergeSort. Par la suite, nous observons un nombre important des valeurs aberrantes. Pour cela, nous calculons le pourcentage d'outliers pour cet algorithme. Alors, le tableau A.8 donne un aperçu sur le pourcentage total d'outliers et le pourcentage d'outliers par valeurs supérieures pour chaque taille de l'algorithme MergeSort. Nous remarquons que les données aberrantes sont négligeables pour l'algorithme MergeSort.

HeapSort

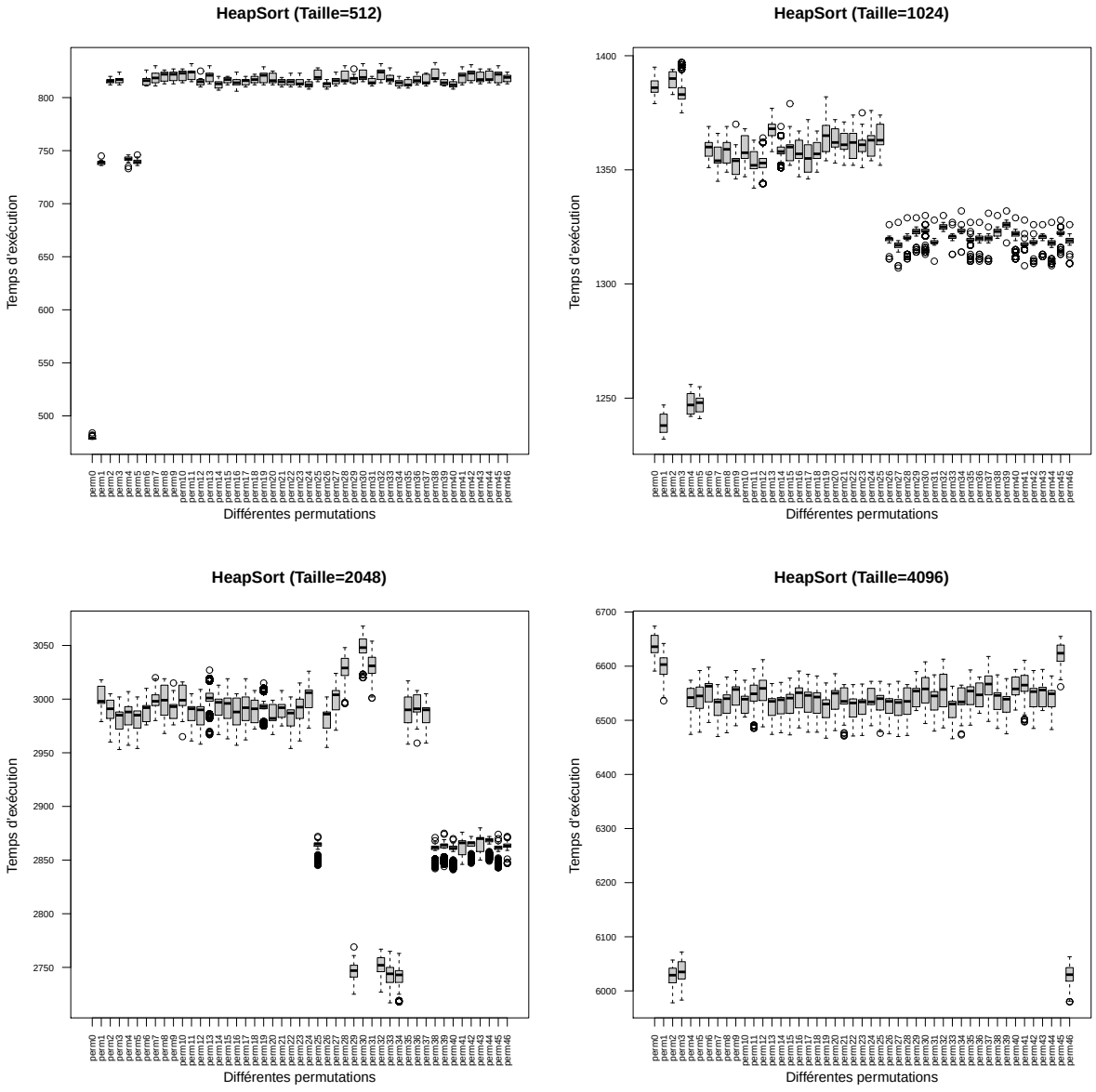


FIGURE A.10 – Boxplots de l'algorithme HeapSort sur le processeur ARM

Tableau A.9 – Critères de performances de l'algorithme HeapSort sur ARM

	512						1024						2048						4096					
	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR	Q0	Q1	Q2	Moyenne	Q3	Q4	IQR
perm0	478	478	479	478,771	479	484	1	1379	1384	1386	1386,124	1389	1395	5	2960	2982	2991	2990,27427	2999	3005	17	6591	6625	6636	6637,245	6657	6674	32
perm1	736	737	739	738,577	740	745	3	1232	1235	1238	1238,875	1243	1247	8	2979	2996	2998	2998,61161	3012	3018	16	6536	6585	6603	6599,605	6615,25	6642	30,25
perm2	812	814	815,5	815,628	817	820	3	1383	1386	1390	1388,601	1393	1394	7	2960	2982	2991	2990,27427	2999	3005	17	5978	6015	6029	6031,775	6042	6057	27
perm3	812	814	817	815,791	818	824	4	1375	1381	1383	1383,803	1386	1397	5	2953	2972	2985	2983,73073	2988	3002	16	5983	6022	6035	6035,337	6054	6072	32
perm4	733	741	742	742,414	744	746	3	1242	1243	1247	1247,358	1252	1256	9	2957	2976	2988	2987,43243	2993	3007	17	6474	6525	6542	6541,436	6559	6574	34
perm5	736	738	739	739,481	741	746	3	1241	1244	1248	1248,221	1250	1255	6	2954	2973	2985	2983,70671	2989	3002	16	6478	6522	6545	6542,874	6561,25	6592	39,25
perm6	811	812	816	815,416	818	826	6	1351	1356	1360	1359,854	1362	1369	6	2976	2979	2992	2991,71672	2994	3010	15	6496	6534	6563	6558,79	6568	6598	34
perm7	811	814	818,5	818,081	823	830	9	1345	1353	1354	1353,932	1360	1366	7	2980	2994	2998	2997,85786	3004	3020	10	6470	6509	6534	6533,232	6539	6566	30
perm8	813	815,75	822	819,731	824	826	8,25	1349	1353	1359	1358,665	1362	1369	9	2968	2985	2999	2998,77778	3013	3019	28	6477	6513	6540	6534,115	6547	6580	34
perm9	813	816	822	820,096	824	827	8	1346	1348	1354	1353,366	1355	1370	7	2976	2982	2993	2992,52553	2995	3015	13	6490	6528	6557	6554,064	6562	6592	34
perm10	814	816	823	820,91	825	827	9	1347	1355	1357,5	1357,617	1365	1368	10	2965	2994	2999	2998,77578	3013	3016	19	6506	6513	6539	6538,923	6544,25	6574	31,25
perm11	815	817	824	821,403	825	832	8	1342	1350,75	1352	1352,312	1358	1363	7,25	2961	2980	2991	2989,97998	2993	3005	13	6485	6535,75	6549	6550,285	6565	6595	29,25
perm12	810	812	815	814,678	817	825	5	1344	1351	1353	1353,036	1355	1364	4	2958	2976	2990	2987,4995	2993	3009	17	6488	6536,75	6559	6556,723	6574	6612	37,25
perm13	813	815	821	819,219	823	830	8	1358	1365	1368	1367,916	1370	1377	5	2967	2998	3001	3000,93493	3005,5	3027	7,5	6474	6509	6535	6529,888	6540	6568	31
perm14	807	809	813	811,926	815	820	6	1351	1357	1358	1358,048	1359,5	1369	2,5	2967	2985	2997	2995,94995	3000	3013	15	6477	6512	6538	6531,788	6542	6570	30
perm15	812	814	817	816,727	817	820	5	1352	1354	1360	1360,104	1361	1379	7	2963	2982	2996	2994,79279	3001	3019	19	6473	6513	6541	6539,864	6548	6578	35
perm16	806	812	814	814,598	817	824	5	1347	1355	1357	1357,398	1363	1367	8	2957	2976	2988	2988,5976	3001	3005	25	6486	6523	6551	6545,518	6560	6591	37
perm17	810	812	816	815,087	817	820	5	1346	1349	1355	1354,729	1361	1372	12	2962	2980	2992	2992,94094	3002	3019	22	6478	6515	6546	6539,179	6552	6585	37
perm18	812	814	817	816,853	820	822	6	1349	1355	1357	1356,816	1362	1367	7	2972	2978	2991	2991,04004	2999	3008	21	6478	6513	6543	6536,88	6551	6582	38
perm19	812	814	821	818,845	823	829	9	1354	1358	1365	1365,153	1369,25	1382	11,25	2975	2991	2993	2992,68569	2995	3015	4	6467	6505	6530	6523,998	6538	6569	33
perm20	812	814	816	818,472	823	825	9	1353	1360	1362	1362,784	1368	1372	8	2967	2980	2982	2982,81081	2995	2999	15	6481	6520,75	6550	6546,033	6555	6586	34,25
