WSCOM: Online task scheduling with data transfers

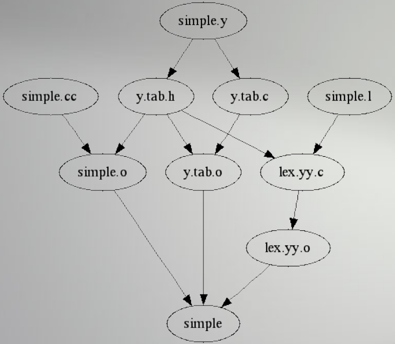
Quintin Jean-Noël, Frédéric Wagner

MOAIS research team, INRIA/LIG, University of Grenoble

May 15, 2012

Processor Evolution

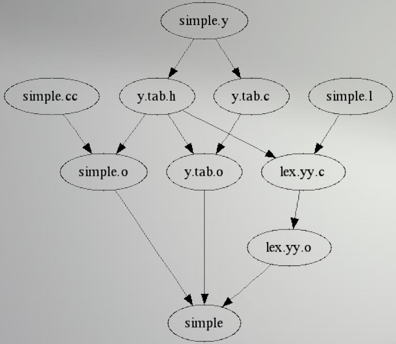
Makefile example



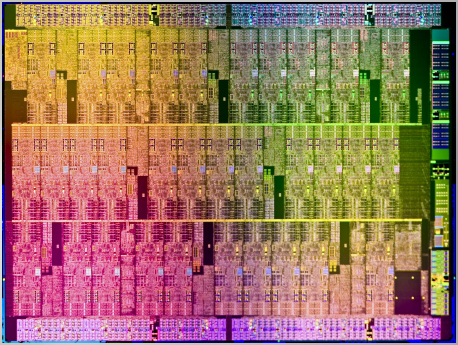
Cluster

Processor Evolution

Makefile example

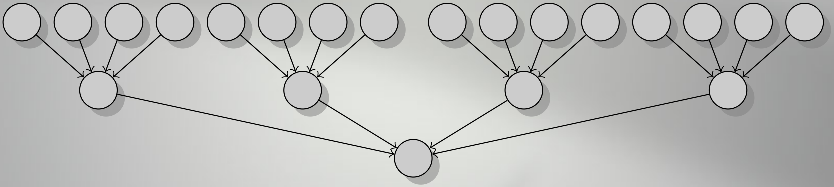


Knight Corner



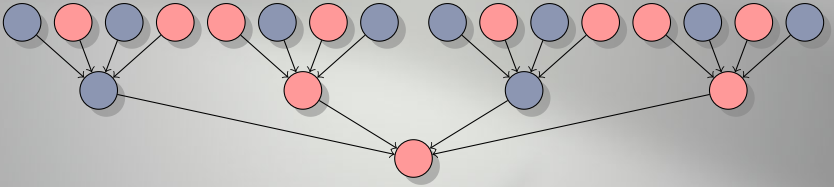
Possible Schedule with Work-Stealing

On two processors



Possible Schedule with Work-Stealing

On two processors



Outline

1 Introduction

2 Work-stealing

3 Enhancement for Makefile Applications

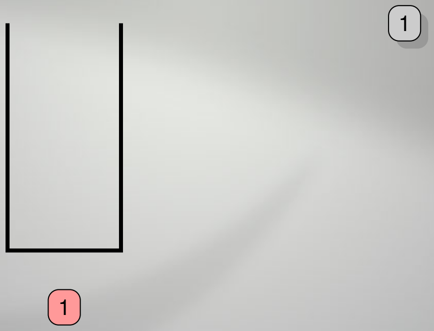
4 Conclusion

Work-Stealing [Blumofe-95]

A distributed list scheduling

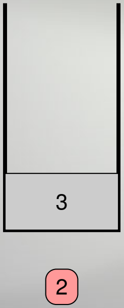
Work-Stealing [Blumofe-95]

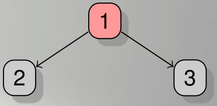
A distributed list scheduling



Work-Stealing [Blumofe-95]

