Équilibrage de charge dynamique sur plates-formes hiérarchiques

Quintin Jean-Noël

sous la direction de Denis Trystram et Frédéric Wagner
MOAIS, INRIA/LIG, Université de Grenoble

8 Decembre 2011
Clock Frequency

Intel Processor Clock Speed (MHz)

- Pentium 4 Prescott
- Core 2 Extreme
- Pentium III
- Celeron
- Pentium
- 80486
- 80386
- 80286
- 8080
Processor Evolution

- Pentium M (Mono-core 2003)
- Core 2 duo (Dual-core 2005)
- Core i7 (Quad-core 2007)
Next Generation: Network On Chip

Knight Corner
50 cores
2012?
Parallel Programming

- Parallelization of an application:
Parallel Programming

- Parallelization of an application:
  - Identify independent parts.
Parallel Programming

- Parallelization of an application:
  - Identify independent parts.

- A
- B
- C
- D
- E
- F
Parallel Programming

- Parallelization of an application:
  - Identify independent parts.
  - Describe tasks dependencies.
Parallel Programming

- Parallelization of an application:
  - Identify independent parts.
  - Describe tasks dependencies.
  - Schedule tasks on processors.
Scheduling

- Tasks scheduling:
  - Assigned a processor to each task.
  - Determined the starting time of tasks.
Outline

1. Introduction
2. Scheduling, Communication and Efficiency
3. Enhancement for Makefile Applications
4. Enhancement for the Hierarchical Platform
5. Conclusion
Online Scheduling

- List-scheduling [Graham-69].

![Diagram of a task graph and processors]

Ready tasks list

- A
Online Scheduling

- List-scheduling [Graham-69].

![Diagram showing process scheduling and timing]

- `B` and `C` are ready tasks.
- `A`, `B`, `C`, `D`, `E`, and `F` are tasks.
- `P1` and `P2` are processors.
- Time axis shows scheduling progression.

Ready tasks list:

```
B C
```
Online Scheduling

- List-scheduling [Graham-69].

![Diagram of processors and tasks]

- Processors: A, B, C, D, E, F
- Ready tasks list: D, E
- Time
- Processors

- $P_1$: A, B
- $P_2$: C
Online Scheduling

- List-scheduling [Graham-69].
- Centralized

```
+---+---+---+---+---+
| P1 |   |   |   | P2 |
+---+---+---+---+---+
| A  | B  | D  | E  | F  |
+---+---+---+---+---+
```

Processors

Time

Ready tasks list
Work-Stealing [Blumofe-95]
A distributed list scheduling
Work-Stealing [Blumofe-95]
A distributed list scheduling
Work-Stealing [Blumofe-95]

A distributed list scheduling
Work-Stealing [Blumofe-95]
A distributed list scheduling
Choose the stolen processor.
Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.
- Choose the stolen task.
Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.
- Choose the stolen task.
Work-Stealing
Performance analysis

- Assumptions:
  - Constant communication time and no data transfers.
  - DAG arity: 2, and unitary task.
  - Homogeneous processor (Bender & Rabin for heterogeneous).

- Bounds [Arora-01]:

\[ T_p \leq W_p + O(D) \]

Data transfers: bounded by the number of edges.
Steal requests: \( \#S \leq O(p \times D) \).
Work-Stealing
Performance analysis

Assumptions:
- Constant communication time and no data transfers.
- DAG arity: 2, and unitary task.
- Homogeneous processor (Bender & Rabin for heterogeneous).

Bounds [Arora-01]:

Bounds:
- Execution time: $T_p \leq W_p + O(D)$.
- Data transfers: bounded by the number of edges.
- Steal requests: $\#S \leq O(p \times D)$.
Assumptions:
- Constant communication time and no data transfers.
- DAG arity: 2, and unitary task.
- Homogeneous processor (Bender & Rabin for heterogeneous).

Bounds [Arora-01]:

\[ T_p \leq W_p + O(D) \]

Data transfers: bounded by the number of edges.