perm21	810	812	815	814,841	817	819	5	1352	1359	1361	1361,504	1365	1371	6	2975	2983	2992	2991,62763	2995	3008	12	6471	6530	6535	6535,816	6560	6566	30
perm22	810	812	815	814,391	817	823	5	1352	1355	1362	1361,54	1366	1374	11	2954	2975	2987	2985,10711	2990	3002	15	6471	6506	6532	6527,875	6539,25	6566	33,25
perm23	810	812	813	814,236	817	823	5	1351	1358	1361	1360,706	1363	1375	5	2961	2982	2992	2991,9009	3000	3015	18	6472	6511	6534	6531,081	6538	6567	27
perm24	808	810	812	812,772	815	817	5	1354	1356	1363	1362,345	1365	1376	9	2973	2992	3006	3004,96597	3008,5	3026	16,5	6490	6529	6534	6534,831	6559	6572	30
perm25	815	817	819	821,301	826	828	9	1352	1361	1363	1363,709	1370	1374	9	2845	2863	2865	2861,87788	2866	2872	3	6476	6519	6540	6537,599	6546	6572	27
perm26	808	810	813	812,495	814	817	4	1311	1319	1320	1319,533	1320	1326	1	2955	2973	2986	2983,33534	2988	3002	15	6475	6513	6535	6531,138	6540	6567	27
perm27	811	813	816	815,681	818	824	5	1307	1316	1317	1316,718	1318	1327	2	2971	2990	3004	3003,18118	3008	3024	18	6470	6509	6533	6532,849	6538,25	6572	29,25
perm28	813	815	816	819,2	825	830	10	1311	1320	1320	1320,217	1321	1329	1	2996	3022	3029	3029,22122	3038	3048	16	6472	6511	6535	6535,029	6560	6566	49
perm29	813	814	818	817,129	819	827	5	1314	1322	1323	1322,941	1324	1329	2	2725	2741	2747	2746,79479	2752	2769	11	6518	6525	6554	6553,609	6559	6590	34
perm30	815	817	819	821,689	826	832	9	1313	1323	1323	1323,129	1324	1330	1	3020	3043	3048	3049,15415	3056	3068	13	6494	6532	6558	6558,593	6579	6608	47
perm31	811	813	814	815,55	818	820	5	1310	1318	1318	1318,447	1319	1328	1	3001	3024	3031	3031,1021	3039	3054	15	6480	6519	6545	6539,488	6553	6584	34
perm32	814	817	824	821,585	826	832	9	1323	1324	1325	1324,696	1326	1330	2	2727	2746	2752	2752,13113	2759	2767	13	6486	6525	6557	6556,181	6585	6613	60
perm33	813	815	817	817,936	821	828	6	1313	1320	1321	1320,642	1321	1327	1	2717	2736	2744	2743,64665	2750	2765	14	6466	6505	6530	6527,852	6535	6563	30
perm34	809	811	814	813,814	816	820	5	1314	1323	1323	1323,441	1324	1332	1	2718	2736	2743	2742,64364	2747	2763	11	6473	6529	6534	6535,076	6560	6565	31
perm35	809	811	812	813,704	817	819	6	1310	1319	1319	1319,264	1320	1327	1	2958	2978	2990	2989,23223	3002	3017	24	6491	6529	6554	6549,617	6562	6593	33
perm36	812	814	816	816,838	820	824	6	1310	1319	1320	1319,914	1321	1327	2	2959	2988	2991	2991,11612	3003,5	3008	15,5	6513	6525	6547	6547,118	6569	6580	44
perm37	811	813	814	817,25	822	823	9	1310	1319	1320	1319,862	1321	1331	2	2959	2978	2990	2988,73073	2992	3005	14	6498	6547	6567	6566,243	6582	6618	35
perm38	815	817	818	821,749	827	833	10	1320	1321	1323	1322,447	1324	1330	3	2842	2860	2862	2859,15315	2863	2871	3	6485	6520	6546	6539,007	6551	6584	31
perm39	811	812	814	814,78	817	823	5	1318	1325	1326	1325,737	1327	1332	2	2844	2861,5	2864	2860,93393	2865	2875	3,5	6475	6515	6539	6534,939	6544	6577	29
perm40	808	810	812	812,091	815	817	5	1311	1321	1322	1321,656	1323	1329	2	2841	2860	2861	2859,18519	2863	2870	3	6519	6547	6558	6557,347	6580	6594	33
perm41	812	814	821	818,937	823	829	9	1308	1317	1317	1317,102	1318	1328	1	2846	2855	2866	2862,86186	2868	2876	13	6497	6553,75	6565	6566,479	6583	6611	20,25
perm42	814	816	823	820,689	825	831	9	1309	1318	1318	1318,289	1319	1326	1	2847	2863	2866	2863,01401	2867	2872	4	6485	6525	6553	6551,598	6559	6592	34
perm43	813	815	817	819,655	824	827	9	1312	1320	1321	1320,761	1321	1326	1	2850	2858	2870	2866,17017	2871	2880	13	6518	6525	6556	6554,756	6561	6594	36
perm44	814	816	817	820,239	825	827	9	1308	1317	1318	1317,76	1319	1327	2	2849	2867	2869	2866,02302	2870	2872	3	6483	6525	6549	6547,734	6555	6582	30
perm45	812	814	822	819,134	824	830	10	1313	1322	1322	1322,551	1323	1328	1	2843	2860	2862	2859,62062	2863	2874	3	6562	6609	6624	6621,136	6639	6655	30
perm46	813	815	819	818,361	821	824	6	1309	1318	1319	1319,187	1320	1326	2	2847	2862	2863	2863,35536	2865	2872	3	5980	6018	6030	6028,625	6043	6063	25

