Detecting and Using Critical Paths at Runtime in Message Driven Parallel Programs

Isaac Dooley, Laxmikant V. Kale
Department of Computer Science
University of Illinois

idooley2@illinois.edu
kale@illinois.edu

12th Workshop on Advances in Parallel and Distributed Computational Models
IPDPS
April 19, 2010

Motivation

- Critical paths historically have been used important in post-mortem (offline) parallel performance analysis.
- Can they be computed online in message driven parallel languages?
- Is critical path information useful in running parallel HPC programs?

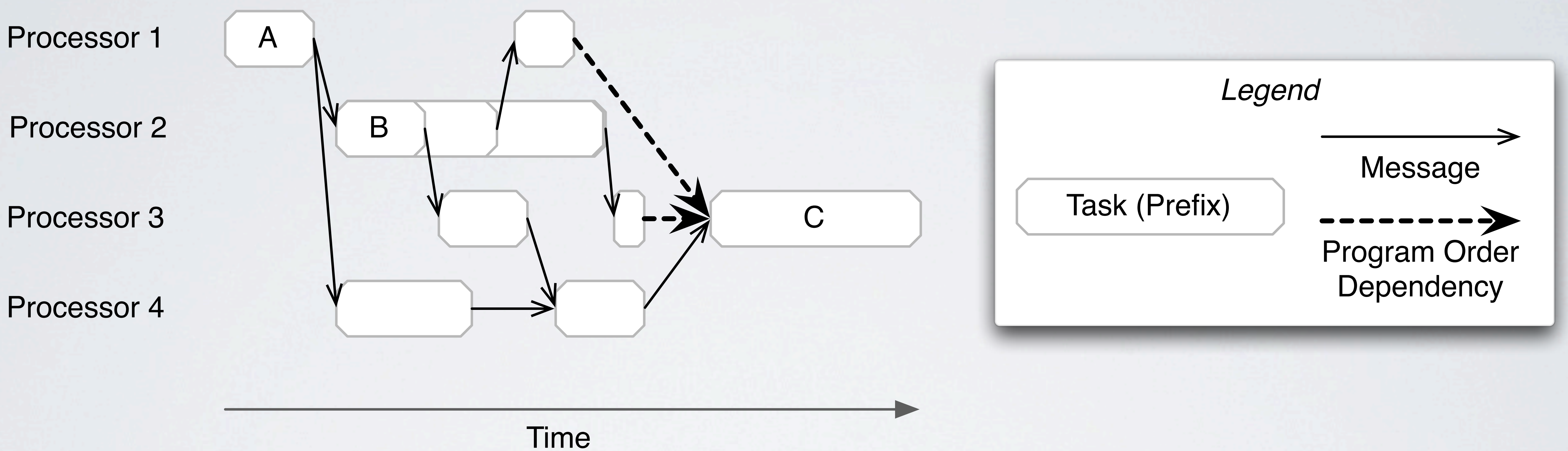
Critical Paths in Parallel Programs

- Existing algorithms for recording critical paths in a hybrid online/offline manner:
  - J Hollingsworth. *An online computation of critical path profiling.* In SPDT '96: Proceedings of the SIG-METRICS symposium on Parallel and distributed tools, pages 11–20, New York, NY, USA, 1996
  - J Hollingsworth. *Critical path profiling of message passing and shared-memory programs*. Parallel and Distributed Systems, IEEE Transactions on, 9(10):1029– 1040, Oct 1998
  - C Yang, B P Miller. *Path Analysis for the Execution of Parallel and Distributed Programs.* IEEE Transactions on Parallel and Distributed Systems, pages 1029 - 1040, Oct 1998
  - M Schulz. *Extracting critical path graphs from MPI applications*. Cluster Computing, 2005, pages 1 – 10, Sep 2005
  - For guiding expert post-mortem performance analysis
  - For visualizing parallel program execution to gain understanding

