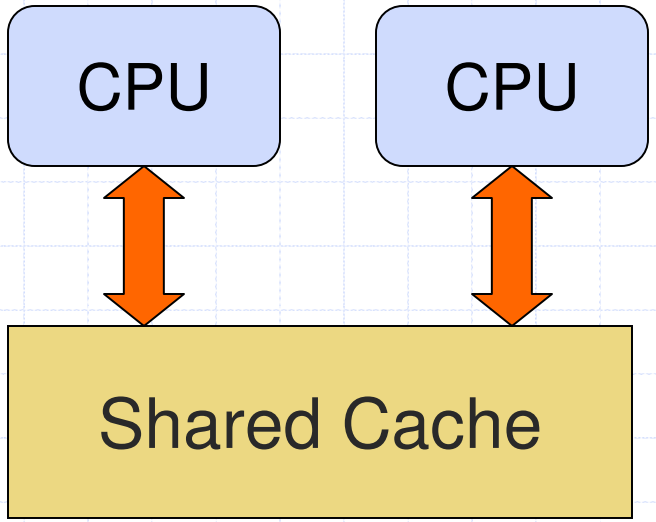
A Study on Optimally Co-scheduling Jobs of Different Lengths on CMP

Kai Tian, Yunlian Jiang and Xipeng Shen
Computer Science Department,
College of William and Mary, Virginia, USA
5/18/2009

Cache Sharing on CMP

- Advantage
  - Reduce inter-thread communication latency
  - Flexible use of cache
- Disadvantage
  - Cause resource contention among co-running processes
  - Degrade program performance and system fairness



Job Co-Scheduling

- A solution to alleviate resource contention among co-running jobs
- Capitalizes on the differences among jobs and cleverly assigns compatible jobs to same processors

Previous Work

- Symbiotic job scheduling. [Tullsen+:ASPLOS'00,SIGMETRICS'02]
- Phase co-scheduling [El-Moursy+:IPDPS'06]
- Performance isolation [Fedorova+:PACT'07]
- Thread clustering [Tam+:EuroSys'07]

... ...

Common focus:
empirical heuristics-based scheduler

An Open Question

- To obtain/approximate optimal co-schedules
  - Important for assessment of practical co-scheduling systems
  - Help expose the inherent complexity
  - Offer insights to the development of practical co-scheduling systems