Tableau A.10 – Pourcentage d'outliers pour HeapSort sur ARM

Tailles/Nbre d'outliers		% d'outliers total	% d'outliers supérieur
HeapSort	512	0	0
	1024	1.5	0.4
	2048	5.3	0.9
	4096	0	0

La figure A.10 et le tableau A.9 présentent les résultats obtenus pour la médiane et la moyenne des permutations avec différentes taille du tableau de l'algorithme HeapSort. Nous calculons le pourcentage d'outliers liée au bruit de système d'exploitation.

Le tableau A.10 indique que les pourcentages des tailles des permutations sont trop petit. D'autre part, la boite est à peu près symétrique par rapport à la médiane.

Publications

International Journal

Computacion y Sistemas 2018 Yomna BEN JMAA, Rabie BEN ATITALLAH, David DUVIVIER, Maher BEN JEMAA. A comparative Study of Sorting Algorithms with FPGA Acceleration by High Level Synthesis. *International Journal of Computacion y Sistemas*, 2018.

International Conference

AICCSA 2017 Yomna BEN JMAA, Rabie BEN AATITALLAH, Maher BEN JEMAA.An Optimized Software Radio Application Using a Dynamic Slack Reclamation Technique on a Real Platform OMAP 3530. In *Proceedings of the* 13^{th} *International Conference on Computer Systems and Applications*, pages 637-644, 2017, Hamammet, Tunisie, IEEE.

HPCS 2017 Yomna BEN JMAA, Karim M.A.Ali, David DUVIVIER, Maher BEN JEMAA, Rabie BEN ATITALLAH. An Efficient Hardware Implementation of TimSort and MergeSort Algorithms Using High Level Synthesis. In *Proceedings of the* 15^{th} *International Conference on High Performance Computing & Simulation*, pages 580-587, 2017, Genoa, Italy, IEEE.