Steal requests: \[ \#S \leq O(p \times D) \].
Work-Stealing

Performance analysis

Assumptions:
- Constant communication time and no data transfers.
- DAG arity: 2, and unitary task.
- Homogeneous processor (Bender & Rabin for heterogeneous).

Bounds [Arora-01]:
- Execution time: \( T_p \leq \frac{W}{p} + O(D) \).
- Data transfers: bounded by the number of edges.
- Steal requests: \( \#S \leq O(p \times D) \).
Work-Stealing and Communications

- Data intensive application.
- Distributed platform (cluster).

≈ 55 seconds
≈ 35 seconds
≈ 58 seconds
Existing Algorithm
Taking into account platform topology

In Satin [Nieuwpoort-01]:
- **CLS**: Only one processor by cluster can send steal requests to processors inside other clusters.
- **CHS**: Each processor is a node of a binary tree. Steal requests are sent through the tree.
- **CRS**: Each processor can send two steal requests at the same time: one asynchronous and one synchronous.

In Kaapi [Gautier-07]:
- **KWS**: Each processor sends a request to the processor on the same computer before stealing a processor on another computer.
Outline

1. Introduction

2. Scheduling, Communication and Efficiency

3. Enhancement for Makefile Applications
   - WSCOM (Work-Stealing with COMmunication)
   - Performance Analysis

4. Enhancement for the Hierarchical Platform

5. Conclusion
Context

- **DSMake:**
  - Makefiles executions on distributed platforms.
  - Structure unrestricted.
  - Static DAG: structure is known in advance.
**Context**

- **DSMake:**
  - Makefiles executions on distributed platforms.
  - Structure unrestricted.
  - Static DAG: structure is known in advance.

- **Our aim:**
  - Take into account the DAG structure to minimize the number of transfers.
Simple Example

The scheduling depends on tasks management.

Add tasks which generate a task block.

Symmetry of the DAG.
Simple Example

The scheduling depends on tasks management. Add tasks which generate a tasks block. Symmetry of the DAG.
Simple Example
Simple Example

The scheduling depends on tasks management.

Add tasks which generate a tasks block.

Symmetry of the DAG.
Simple Example
Simple Example

The scheduling depends on tasks management.

Add tasks which generate a tasks block.

Symmetry of the DAG.
Simple Example

- The scheduling depends on tasks management.
Simple Example

- The scheduling depends on tasks management.
- Add tasks which generate a tasks block.
Simple Example

- The scheduling depends on tasks management.
- Add tasks which generate a tasks block.
  - Symmetry of the DAG.
Symmetry of the DAG
Symmetry of the DAG
Symmetry of the DAG
Symmetry of the DAG
**WSCOM**

Data communications

1. Add some virtual task before the execution.
2. Execute the new DAG with a work-stealing algorithm.
   - Manage data transfers.
     - Send data the earliest (\(\text{WSCOM}_{pf}\)).
     - Send data the latest (\(\text{WSCOM}\)).
Outline

1. Introduction
2. Scheduling, Communication and Efficiency
3. Enhancement for Makefile Applications
   - WSCOM (Work-Stealing with COMmunication)
   - Performance Analysis
4. Enhancement for the Hierarchical Platform
5. Conclusion
Practical Analysis

- Experiments vs simulations
- Simulations:
  - Many experiments.
  - Varying the platform characteristics (bandwidth).
  - Control of tasks execution time and communications time.
- Scheduling algorithms:
  - On-line heuristics:
    - Classical Work-Stealing.
  - Off-line heuristics:
    - List\_min \min(HEFT, CPOP, BIL, HBMCT, Sufferage, MinMin, MaxMin).
    - Known tasks execution time and data transfers time.
    - Not contention aware.
Inputs

- Platforms:
  - Clique without network contentions.
  - Cluster with network contentions.
Inputs

- **Platforms:**
  - Clique without network contentions.
  - Cluster with network contentions.

- **Application DAG:**
Inputs

- Platforms:
  - Clique without network contentions.
  - Cluster with network contentions.