Critical Paths in Message Driven Parallel Programs

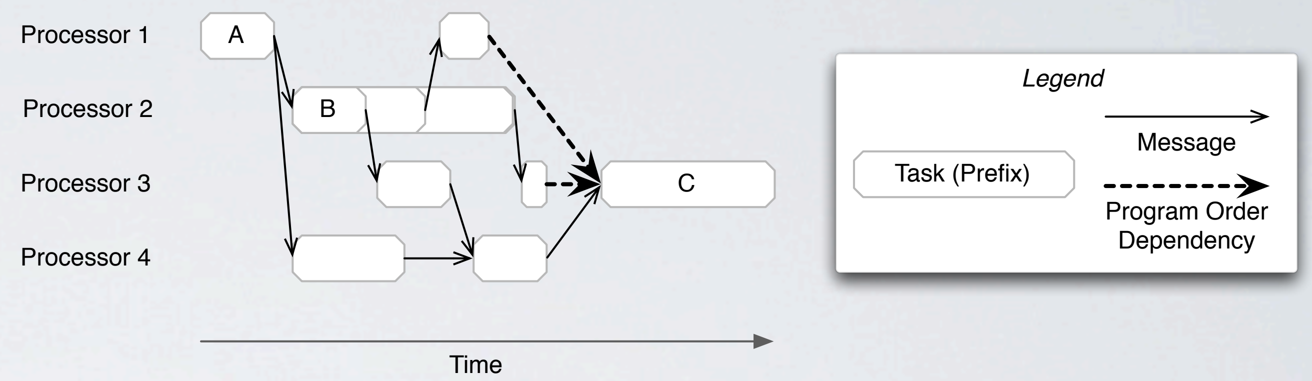
- Message Driven Execution (as implemented in Charm++):
  - Tasks invoke methods asynchronously
  - An asynchronous method invocation results in:
    - New local task in queue, or
    - Message sent to remote processor, resulting in new task

Example
Program Activity Graph



- Critical path profiles represent a path through the Program Activity Graph (PAG) composed of computation and messages.

Program Activity Graph



- The PAG can be recorded as a program runs in a distributed graph

- Path weights include computation time, but not message send time

Processor 1