Bibliographie

[1] D. Solet, J. Béchennec, and M. Briday, "Hw-based architecture for runtime verification of embedded software on sopc systems," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2018, Edinburgh, United Kingdom, August 6-9, 2018*, 2018, pp. 249–256.

[2] B. Riad, "Méthodologies de conception pour multiprocesseurs sur circuits logiques programmables," Ph.D. dissertation, ENSTA ParisTech, 2007.

[3] M. A. Redjedal, "Optimisation multicritere pour le placement d'applications intensives sur systeme-sur-puce (soc)," 2010.

[4] L. Huan and Y. Wenhua, "Efficient implementation of fpga based on vivado high level synthesis," in *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*, 2016, pp. 2810–2813.

[5] P. Gilles, "Méthodologies de conception conjointe logicielle/matérielle," Ph.D. dissertation, Montpellier 2, 2002.

[6] K. Ali, "Processus de conception conjointe logiciel matériel dirigés par les modèles," Ph.D. dissertation, Lille 1, 2010.

[7] F. Lino, J. Isabel, M. J. Tenreiro, F. J. Rui, and D. C. J. Martins, "Towards the development of intelligent transportation systems," in *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, 2001, pp. 1206–1211.

[8] D. George and D. Panagiotis, "Intelligent transportation systems," *IEEE Vehicular Technology Magazine*, vol. 5, no. 1, pp. 77–84, 2010.

[9] A. Johann and S. Steve, "Intelligent transport systems (its)-an overview," *IFAC Proceedings Volumes*, vol. 33, no. 18, pp. 99–106, 2000.

[10] B. Flavien, A. Emmanuel, and M. René, "Positionnement des systèmes multi-agents pour les systèmes de transport intelligents," *Revue des Sciences et Technologies de l'Information-Série RIA : Revue d'Intelligence Artificielle*, vol. 30, no. 3, pp. 299–327, 2016.

[11] N. Satish, M. J. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," in *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, no. 10, 2009, pp. 1–10.

[12] C. Grozea, Z. Bankovic, and P. Laskov, "FPGA vs. multi-core cpus vs. gpus : Hands-on experience with a sorting application," in *Facing the Multicore-Challenge - Aspects of New Paradigms and Technologies in Parallel Computing [Proceedings of a conference held at the Heidelberger Akademie der Wissenschaften, March 17-19, 2010]*, 2010, pp. 105–117.

[13] K. Nikolajevic, "Dynamic autonomous decision-support function for piloting a helicopter in emergency situations. (système décisionnel dynamique et autonome pour le pilotage d'un hélicoptère dans une situation d'urgence)," Ph.D. dissertation, University of Valenciennes and Hainaut-Cambresis, France, 2016.

[14] H. CLAUSSEN, "Vehicle navigation systems," in *Modern Cartography Series*, 1991, vol. 1, pp. 225–235.

[15] M. S, M. T, Y. N, and T. T, "Comprehensive automobile traffic control system," *J. of IECE of Japan*, vol. 62, pp. 870–887, 1979.

[16] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.

[17] N. Barnier, "Application de la programmation par contraintes à des problèmes de gestion du trafic aérien. (application of constraint programming to air traffic management problems)," Ph.D. dissertation, National Polytechnic Institute of Toulouse, France, 2002.

[18] L. Xavier, *La sécurité et la sûreté des transports aériens*, 2005.

[19] P. Yann, "Aviation civile et terrorisme : naissance et enjeux d'une politique européenne de sûreté des transports aériens," *Cultures & Conflits*, no. 56, pp. 83–119, 2004.

[20] G. Afonso, "Vers une nouvelle génération de systèmes de test et de simulation avionique dynamiquement reconfigurables," Ph.D. dissertation, Lille University of Science and Technology, France, 2013.

[21] B. Erik, K. Paul, P. Pavel, and K. Kathleen, "Summary of avionics technologies," *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 9, pp. 6–11, 2015.

[22] B. Gilles, C. Laurent, R. ERBA, and O. Mathieu, "Synchronization of a dual avionic and non-avionic system," 2018.

[23] G. A. Kemayo, "Evaluation et validation des systÃ¨mes distribuÃ©s avioniques," Ph.D. dissertation, LIAS/UniversitÃ© de Poitiers, France, 2014.

[24] S. Tobias, S. Sérgio, T. Cássia, and A. José, "An integrated modular avionics development environment," in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, 2009, pp. 1–A.

[25] L. Sifan, H. Lin, L. Yang, and S. Dong, "Integrated modular avionics anomaly detection based on symbolic time series analysis," in *Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2017 IEEE 2nd*, 2017, pp. 2095–2099.

[26] K. Tammo, G. Christian, R. Konrad, and L. Pavel, "An architecture for inline anomaly detection," in *Computer Network Defense, 2008. EC2ND 2008. European Conference on*, 2008, pp. 11–18.

[27] S. Omar, B. Rabie, D. David, A. AbedlHakim, B. Nicolas, and F. Pierre, "Path planning : A 2013 survey," in *Industrial Engineering and Systems Management (IESM), Proceedings of 2013 International Conference on*, 2013, pp. 1–8.

[28] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[29] M. Brady, "Artificial intelligence and robotics," *Artif. Intell.*, vol. 26, no. 1, pp. 79–121, 1985.

[30] O. Souissi, "Planification de chemin d'hélicoptères sur une architecture hétérogène cpu fpga haute performance," Ph.D. dissertation, University of Valenciennes and Hainaut-Cambresis, France, 2015.