Overview of This Work

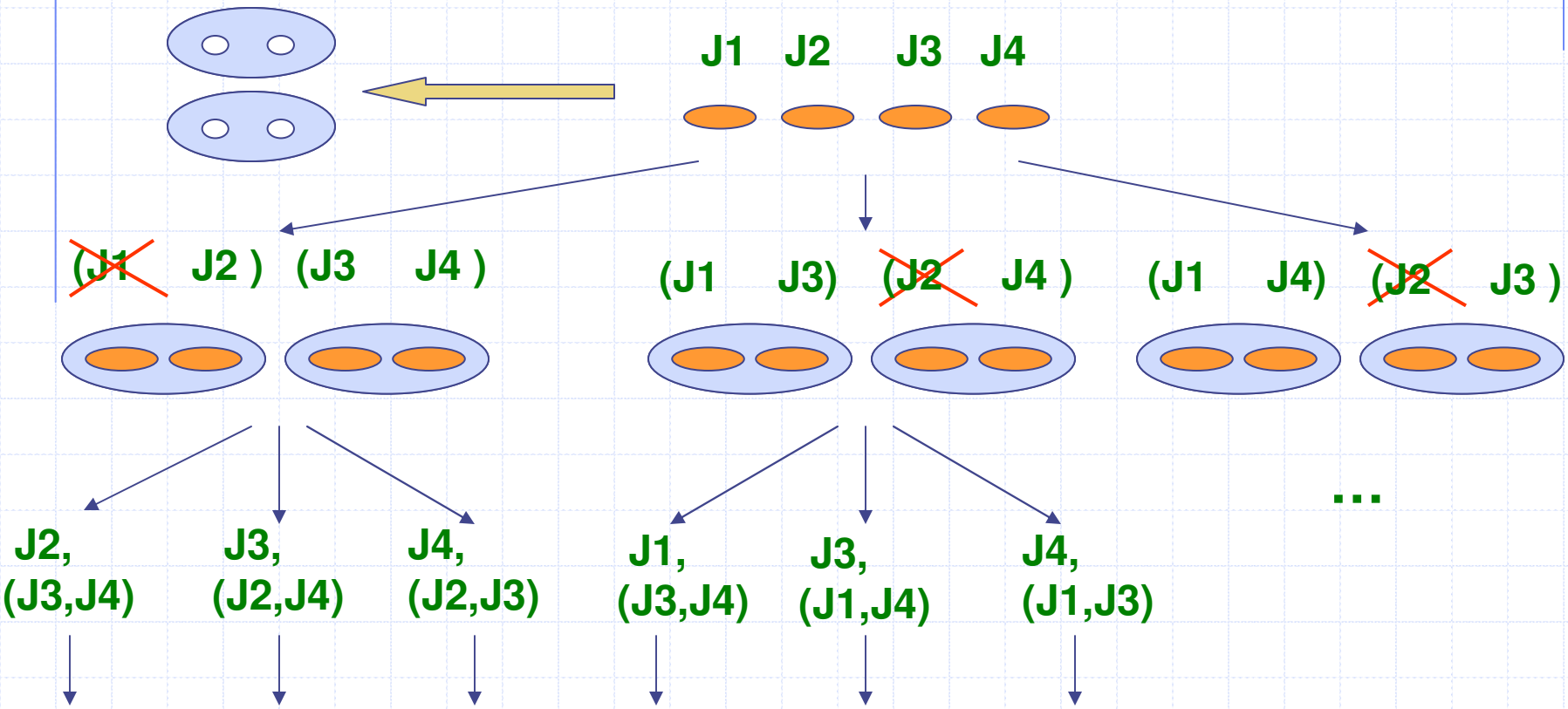
- Goal: offer theoretical insights and scalable solutions for optimal co-scheduling under certain conditions
- Two-fold contributions
  - Offer feasible ways to uncover the optimal solutions in job co-scheduling
  - Enable better evaluation of co-scheduling systems

Outline

- Problem setting
- Optimal co-scheduling algorithms
  - A*-search-based algorithm
- Approximation algorithms
  - A*-cluster-based algorithm
  - Local-matching algorithm
- Performance Evaluation
- Conclusion

Co-Scheduling Snapshot

- 4 Jobs J1, J2, J3, J4 scheduled to 2 dual-core chips



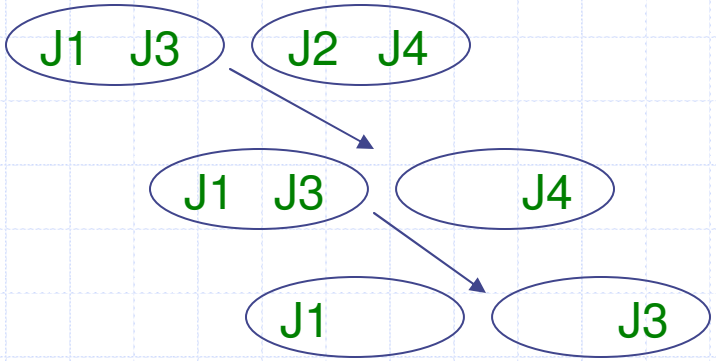
Problem Setting

- Co-run degradation for job i

$$Deg_i = \frac{cCPI_i - sCPI_i}{sCPI_i}$$

- Sub-schedule :
  - A way to assign all the unfinished jobs to different cores of processors

- Schedule :
  - A sequence of sub-schedules from start until all jobs finish



- Goal :
  - Find the optimal scheduling among different schedules to minimize total running time of all jobs

Assumptions

- No cross-chip impact on performance
- Jobs start at the same time
- #Jobs = #cores
- Phase change is ignored
- No migration cost

Outline

- Problem setting
- Optimal co-scheduling algorithms
  - A*-search-based algorithm
- Approximation algorithms
  - A*-cluster-based algorithm
  - Local-matching algorithm
- Performance Evaluation
- Conclusion

Co-Schedule Space

Each node represents a sub-schedule of the remaining jobs.