	Task Prefix Index	In-Edge (processor, Index)
A	1	initial
	2	(2,2)

Processor 2

	Task Prefix Index	In-Edge (processor, Index)
B	1	(1,1)
	2	(1,1)
	3	(1,1)

Processor 3

	Task Prefix Index	In-Edge (processor, Index)
	1	(2,1)
	2	(2,3)
C	3	(1,2) or (3,2) or (4,2)

Processor 4

Task Prefix Index	In-Edge (processor, Index)
1	(1,1)
2	(3,1) or (4,1)

Finding Critical Paths

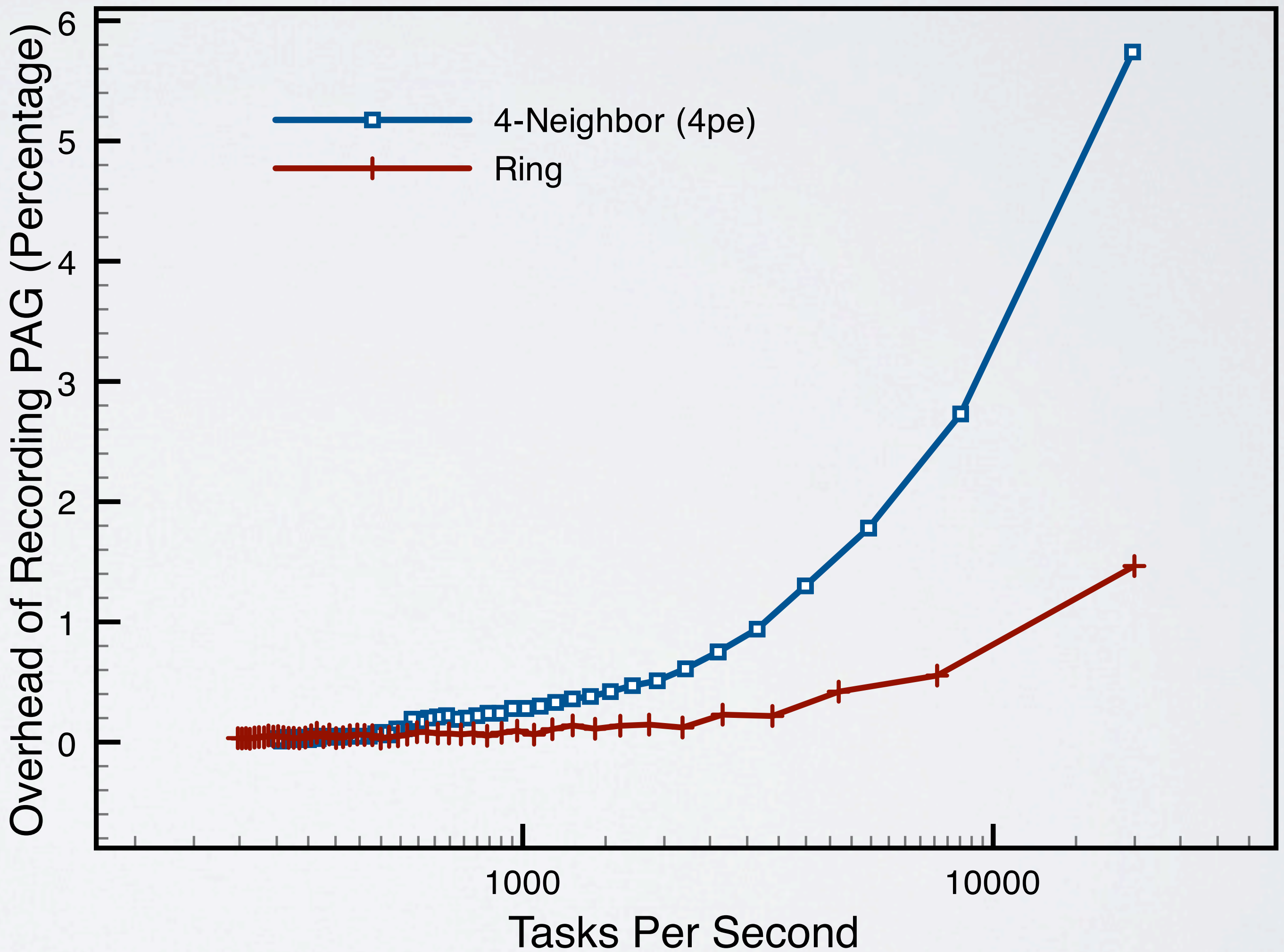
- Record PAG as program runs
  - Augment each message with:
    - an identifier
    - path length
  - Record maximal incoming path for each task in a table
    - Requires compiler support or code modifications
- Retrieve Critical Path for any task with a backwards traversal