[31] B. Said, B. Assia, T. Mohamed, S. Florentin, and V. Luige, "Applying dijkstra algorithm for solving neutrosophic shortest path problem," in *Advanced Mechatronic Systems (ICAMechS), 2016 International Conference on*, 2016, pp. 412–416.

[32] H. I. Kang, B. Lee, and K. Kim, "Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm," in *PACIIA 2008, Volume 2, 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 19-20 December 2008, Wuhan, China*, 2008, pp. 1002–1004.

[33] J. Yao, C. Lin, X. Xie, A. J. A. Wang, and C. Hung, "Path planning for virtual human motion using improved a* star algorithm," in *Seventh International Conference on Information Technology : New Generations, ITNG 2010, Las Vegas, Nevada, USA, 12-14 April 2010*, 2010, pp. 1154–1158.

[34] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[35] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, USA, March 25-28, 1985*, vol. 2, 1985, pp. 500–505.

[36] L. S. M, "Rapidly-exploring random trees : A new tool for path planning," 1998.

[37] C. Shi-Gang, W. Hui, and Y. Li, "A simulation study of a-star algorithm for robot path planning," in *16th international conference on mechatronics technology, PP*, 2012, pp. 506–510.

[38] S. Shudong and L. Mao, "The coordination path planning of multiple moving robots based on ga," *Automation Journal*, vol. 26, no. 5, pp. 672–676, 2000.

[39] R. Bernard and B. Denis, *Aide multicritère à la décision : méthodes et cas.* Economica Paris, 1993.

[40] S. T. L, *Décider face à la complexité : une approche analytique multicritère d'aide à la décision.* Esf Editeur, 1984.

[41] F. Sitorus, J. J. Cilliers, and P. R. Brito-Parada, "Multi-criteria decision making for the choice problem in mining and mineral processing : Applications and trends," *Expert Syst. Appl.*, vol. 121, pp. 393–417, 2019.

[42] P. SD and R. M, "Application of multi-criteria decision making to sustainable energy planning a review," *Renewable and sustainable energy reviews*, vol. 8, no. 4, pp. 365–381, 2004.

[43] C. João, *Multicriteria analysis.* Springer Verlag, 1997.

[44] T. Efraim, "Decision support and expert systems," *Management*, 1995.

[45] K. Reuven, F. Paul, and B. Avishai, "Multicriterion issues in energy policymaking," *European Journal of Operational Research*, vol. 56, no. 1, pp. 30–40, 1992.

[46] M. S and D. D, "Multicriteria analysis vs. externalities assessment for the comparative evaluation of electricity generation systems," *European Journal of Operational Research*, vol. 102, no. 2, pp. 364–379, 1997.

[47] S. Pekka, H. Joonas, and L. Risto, "Comparing multicriteria methods in the context of environmental problems," *European Journal of Operational Research*, vol. 104, no. 3, pp. 485–496, 1998.

[48] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y. Chen, A. Baransi, S. Kumar, and P. Dubey, "Efficient implementation of sorting on multi-core SIMD CPU architecture," *PVLDB*, vol. 1, no. 2, pp. 1313–1324, 2008.

[49] P. Séchet, "Efficacité du tri dans le contexte de mémoire virtuelle," Ph.D. dissertation, Université de Nancy 1 : UER de Sciences Mathématiques, 1984.

[50] A. Aggarwal and J. S. Vitter, "The input/output complexity of sorting and related problems," *Commun. ACM*, vol. 31, no. 9, pp. 1116–1127, 1988.

[51] K. A. kumar, "A survey, discussion and comparison of sorting algorithms," Master's thesis, Umea University, Department of Computing Science, 2014.

[52] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of fpga, GPU and CPU in image processing," in *19th International Conference on Field Programmable Logic and Applications, FPL 2009, August 31 - September 2, 2009, Prague, Czech Republic*, 2009, pp. 126–131.

[53] Y. Yu, V. Kushal, J. Yiyue, and O. Erdal, "Comparison of gpu and fpga based hardware platforms for ultrasonic flaw detection using support vector machines," in *Ultrasonics Symposium (IUS), 2017 IEEE International*, 2017, pp. 1–4.

[54] Z. Dominik, P. Marcin, W. Maciej, and W. Kazimierz, "The comparison of parallel sorting algorithms implemented on different hardware platforms," *Computer Science*, vol. 14, no. 4), pp. 679–691, 2013.

[55] M. Danelutto, T. D. Matteis, G. Mencagli, and M. Torquati, "A divide-and-conquer parallel pattern implementation for multicores," in *Proceedings of the 3rd International Workshop on Software Engineering for Parallel Systems, SEPS@SPLASH 2016, Amsterdam, Netherlands, November 1, 2016*, no. 10, 2016, pp. 10–19.

[56] G. Konstantinos, C. Grigorios, M. Pavlos, N. Antonis, T. Nikos, D. Apostolos, P. Dionisios, and P. Yannis, "An evaluation of vivado hls for efficient system design," in *ELMAR, 2016 International Symposium.* IEEE, 2016, pp. 195–199.

[57] Y. B. Jmaa, K. M. A. Ali, D. Duvivier, M. B. Jemaa, and R. B. Atitallah, "An efficient hardware implementation of timsort and mergesort algorithms using high level synthesis," in *2017 International Conference on High Performance Computing & Simulation, HPCS 2017, Genoa, Italy, July 17-21, 2017*, 2017, pp. 580–587.

[58] A. R, "Vhdl implementation of merge sort algorithm," in *International Journal of Computer Science and Communication Engineering*, 2014, pp. 1–6.