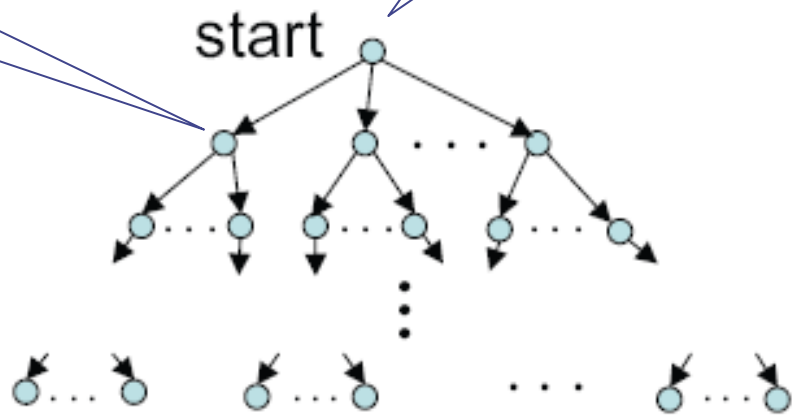
contains all jobs to be scheduled

stage 1: co-sched. *N* jobs

stage 2: co-sched. *N-1* jobs

⋮

stage N: co-sched. *1* job



Search Tree of Optimal Job Co-scheduling

Brute-force Search

- Enumerate every possible sub-schedules at each stage
  - 4 jobs : 3 (number of sub-schedules)
  - 8 jobs : 105
  - 12 jobs: 10395

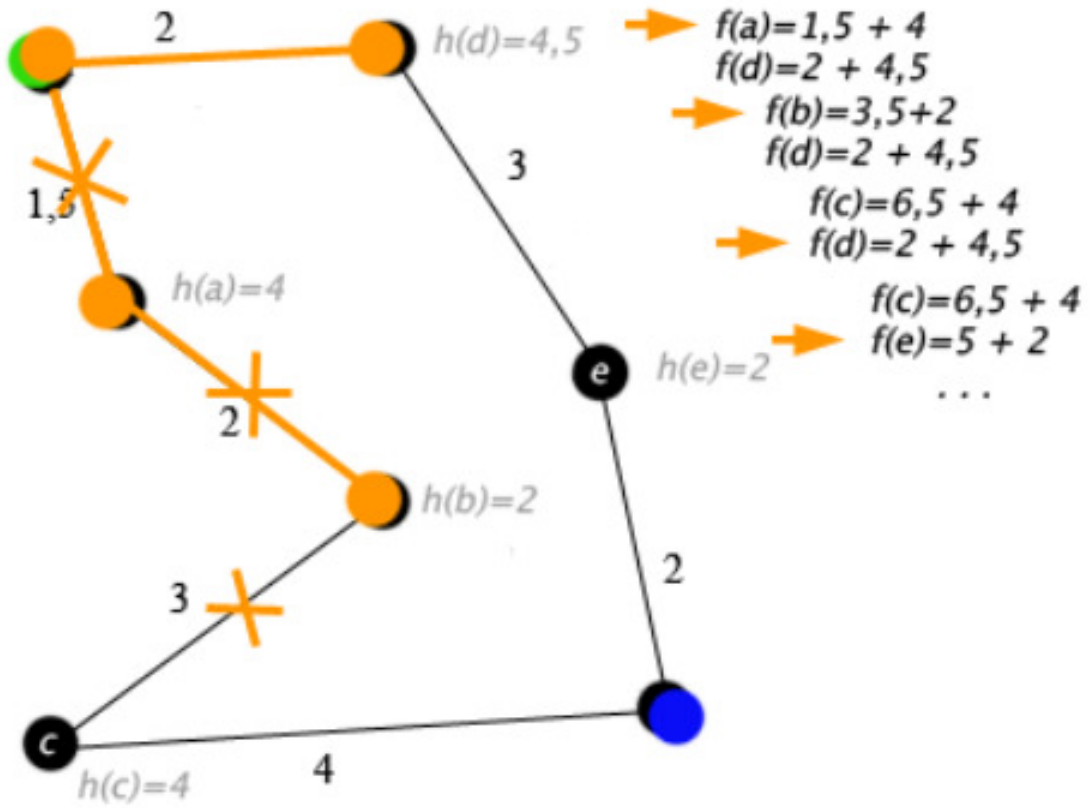
    …
- Recursively search the whole tree space
- Scheduling time increases exponentially to the number of jobs