Implementation

- Implemented in the Charm++ runtime system.
- Supports multiple languages:
  - Charm++
  - Structured Dagger
  - Charisma
- Trickiness is in how multiple incoming dependencies are captured.
  - Reductions
  - User maintains knowledge of dependencies satisfied by earlier tasks
  - Language specific dependency mechanisms

Costs of Recording Critical Paths

- Cost of extra 8 bytes in message
- Cost of adding table entries for each task execution
    - Microbenchmarks:



- Cost of backwards traversal retrieval: Application Dependent

Use: Automatic Task Priorities

- Automatically Tuning Task Priorities:
  - OpenAtom Application
  - Record critical path for 20 iterations, then switch to new priorities based on observed critical path.
  - **10.2% speedup** when prioritizing critical path task types

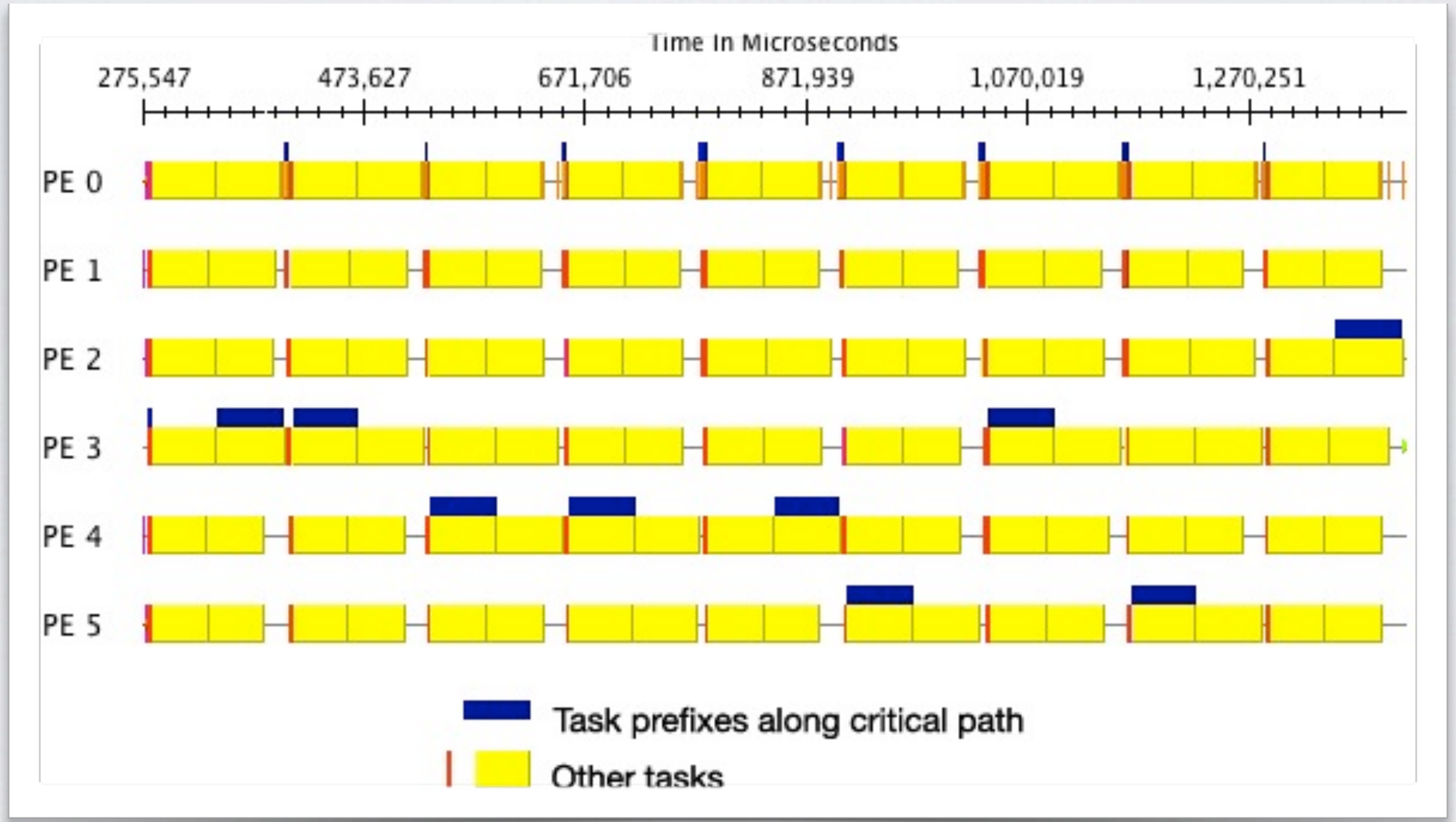
Uses: Phase Detection

- Critical path is retrieved
- Frequently repeated subpaths are extracted
- Cheap!

a b b b b b b c d e f g g g g g h i j b b b b b k l m n o p o p q r b b b b s t i u v w b b b b x x y v w b b b b
b A A x y v w b b b b x x x y v w b b b b b x x x y v w b b b b b x x x a b b b b b b c d e f g g g g g h i j b
b b b b b k l m n o p o p q r b b b b s t i u v w b b b b b A A x y v w b b b b b A A x y v w b b b b b A A x
y v w b b b b b x x x y v w b b b b b x x x **a b b b b b b c d e f g g g g g h i j b b b b b k l m n o p o p q r**
b b b b s t i u v w b b b b x x y v w b b b A A x y v w b b b b x x x y v w b b b b A x x y w b b b A A x a b
b b b b b c d e f g g g h i j b b b k l m n o p o p q r b b b b s t i u v w b b b b b A x x y v w b b b b b A x x
y v w b b b b b x x x y v w b b b b x x x y w b b b b x x A **a b b b b b b c d e f g g g g g h i j b b b b b k l**
m n o p o p q r b b b b s t i u v w b b b b x x y v w b b b b x x x y v w b b b b x x x y v w b b b b x x x y v
w b b b b b x x x **a b b b b b b c d e f g g g g g h i j b b b b b k l m n o p o p q r b b b b s t i u v w b b b b**

Uses: Performance Analysis

- Visualization:



Uses: Filter Performance Data

- Reducing volume of performance analysis data
  - Filter out processors not on critical path
  - Performance analyst only needs to manipulate & view fewer files

Conclusion

- Our Contribution:

  Critical paths can be recorded and used in message driven parallel programs at runtime for tuning message priorities.

Thanks & Questions

Detecting and Using
Critical Paths at Runtime
in Message Driven Parallel Programs

Isaac Dooley, Laxmikant V. Kale
Department of Computer Science
University of Illinois

12th Workshop on Advances in Parallel and Distributed Computational Models
April 19, 2010

Handling Multiple Input Dependencies