[59] D. Koch and J. Tørresen, "Fpgasort : a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting," in *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27, March 1, 2011*, 2011, pp. 45–54.

[60] D. Pasetto and A. Akhriev, "A comparative study of parallel sort algorithms," in *Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011, Portland, OR, USA, October 22 - 27, 2011*, no. 2, 2011, pp. 203–204.

[61] J. Bilal, M. Bartolomeo, R. Carlo, K. F. Gul, and K. Omar, "Fast parallel sorting algorithms on gpus," *International Journal of Distributed and Parallel Systems*, p. 107, 2012.

[62] S. Dong, X. Wang, and X. Wang, "A novel high-speed parallel scheme for data sorting algorithm based on fpga," in *Image and Signal Processing, 2009. CISP'09. 2nd International Congress on.* IEEE, 2009, pp. 1–4.

[63] F. A. Alquaied, A. I. Almudaifer, and M. A. AlShaya, "A novel high-speed parallel sorting algorithm based on fpga," in *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International.* IEEE, 2011, pp. 1–4.

[64] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A hybrid design for high performance large-scale sorting on FPGA," in *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, Riviera Maya, Mexico, December 7-9, 2015*, 2015, pp. 1–6.

[65] J. Matai, D. Richmond, D. Lee, Z. Blair, Q. Wu, A. Abazari, and R. Kastner, "Resolve : Generation of high-performance sorting architectures from high-level synthesis," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 21-23, 2016*, 2016, pp. 195–204.

[66] R. Chen, S. Siriyal, and V. K. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 22-24, 2015*, 2015, pp. 240–249.

[67] P. D. M. Jati, A. Ahmad, P. D. Teguh, W. Ari, and F. Andreas, "Implementation of serial and parallel bubble sort on fpga," *Jurnal Ilmu Komputer dan Informasi*, vol. 9, no. 2, pp. 113–120, 2016.

[68] J. F. Mart'inez Trinidad, R. Cumplido-Parra, and C. F. Uribe, "An fpga-based parallel sorting architecture for the burrows wheeler transform," in *2005 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2005, Puebla City, Mexico, September 28-30, 2005*, 2005.

[69] B. Stephen, A. Wendell, L. Marco, and R. Robert, "Sorting using the xilinx virtex-4 field programmable gate arrays on the cray xd1," *Proc. Craig User Group*, pp. 1–12, 2008.

[70] Y. Sogabe and T. Maruyama, "FPGA acceleration of short read mapping based on sort and parallel comparison," in *24th International Conference on Field Programmable Logic and Applications, FPL 2014, Munich, Germany, 2-4 September, 2014*, 2014, pp. 1–4.

[71] S. T. Singh, K. Surmeet, and K. Snehdeep, "Enhanced insertion sort algorithm," *International journal of Computer applications*, vol. 64, no. 21, 2013.

[72] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A hybrid design for high performance large-scale sorting on FPGA," in *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, Riviera Maya, Mexico, December 7-9, 2015*, 2015, pp. 1–6.

[73] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.

[74] P. Xuan, W. B. Ligon, P. K. Srimani, R. Ge, and F. Luo, "Accelerating big data analytics on HPC clusters using two-level storage," *Parallel Computing*, vol. 61, pp. 18–34, 2017.

[75] P. Dwi, A. M. Roby, W. Ari, and T. M. Iqbal, "Investigation of flip-flop performance on different type and architecture in shift register with parallel load applications," *Jurnal Ilmu Komputer dan Informasi*, vol. 8, no. 2, pp. 83–91, 2015.

[76] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. A. Vissers, and Z. Zhang, "High-level synthesis for fpgas : From prototyping to deployment," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[77] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009.

[78] J. Harkins, T. A. El-Ghazawi, E. El-Araby, and M. Huang, "Performance of sorting algorithms on the SRC 6 reconfigurable computer," in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology, FPT 2005, 11-14 December 2005, Singapore*, 2005, pp. 295–296.

[79] B. Shahzad and M. Afzal, "Enhanced shellsorting algorithm," *computer journal of Enformatika*, vol. 21, no. 6, pp. 66–70, 2007.

[80] A. K. Sikder, "A comparative analysis of sorting algorithms with focus on merge sort," Ph.D. dissertation, United International University, 2018.

[81] A.-J. E. S, A.-A. H. A, and B. G. H, "Comparative performance evaluation of heap-sort and quick-sort algorithms," *International Journal of Computing*, vol. 3, no. 2, pp. 39–57, 2014.

[82] E. Manca, A. Manconi, A. Orro, G. Armano, and L. Milanesi, "Cuda-quicksort : an improved gpu-based implementation of quicksort," *Concurrency and Computation : Practice and Experience*, vol. 28, no. 1, pp. 21–43, 2016.

[83] A. Ribon, "Amelioration du processus de vÃ©rification des achitectures gÃ©nÃ©rÃ©es Ã l'aide d'outils de synthese de haut-niveau," Ph.D. dissertation, University of Bordeau, 2016.

[84] V. Sharma, P. S. Sandhu, S. Singh, and B. Saini, "Analysis of modified heap sort algorithm on different environment," *World Academy of Science, Engineering and Technology*, vol. 42, 2008.

[85] M. Mehdi, "Parallel hybrid optimization methods for permutation based problems," Ph.D. dissertation, Lille University of Science and Technology, France, 2011.

[86] D. L. Shell, "A high-speed sorting procedure," *Commun. ACM*, vol. 2, no. 7, pp. 30–32, 1959.