A* search algorithm

- Comes from Artificial Intelligence
  - Heuristic-based graph search algorithm that finds the least-cost path from a given initial node to one goal node
  - Effectively avoids visiting some portion of the search space that are impossible to contain the optimal solutions
  - Proved to be optimally efficient for any given heuristic function

A* search algorithm

- An example



A* in Job Co-Scheduling

- Goal:
  - Compute a path with minimal total running time from a start node **S** (contains all jobs to be scheduled) to a goal node **G**(#Remaining jobs <= #Processors)

- Cost function at node *n*:

  Each node contains a job list at a certain state

  - *g(n)*:
    - The total running time from the start node **S** to node n
  - *h(n)*:
    - Estimated total time to finish all jobs from node n to goal node **G**
  - *f(n) = g(n) + h(n)*
    - Current estimated total running time for all jobs to finish in node n

A* in Job Co-scheduling (cont'd)

- Priority queue
  - Initially only contains the start node **S**
  - Nodes in queue are always sorted, priority is proportional to 1/f(n)
  - Each time pop out a node with lowest f value and expand the node
  - Compute f value for newly expanded nodes and add them into priority queue
  - Stops when the top node in the queue is a goal node

Outline

- Problem setting
- Optimal co-scheduling algorithms
  - A*-search-based algorithm
- Approximation algorithms
  - A*-cluster-based algorithm
  - Local-matching algorithm
- Performance Evaluation
- Conclusion

Approximation Algorithms

- A* search algorithm
  - Reduces the searching space and time significantly
  - High requirement of memory space
    - Keep all open nodes in the priority list
    - Run out of memory with more than 12 jobs
- Two approximation algorithms
  - A* cluster
    - Combine A* and clustering techniques together
  - Local-matching
    - Select locally optimal sub-schedule at each stage

A*-cluster Algorithm

- Add clustering techniques to A* algorithm
  - Cluster jobs based on running time under current sub-schedule at each stage

    Eg: J1: 20ms, J2: 202ms, J3: 24 ms, J4: 210 ms
    ➔ Two clusters: {J1, J3} and {J2, J4}

  - Generate sub-schedules based on clusters
    - jobs in the same cluster are considered same
    - Eg: {J1,J3} {J2,J4} ➔ [(J1, J2), (J3, J4)] = [(J1, J4), (J2, J3)]

  - Reschedule until all the jobs in the first cluster finish

Local-matching algorithm

- Explores only one path from start node to goal node in the tree searching space
- At each scheduling point, select the sub-schedule that minimizes the total time at current state
  - Under condition that no rescheduling happens
- The key idea
  - Select a local optimum case for each stage, to approximate the optimal case
  - Minimum weight perfect matching for 2-core
  - Hierarchical perfect matching for k-core (k>=3)

Outline

- Problem setting
- Optimal co-scheduling algorithms
  - A*-search-based algorithm
- Approximation algorithms
  - A*-cluster-based algorithm
  - Local-matching algorithm
- Performance Evaluation
- Conclusion

Evaluation Setup

- CMP co-scheduling
  - Quad-core Intel Xeon 5150 processors, 2.66 GHz
  - Two 4MB L2 cache per chip
  - 32KB L1 data cache for each core
- SMT co-scheduling
  - Intel Xeon 5080 processors, 3.73 GHz
  - Two 2MB L2 cache per chip
  - Hyper-threading enabled
- Evaluate the scheduling algorithms on 14 programs
  - 12 programs randomly selected from SPEC CPU2000
  - 2 parallel programs from SPLASH-2
    - (two threads for each parallel program)