- Application DAG:
  - Random DAG (TGFF [Dick-98], LBL [Tobita-02]).
  - DAG extracted from Makefile execution.
No Contention on Links
Clique platform, random DAG

![Graph showing execution time vs. number of computers for different algorithms.]

- **WSCOM$_{pf}$**
- **WSCOM**
- **list_min**
- **WS**
Contention on Links
Cluster Platform, Random DAG

![Graph](image)

- **Execution time (s)** vs **Number of computers**
- Lines represent different algorithms:
  - WSCOM
  - WSCOM\_pf
  - list\_min
  - WS

Quintin (MOAIS/Grenoble U.)
Dynamic Load-Balancing on Hierarchical Platforms
December 8, 2011
Data Transfers

Number of computers

Amount of transferred data (Gbits)

- WSCOM
- WSCOM
- list_min
- WS

Quintin (MOAIS/Grenoble U.)
DAG from Makefile Executions

WSCOM vs WS

DAG:
- 500 different DAG.
- Compilation of open-source softwares (MacPort [Rothman-08]).
DAG from Makefile Executions
WSCOM vs WS

- DAG:
  - 500 different DAG.
  - Compilation of open-source softwares (MacPort [Rothman-08]).
- Two kind of experiments:
  - Experiments as previous on random DAG.
    - Slight different results
  - Highlight the ability to exploit different platforms:
DAG from Makefile Executions

WSCOM vs WS

- **DAG:**
  - 500 different DAG.
  - Compilation of open-source softwares (MacPort [Rothman-08]).

- **Two kind of experiments:**
  - Experiments as previous on random DAG.
    - Slighty different results
  - Highlight the ability to exploit different platforms:
    - Can WSCOM achieve a significant speed-up with a low bandwidth
DAG from Makefile Execution

WSCOM vs WS

\[ \text{WSCOM (Mb/s)} \]
\[ \text{WS (Mb/s)} \]
DAG from Makefile Execution

WSCOM vs WS

![Graph showing the relationship between WSCOM and WS](image-url)
DAG from Makefile Execution

WSCOM vs WS

![Graph showing comparison between WSCOM and WS]

- **WSCOM (Mb/s)** vs **WS (Mb/s)**
- **Speed-up not achieved**
- **$10^0$** to **$10^3$** on the y-axis
- **$10^0$** to **$10^3$** on the x-axis

Quintin (MOAIS/Grenoble U.)

Dynamic Load-Balancing on Hierarchical Platforms

December 8, 2011
DAG from Makefile Execution

WSCOM vs WS

![Graph showing speed-up and bandwidth comparison between WSCOM and WS.](image-url)
Outline

1. Introduction
2. Scheduling, Communication and Efficiency
3. Enhancement for Makefile Applications
4. Enhancement for the Hierarchical Platform
   - Probabilistic Work-Stealing and Hierarchical Work-Stealing
   - Performance Analysis
5. Conclusion
Modeling Hierarchical Platforms
Request Steal Policies

**PWS: Probabilistic Work-Stealing**
- Main idea: reduce the time to find some tasks.
- Steal probability depends on the distance between computers.
- No limit on hierarchy level.

**HWS: Hierarchical Work-Stealing**
- Main idea: reduce the amount of data transferred between groups.
HWS: Reduce Expensive Steal Requests

- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.
HWS: Reduce Expensive Steal Requests

- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.
  ⇒ Steal greater amount of work.
  ⇒ Avoid stealing tasks with few amount of work.
HWS: Reduce Expensive Steal Requests

- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.
  - Steal greater amount of work.
  - Avoid stealing tasks with few amount of work.