[87] M. Abid, K. Jerbi, M. Raulet, O. Déforges, and M. Abid, "Efficient system-level hardware synthesis of dataflow programs using shared memory based FIFO - HEVC decoder case study," *Signal Processing Systems*, vol. 90, no. 1, pp. 127–144, 2018.

[88] K. Akash, D. Akshay, and S. Gautam, "Merge sort algorithm," *International Journal of Research*, vol. 1, no. 11, pp. 16–21, 2014.

[89] N. Auger, C. Nicaud, and C. Pivoteau, "Merge Strategies : from Merge Sort to TimSort," Dec. 2015, working paper or preprint. [Online]. Available : https://hal-upec-upem.archives-ouvertes.fr/hal-01212839

[90] Y. Zhang, Y. Zhao, and D. Sanán, "A verified timsort C implementation in isabelle/hol," *CoRR*, vol. abs/1812.03318, 2018.

[91] D. Knuth, "The art of computer programming, volume 3 : Sorting and searching. addison," 1998.

[92] J. Barbay and G. Navarro, "On compressing permutations and adaptive sorting," *Theor. Comput. Sci.*, vol. 513, pp. 109–123, 2013.

[93] R. C. Team, "R : A language and environment for statistical computing," *dim (ca533)*, vol. 1, no. 1358, p. 34, 2018.

[94] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection : A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[95] *ASME. The Performance Test Standard PTC 19.1-2005 Test Uncertainty*, 2005. [Online]. Available : Technicalreport,AmericanSocietyofMechanicalEngineers, NewYork,UnitedStates

[96] *Xilinx. Inc.Zynq-7000 All Programmable SoC Software Developers Guide, Version 9.0*, 2014. [Online]. Available : http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf

[97] *ARM. Inc. AMBA AXI and ACE protocol specification*, 2013. [Online]. Available : http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html

[98] M. Baklouti, Y. Aydi, P. Marquet, J. Dekeyser, and M. Abid, "Scalable mpnoc for massively parallel systems - design and implementation on FPGA," *Journal of Systems Architecture - Embedded Systems Design*, vol. 56, no. 7, pp. 278–292, 2010.

[99] *Processeur Leon.* [Online]. Available : http://www.gaisler.com/index.html

[100] E. Matthews, L. Shannon, and A. Fedorova, "Shared memory multicore microblaze system with SMP linux support," *TRETS*, vol. 9, no. 4, pp. 26 :1–26 :22, 2016.

[101] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 131 :1–131 :35, 2019.

[102] S. Guan-Ming, C. Qian, Y. Peng, and Q. Sheng, "Hardware efficient sparse fir filtering in layered video coding," 2019.

[103] S. Al-Azawi, O. Nibouche, S. Boussakta, and G. Lightbody, "New fast and area-efficient pipeline 3-d DCT architectures," *Digital Signal Processing*, vol. 84, pp. 15–25, 2019.

[104] G. Mario, Q. Fahad, T. Jarmo, and G. Oscar, "Hardware architectures for the fast fourier transform," in *Handbook of Signal Processing Systems*, 2019, pp. 613–647.

[105] *Cours FPGA : L'architecture SOPC Des FPGAs*, 2014. [Online]. Available : http://philpetitpa.890m.com/SOPC.pdf

[106] R. S. Patti, "Three-dimensional integrated circuits and the future of system-on-chip designs," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1214–1224, 2006.

[107] *Xilinx. Inc. ZC-702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC*, 2013. [Online]. Available : http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf

[108] *ARM. Inc. AMBA AXI and ACE protocol specification*, 2013. [Online]. Available : http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html

[109] *Xilinx UG761 (v13.4) axi reference guide*, 2012. [Online]. Available : https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[110] F. Tom, "Vivado design suite user guide high-level synthesis," *White Paper*, vol. 5, p. 30, 2012.

[111] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. D. Brown, F. Ferrandi, J. H. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016.

[112] X. Liu, Y. Chen, T. Nguyen, S. T. Gurumani, K. Rupnow, and D. Chen, "High level synthesis of complex applications : An H.264 video decoder," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 21-23, 2016*, 2016, pp. 224–233.

[113] E. Kalali and I. Hamzaoglu, "FPGA implementations of HEVC inverse DCT using high-level synthesis," in *2015 Conference on Design and Architectures for Signal and Image Processing, DASIP 2015, Krakow, Poland, September 23-25, 2015*, 2015, pp. 1–6.

[114] P. Sjovall, V. Viitamaki, J. Vanne, and T. D. Hämäläinen, "High-level synthesis implementation of HEVC 2-d DCT/DST on FPGA," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, 2017, pp. 1547–1551.

[115] M. Kammoun, A. Ben Atitallah, R. Ben Atitallah, and N. Masmoudi, "Design exploration of efficient implementation on soc heterogeneous platform : Hevc intra prediction application," *International Journal of Circuit Theory and Applications*, vol. 45, no. 12, pp. 2243–2259, 2017.

[116] M. Le Guen, "La boîte à moustaches de tukey, un outil pour initier à la statistique," *Statistiquement votre-SFDS*, no. 4, pp. 1–3, 2001.