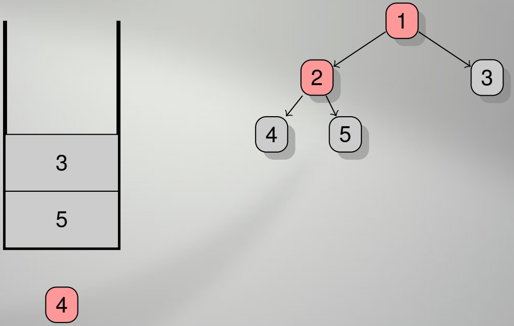
A distributed list scheduling





Work-Stealing [Blumofe-95]

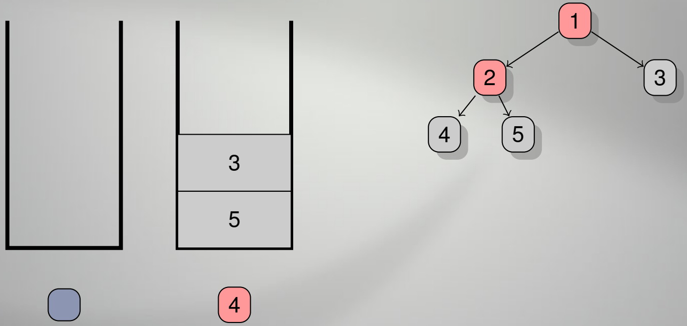
A distributed list scheduling



Work-Stealing [Blumofe-95]

A distributed list scheduling

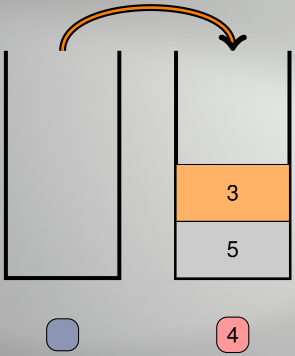
- Choose the stolen processor.

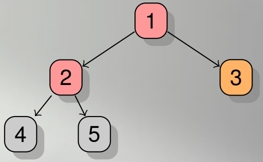


Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.
- Choose the stolen task.

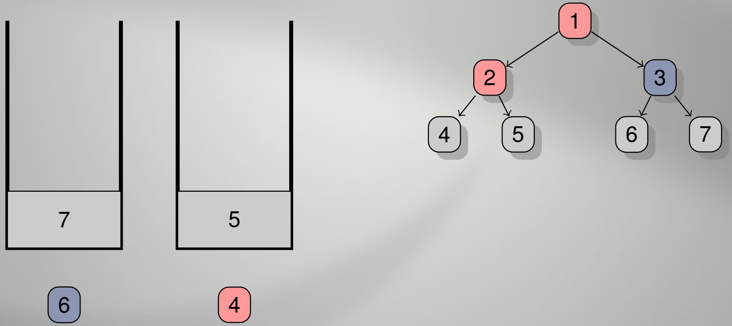




Work-Stealing [Blumofe-95]

A distributed list scheduling

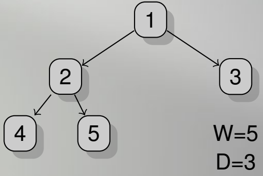
- Choose the stolen processor.
- Choose the stolen task.



Work-Stealing

Performance analysis

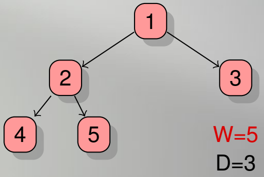
- Assumptions:
  - Constant communication time and no data transfers
  - DAG arity: 2, and unitary task
  - Homogeneous processor (Bender & Rabin for heterogeneous)
- Bounds [Arora-01]:



Work-Stealing

Performance analysis

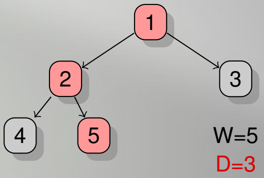
- Assumptions:
  - Constant communication time and no data transfers
  - DAG arity: 2, and unitary task
  - Homogeneous processor (Bender & Rabin for heterogeneous)
- Bounds [Arora-01]:



Work-Stealing

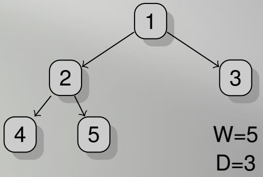
Performance analysis

- Assumptions:
  - Constant communication time and no data transfers
  - DAG arity: 2, and unitary task
  - Homogeneous processor (Bender & Rabin for heterogeneous)
- Bounds [Arora-01]:



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 : $T_p \leq \frac{W}{p} + O(D)$
  - Steal requests : $\#S \leq O(p * D)$
  - Data transfers :
    - Each steal generates at least one communication
    - The number of communication does NOT directly depend on the number of steal requests



Outline

1. Introduction
2. Work-stealing
3. Enhancement for Makefile Applications
   - WSCOM (Work-Stealing with COMmunication)
   - Performance Analysis
4. Conclusion

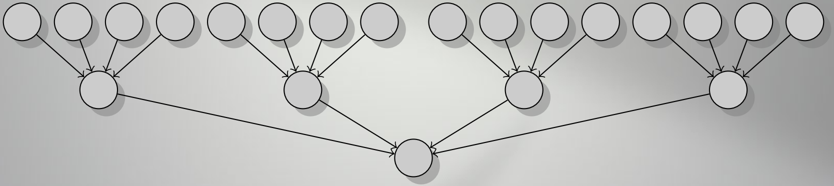
Context

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

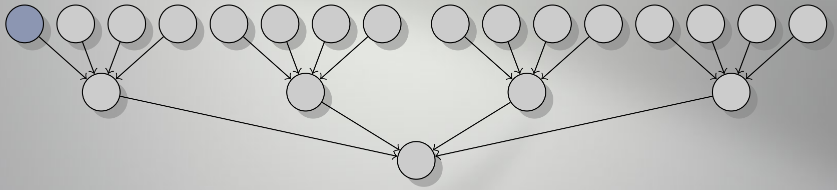
Context

- DSMake:
  - Makefiles executions on distributed platforms
  - Structure unrestricted
  - Static DAG: structure is known in advance
- Our aim:
  - Using the DAG structure to minimize the number of transfers