Only one computer steal other clusters.
Hierarchical Work-Stealing

Only leaders could steal another leader

---

Quintin (MOAIS/Grenoble U.)
Dynamic Load-Balancing on Hierarchical Platforms
December 8, 2011
Hierarchical Work-Stealing

- Execute tasks in the same order as the classical work-stealing.
Hierarchical Work-Stealing

- Manage a second tasks stack restricted to its cluster.
- The Leader waits the end of block execution.
Enhancement for the Hierarchical Platform  
Performance Analysis

Execution Time Bound Analysis

\[
\frac{W_u}{P_u} + \frac{W_d}{P} + O(D \frac{B}{P_u}) + O(D) + O(\max_i \left( \frac{W_i}{P_d} \right))
\]

- Execution time depends on the work done by leaders.
Execution Time Bound Analysis

\[ \frac{W_u}{p_u} + \frac{W_d}{p} + O\left(\frac{D B}{p_u}\right) + O(D) + O\left(\max_i\left(\frac{W_i}{p_d}\right)\right) \]

- The load is balanced between all processors.
Execution Time Bound Analysis

\[
\frac{W_u}{P_u} + \frac{W_d}{P} + O(D \frac{B}{P_u}) + O(D) + O(\max_i \left( \frac{W_i}{P_d} \right))
\]

- Theoretical model limit.
- Only one block on a cluster at the same time.
Enhancement for the Hierarchical Platform

Performance Analysis

Execution Time Bound Analysis

\[
\frac{W_u}{p_u} + \frac{W_d}{p} + O(D \frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))
\]

Classical Work-Stealing.
Enhancement for the Hierarchical Platform

Performance Analysis

Execution Time Bound Analysis

\[ \frac{W_u}{p_u} + \frac{W_d}{p} + O(D \frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d})) \]

- The work inside a cluster cannot be stolen by another cluster.
- The largest block must be executed by one cluster.
Conclusion of the Theoretical Analysis

- The load seems to be balanced on the whole platform.
- The number of steal request between clusters:
  - Classical Work-Stealing: $O((p - \min p_i)D)$.
  - HWS: $O(p_u \times (D + \frac{\max_i W_i}{p_d}))$. 
Executive Platform and Application

- Genepi cluster (Grid’5000).
  - Two quad-core processors
  - High performance network: Infiniband.
- Merge-Sort.
  - Array of doubles.
  - 4 GBytes of data.
Merge-Sort with Hierarchical Heuristics

8 cores divided on two computers

- PWS
- HWS
- KWS
- CRS
- WS

on one computer

- WS (4 cores)
- WS (8 cores)
Merge-Sort with HWS and PWS

8 cores divided on two computers

- PWS
- HWS
- KWS
- CRS
- WS

on one computer

- WS
  - 4 cores
- WS
  - 8 cores

Execution time in seconds

Work-stealing policies
Outline

1. Introduction
2. Scheduling, Communication and Efficiency
3. Enhancement for Makefile Applications
   - WSCOM (Work-Stealing with COMmunication)
   - Performance Analysis
4. Enhancement for the Hierarchical Platform
   - Probabilistic Work-Stealing and Hierarchical Work-Stealing
   - Performance Analysis
5. Conclusion
Extend the Work-Stealing Utilization:

- For specific applications.
- For hierarchical platforms.
Conclusion

Extend the Work-Stealing Utilization:
- For specific applications.
- For hierarchical platforms.

Additional Information:
- DAG structure.
- Platform structure.
Conclusion

Extend the Work-Stealing Utilization:
- For specific applications.
- For hierarchical platforms.

Additional Information:
- DAG structure.
- Platform structure.

Algorithms:
- information vs performances
- practical problems vs theory
Perspectives

For WSCOM:
- Improve the pre-fetching ($\text{WSCOM}_pf$).
- Experiments WSCOM inside DSMake.
- Propose a new DAG generator.
- Parallelize the compilation of a Linux distribution.

For hierarchical platforms:
- Extend theoretical results
- Handle heterogeneous platforms.
- Automatically manage the limit inside HWS.
Conclusion

General DAG

Problem
Conclusion

General DAG

Problem

Resolved before the execution: a spanning tree.
Resolved during the execution: FIFO.
Resolved before the execution: a spanning tree.
Resolved during the execution: FIFO.
WSCOM$_{fork}$