République Tunisienne
Ministère de l'Enseignement Supérieur, de
la Recherche Scientifique
et de la Technologie

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Ecole Doctorale
Sciences et Technologies

Thèse de *DOCTORAT*
Nom du Doctorat

N° d'ordre: 2018– ??nn nn

Implémentation temps réel des algorithmes de tri dans les applications de transports intelligents en se basant sur l'outil de synthèse haut niveau HLS

Yomna BEN JMAA CHTOUROU

الخلاصة: تلعب أنظمة النقل الذكية دورًا مهمًا في تقليل الحوادث وازدحام حركة المرور وتلوث الهواء. من بين هذه الأنظمة، نذكر مجال إلكترونيات الطيران الذي يستخدم في عدة حالات خوارزميات الفرز التي تعد واحدة من العمليات الهامة للتطبيقات المضمنة في الوقت الفعلي. ومع ذلك، فإن التطور التكنولوجي يتحرك نحو معماريات أكثر تعقيدًا لتلبية متطلبات التطبيق. وفي هذا الصدد، يجد المصممون الحل المثالي في الحوسبة القابلة لإعادة التشكيل، التي تستند إلى معماريات غير متجانسة من وحدة المعالجة المركزية FPGA التي تحتوي على معالجات متعددة المراكز (CPUs) و FPGAs تقدم أداءً عاليًا وقدرة على التكيف مع القيود في الوقت الفعلي. من التطبيق. الهدف الرئيسي من عملي هو تطوير تطبيقات الأجهزة لتطبيقات النقل الذكية (خوارزمية تخطيط خطة الطيران A *) وفرز الخوارزميات على بنية غير متجانسة من وحدة المعالجة المركزية FPGA باستخدام أداة التوليف العالية المستوى لتوليد تصميم RTL من الوصف السلوكي. تتطلب هذه الخطوة بذل جهود إضافية من جانب المصمم للحصول على تنفيذ فعال للأجهزة باستخدام العديد من التحسينات مع حالات الاستخدام المختلفة: البرامج والأجهزة المحسنة وغير المحسنة وكذلك للعديد من التبديلات / المتجهات التي تم إنشاؤها باستخدام مولد مبني على أساس Lehmer. لتحسين الأداء، قمنا بحساب وقت التشغيل والانحراف المعياري ورقم المورد المستخدم لفرز الخوارزميات من خلال النظر في أحجام عدة للبيانات تتراوح بين 8 إلى 4096 عنصرًا. وفي النهاية، قمنا بمقارنة أداء هذه الخوارزميات. ستعمل هذه الخوارزمية على دمج تطبيقات دعم القرار، والتخطيط لخطة الطيران.

Résumé : Les systèmes de transports intelligents jouent un rôle important dans la minimisation des accidents, embouteillage, et la pollution d'air. Parmi ces systèmes, nous citons le domaine avionique qui utilise dans plusieurs cas les algorithmes de tri qui sont l'une des opérations importante pour des applications embarquées temps réels. Cependant, l'évolution technologique tend vers des architectures de plus en plus complexes pour répondre aux exigences applicatives. À cet égard, les concepteurs trouvent leur solution idéale dans l'informatique reconfigurable, basée sur des architectures hétérogènes CPU/FPGA qui abritent des processeurs multi-core (CPU) et des FPGAs qui offrent de hautes performances et une adaptabilité aux contraintes temps-réel de l'application. L'objectif principal de mes travaux est de développer une implémentions matérielle des application de transports intelligents (algorithme de planification de plan de vol A*)et les algorithmes de tri sur l'architecture hétérogène CPU/FPGA en utilisant l'outil de synthèse haut niveau pour générer le design RTL à partir de la description comportementale. Cette étape nécessite des efforts supplémentaires de la part du concepteur afin d'obtenir une implémentation matérielle efficace en utilisant plusieurs optimisations avec différents cas d'utilisation : logiciel, matérielle optimisé et non optimisé et aussi pour plusieurs permutations/vecteurs générer à l'aide d'un générateur de permutation basé sur Lehmer. Pour améliorer les performances, nous avons calculés le temps d'exécution, l'écart type et le nombre de ressource utilisé pour les algorithmes de tri en considérant plusieurs tailles de données varient entre 8 et 4096 éléments. Finalement, nous avons comparé les performances de ces algorithmes. Cet algorithme sera intégrer les applications d'aide à la décision, planification du plan de vol.

Abstract : Intelligent transport systems play an important role in minimizing accidents, traffic congestion, and air pollution. Among these systems, we mention the avionics domain, which uses in several cases the sorting algorithms, which are one of the important operations for real-time embedded applications. However, technological evolution is moving towards more and more complex architectures to meet the application requirements. In this respect, designers find their ideal solution in reconfigurable computing, based on heterogeneous CPU / FPGA architectures that house multi-core processors (CPUs) and FPGAs that offer high performance and adaptability to real-time constraints. Of the application. The main objective of my work is to develop hardware implementations of sorting algorithms on the heterogeneous CPU / FPGA architecture by using the high-level synthesis tool to generate the RTL design from the behavioral description. This step requires additional efforts on the part of the designer in order to obtain an efficient hardware implementation by using several optimizations with different use cases: software, optimized and non-optimized hardware and for several permutations / vectors generated using the generator pf permutation based on Lehmer method. To improve performance, we calculated the runtime, standard deviation and resource number used for sorting algorithms by considering several data sizes ranging from 8 to 4096 items. Finally, we compared the performance of these algorithms. This algorithm will integrate the applications of decision support, planning the flight plan.

المفاتيح: أنظمة النقل الذكية، وحدة المعالجة المركزية FPGA **، منصة** zynq **، كفاءة الطاقة ، أداة** HLS **، خوارزميات الفرز ،** A *

Mots clés: Systèmes de transports intelligent, CPU/FPGA, plateforme zynq, Efficacité énergétique, l'outil HLS, algorithmes de tri, A*

Key-words: ITS, CPU/FPGA, Zynq platform, Energy efficiency, HLS tool, Sorting algorithms, A*