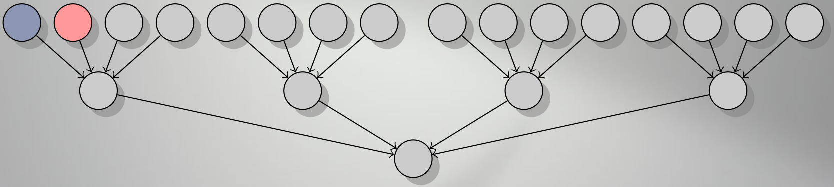
Simple Example



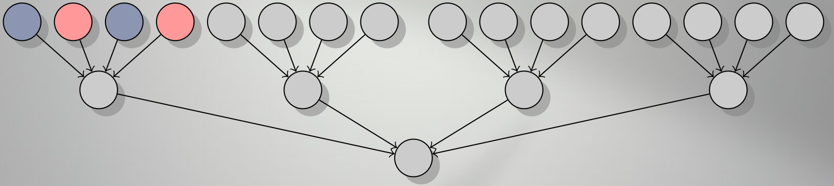
Simple Example



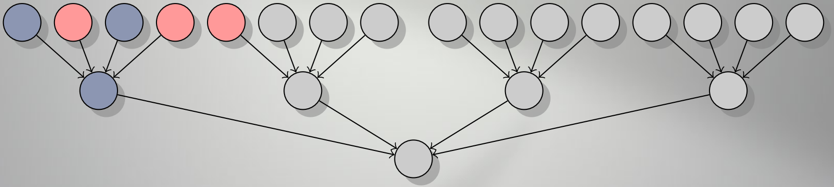
Simple Example



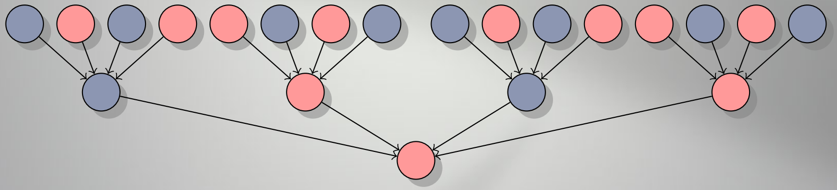
Simple Example



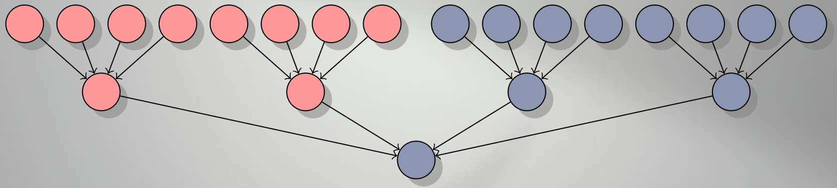
Simple Example



Simple Example

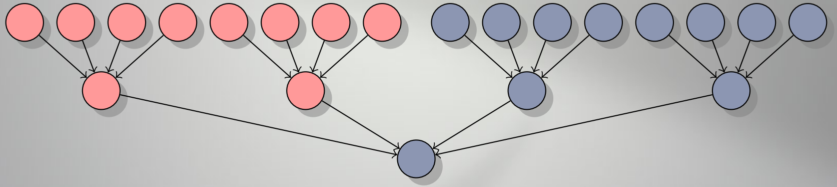


Simple Example



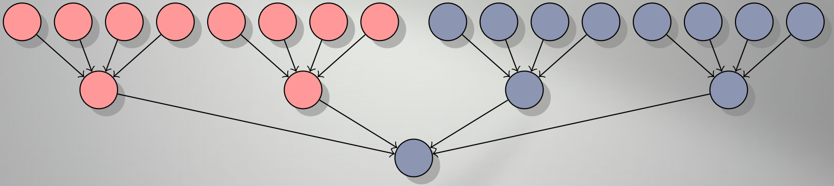