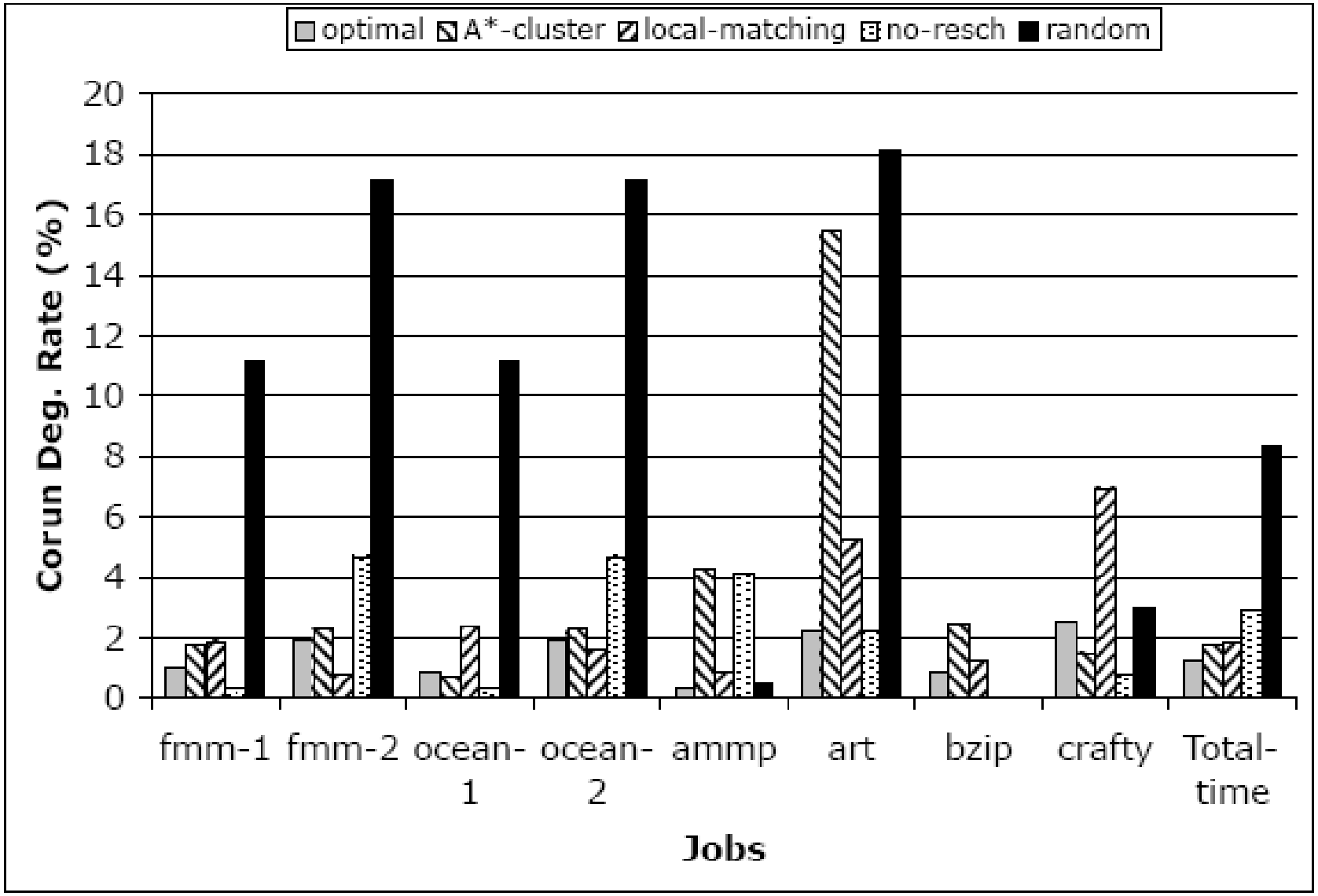
Methodology

- Collect average corun degradation of every possible corun
- Apply scheduling algorithms to those corun degradations offline to get minimized total running time

8 jobs on Quad-Core CMP

algorithm	visited nodes	scheduling time (s)	total exec time (s)	deg. rate (%)
brute-force	16 M	470	80.3	1.3
A*	7760	0.3	80.3	1.3
A*-cluster	11	0.008	80.6	1.7
local-matching	4	0.06	80.7	1.8
no-resch	1	0.02	81.5	2.9
random	-	-	85.9–89.2	8.4–12.5