Simple Example

- The scheduling depends on tasks management



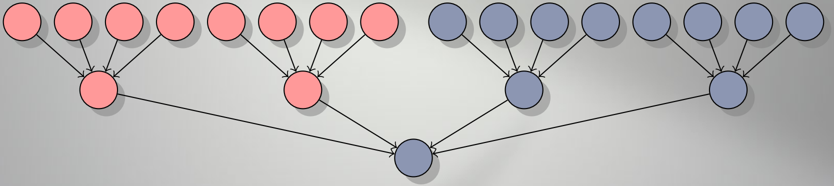
Simple Example

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

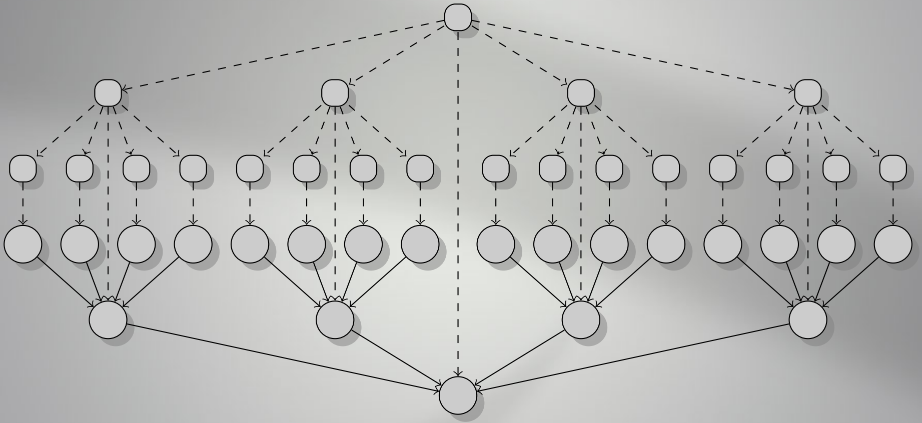


Simple Example

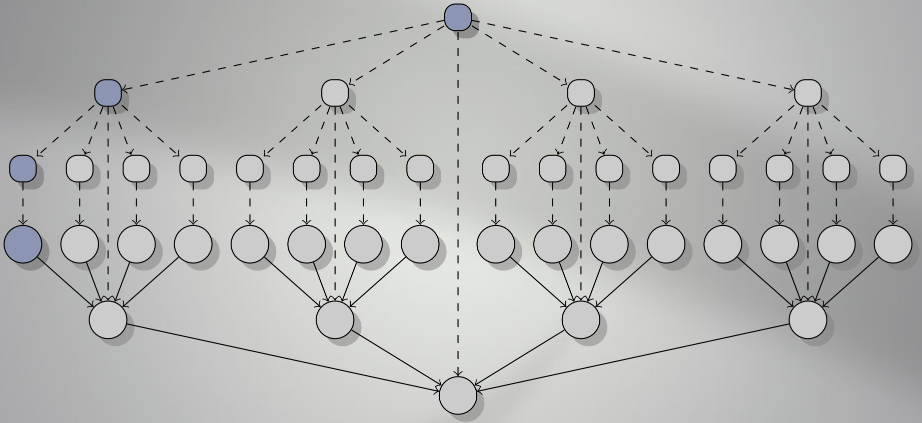
- The scheduling depends on tasks management
- Add tasks which generate a tasks block
  - Symmetry of the DAG