8 jobs on Quad-Core CMP



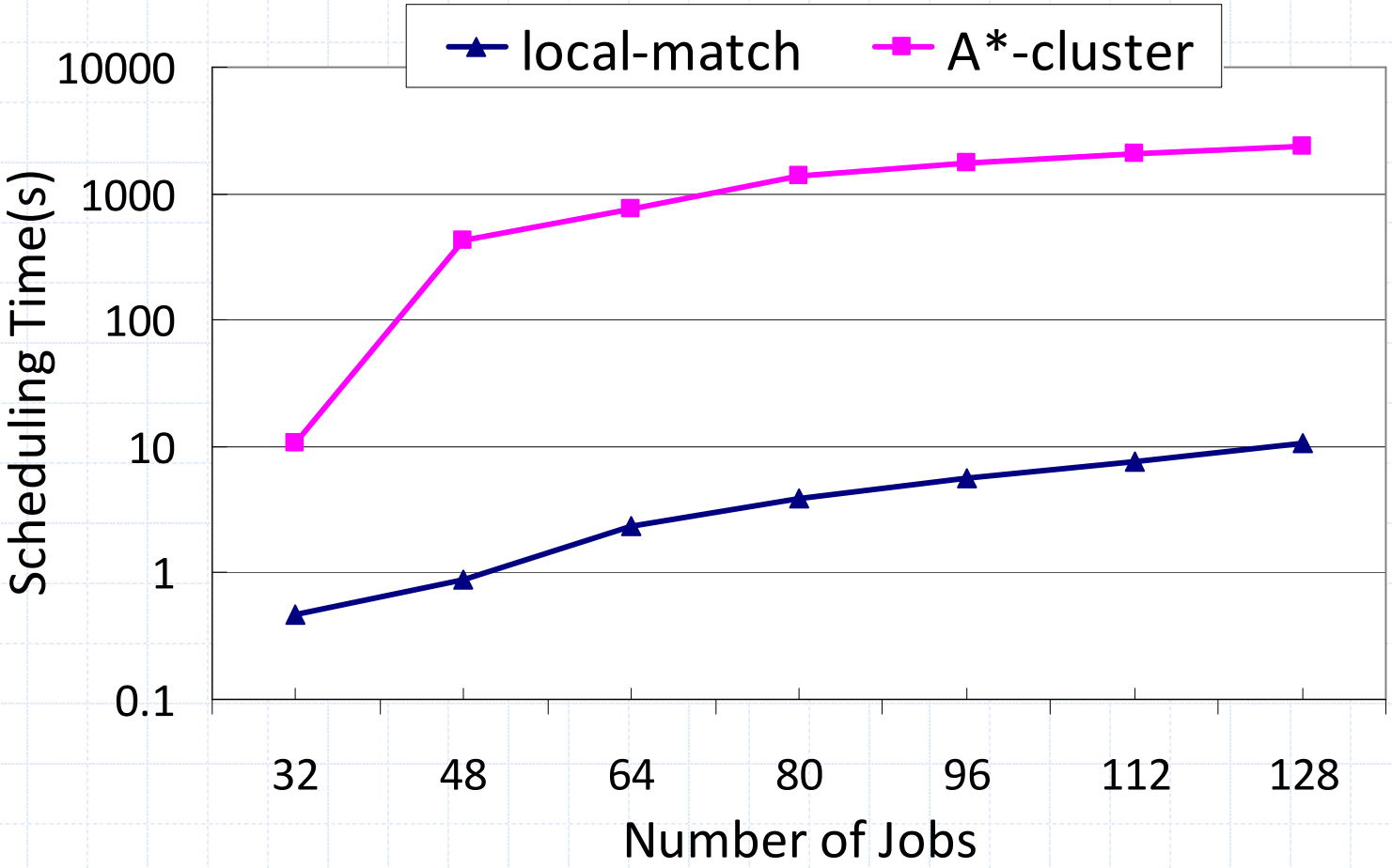
16 jobs on Quad-Core CMP

algorithm	visited nodes	sched. time (s)	total exec time (s)	deg. rate (%)
A*-cluster	721	109	149	3.2
local-matching	8	0.63	147	2.2
no-resch	1	0.03	150	3.7
random	-	-	159–172	9.9–19.2

16 jobs on SMT (2 threads/core)

algorithm	visited nodes	sched. time (s)	total exec time (s)	deg. rate (%)
A*-cluster	315	198	325	26
local-matching	8	0.24	315	22
no-resch	1	0.03	322	25
random	-	-	340–382	32–48

Scalability Evaluation



Conclusion

- A*-based approach reduce searching space significantly for optimal co-scheduling.
- A*-cluster and local-matching algorithms offer effective and efficient approximation.
- This work offers the insights and practical support for the evaluation of co-scheduling systems.