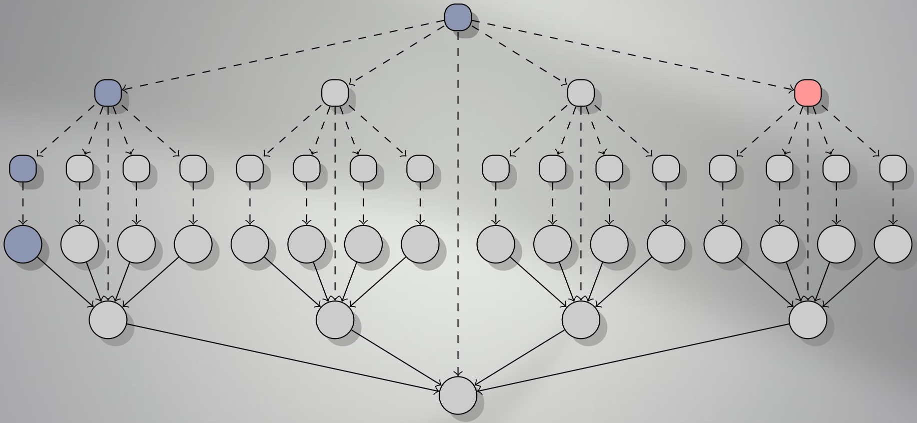
Symmetry of the DAG



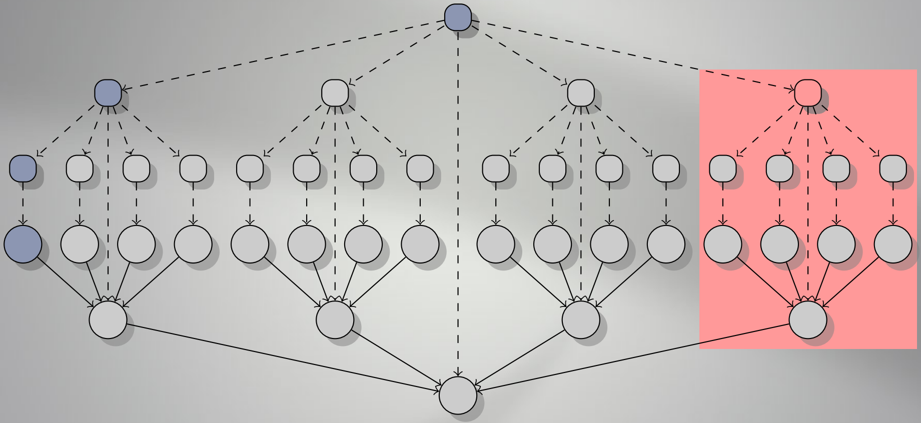
Symmetry of the DAG



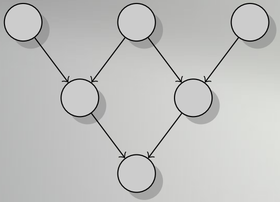
Symmetry of the DAG



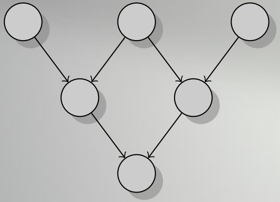
Symmetry of the DAG

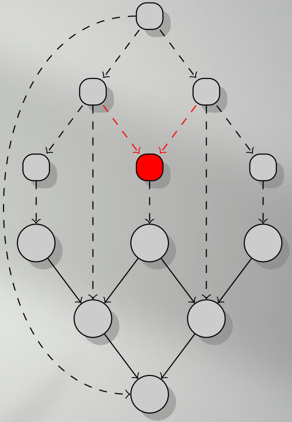


WSCOM on General DAG

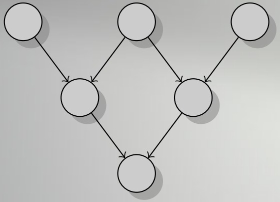


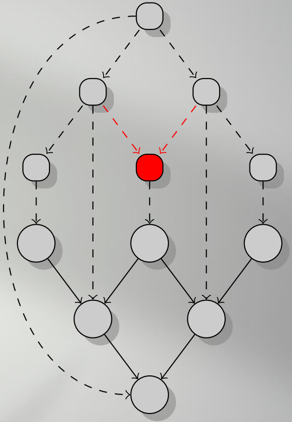
WSCOM on General DAG





WSCOM on General DAG





- Resolved before the execution: a spanning tree
- Resolved during the execution: FIFO

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 (WSCOM$_{pf}$)
     - Send data the latest (WSCOM)

Outline

1 Introduction

2 Work-stealing

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

4 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

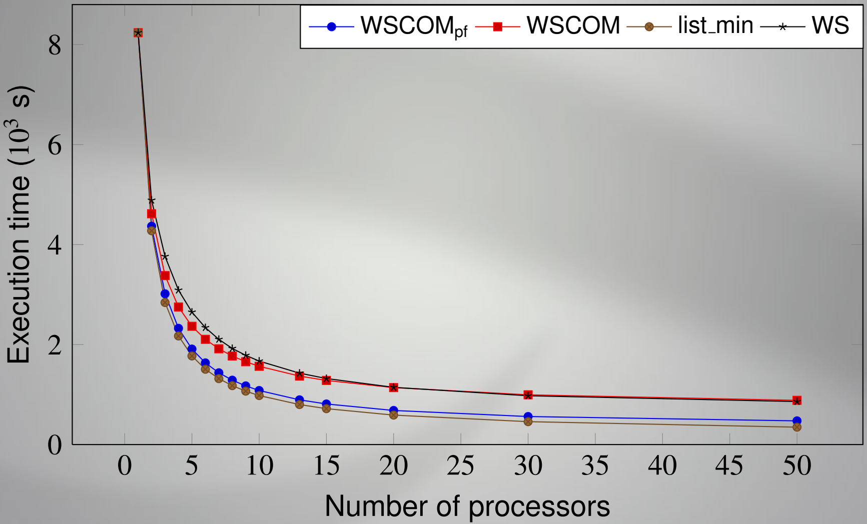
- Platforms:
  - Clique without network contentions
  - Cluster with network contentions
- Application DAG:

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

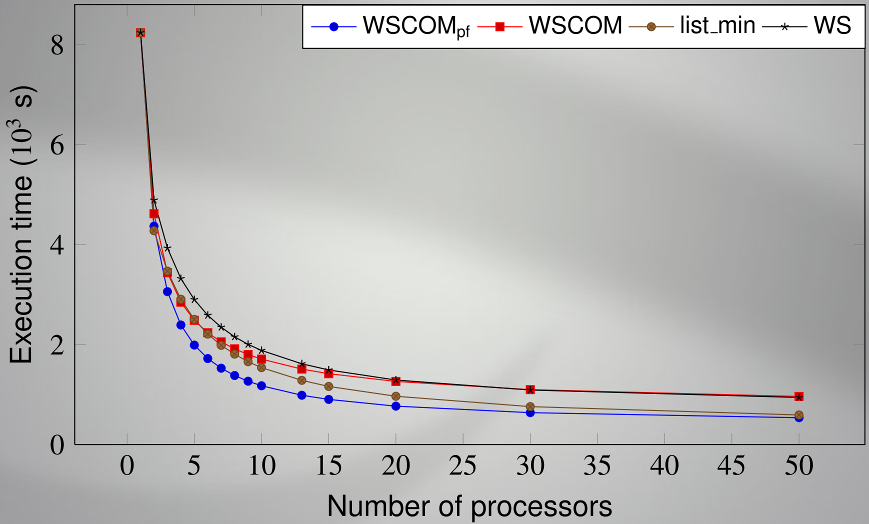
No Contention on Links

Clique platform, random DAG



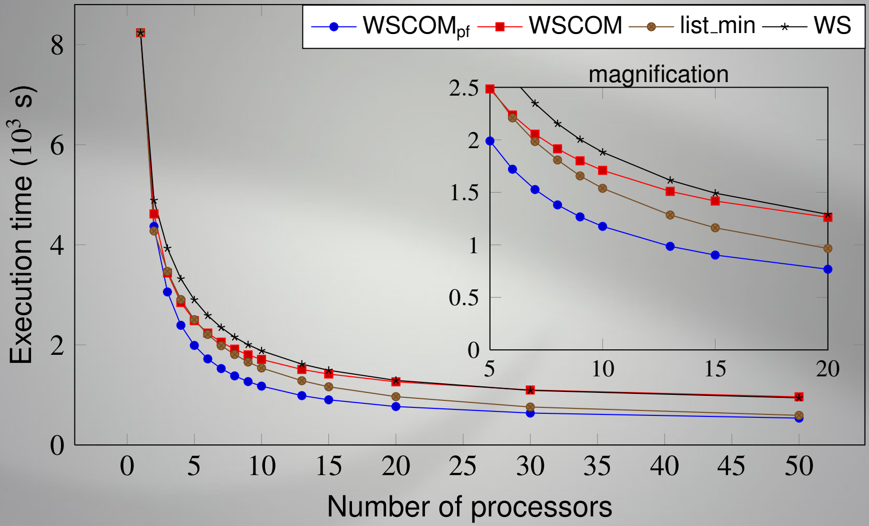
Contention on Links

Cluster Platform, Random DAG

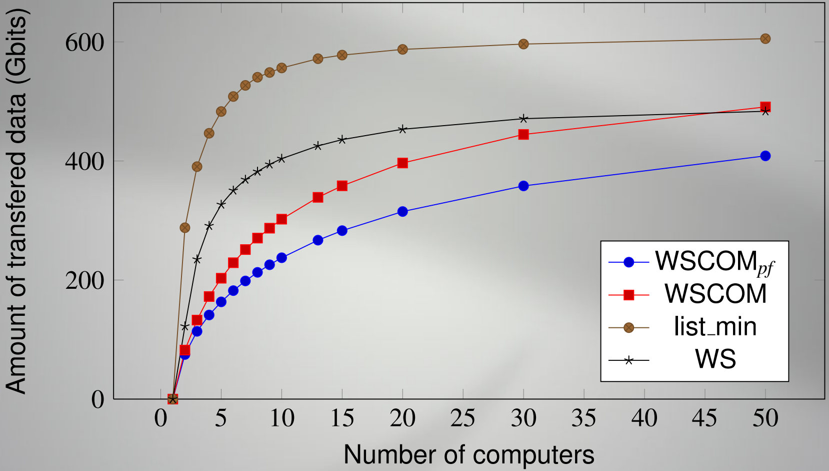


Contention on Links

Cluster Platform, Random DAG



Data Transfers



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
    - Sligthly different results
  - Highlight the ability to exploit different platforms:

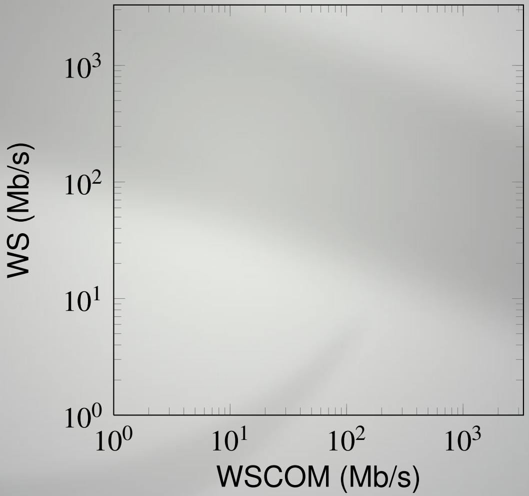
DAG from Makefile Executions

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
    - Sligthly different results
  - Highlight the ability to exploit different platforms:
    - Can WSCOM achieve a significant speed-up with a low bandwidth?

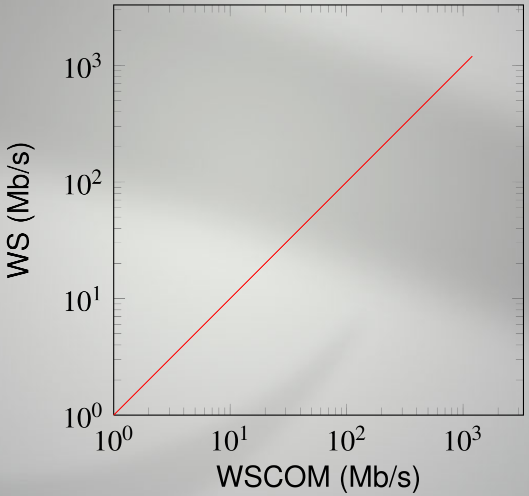
DAG from Makefile Execution

WSCOM vs WS



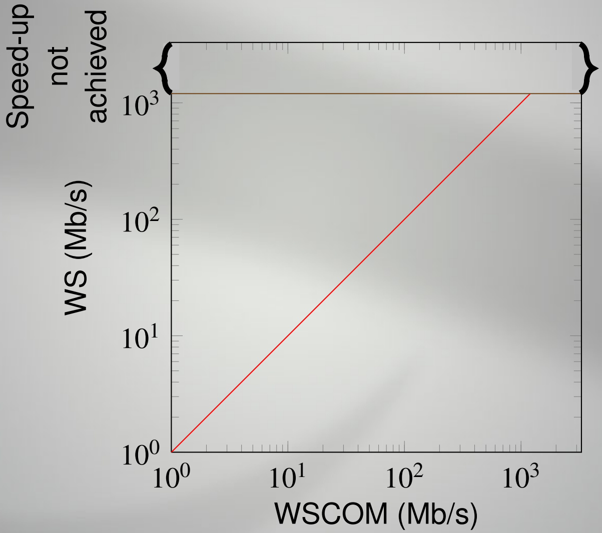
DAG from Makefile Execution

WSCOM vs WS



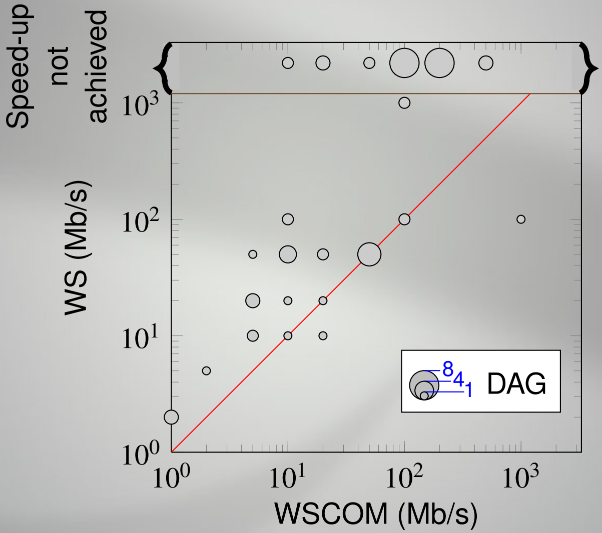
DAG from Makefile Execution

WSCOM vs WS



DAG from Makefile Execution

WSCOM vs WS



Outline

1 Introduction

2 Work-stealing

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

4 Conclusion

Conclusion

Extend the Work-Stealing Utilization:

- Reduce the sensibility to the contention by using DAG structure
- Reduce the required bandwidth

Conclusion

Extend the Work-Stealing Utilization:

- Reduce the sensibility to the contention by using DAG structure
- Reduce the required bandwidth

WSCOM:

- Information vs performances
- Data transfer

Conclusion

Extend the Work-Stealing Utilization:

- Reduce the sensibility to the contention by using DAG structure
- Reduce the required bandwidth

WSCOM:

- Information vs performances
- Data transfer

Experimental contribution:

- Workload of Makefile (MacTrA)
- DSMake: Distributed Scheduling for Makefile
- Simulator of work-stealing scheduling based on SimGrid

Perspectives

For WSCOM:

- Improve the pre-fetching (WSCOM$_{pf}$)
- Experiments WSCOM inside DSMake
- Propose a new DAG generator
- Parallelize the compilation of a Linux distribution

Software and Experiment Details:

Information available on my website:

`http://moais.imag.fr/membres/jean-noel.quintin/WSCOM/`