

Advanced Computer Systems Architecture

Chip-Multiprocessors: Applications and Architectures

CSE 561M

Prof. Patrick Crowley

Plan for Today

- Questions
- Discuss plan for research papers
 - Papers are available, start early!
- Discuss performance optimization and evaluation

Research Papers

- Objective:
 - Approach existing ideas with newly-gained knowledge of one multi-core processor, the IXP
- We will read and discuss papers from 4 areas
 - Classic multiprocessor systems
 - Multithreaded systems
 - Dataflow systems
 - Transactional memory

Paper Presentations

- Each paper discussion will involve
 - A 15-minute **group** presentation
 - Summarize the paper's content and contributions
 - Relate it to your project (e.g., what if you had targeted that system for your project rather than the IXP?)
 - All group members should contribute
 - A 15 minute follow-up lecture
- Each group has an assigned paper & date

Multiprocessors & Multithreading

- Presentations on Thursday, April 10th
- Wulf and Harbison's Reflections in a Pool of Processors/An Experience Report on C.mmp/Hydra
 - Presenters: Erik Church, Paul Sebert
- Seitz's The Cosmic Cube
 - Presenters: Joe Clinch, Rich Hill
- Dean and Ghemawat's MapReduce: Simplified Data Processing on Large Clusters
 - Presenters: Mart Haitjema, Ritun Patney, Shakir James

Dataflow & Transactional Memory

- Presentations on Thursday, April 17
- Smith's Architecture and Applications of the HEP Multiprocessor Computer System
 - Presenters: Steve Nann, Tam Vu Ngoc, Dan Vianello
- Dennis and Misunas' A Preliminary Architecture for a Basic Data-Flow Processor
 - Presenters: Steve Prochazka, Mark Dunn
- Shriraman et al.'s Flexible Decoupled Transactional Memory Support
 - Presenters: Eitan Marder-Eppstein, Stu Glaser

Project Logistics

- 3.5 weeks remaining!
- Weekly Milestones
 - Milestone report due each week
 - Submit proposal Feb 26
 - Final presentation Apr 15
- Focus this week:
Implementation

Project Milestones

<i>M0</i>	<i>Feb 26</i>	<i>Project Proposal</i>
<i>M1</i>	<i>Mar 4</i>	<i>Design</i>
<i>M2</i>	<i>Mar 18</i>	<i>Implementation 1</i>
<i>M3</i>	<i>Mar 25</i>	<i>Implementation 2</i>
<i>M4</i>	<i>Apr 1</i>	<i>Implementation 3</i>
<i>M5</i>	<i>Apr 8</i>	<i>Wrap-up, Prepare reports</i>
<i>M6</i>	<i>Apr 15</i>	<i>Presentations</i>

Objectives

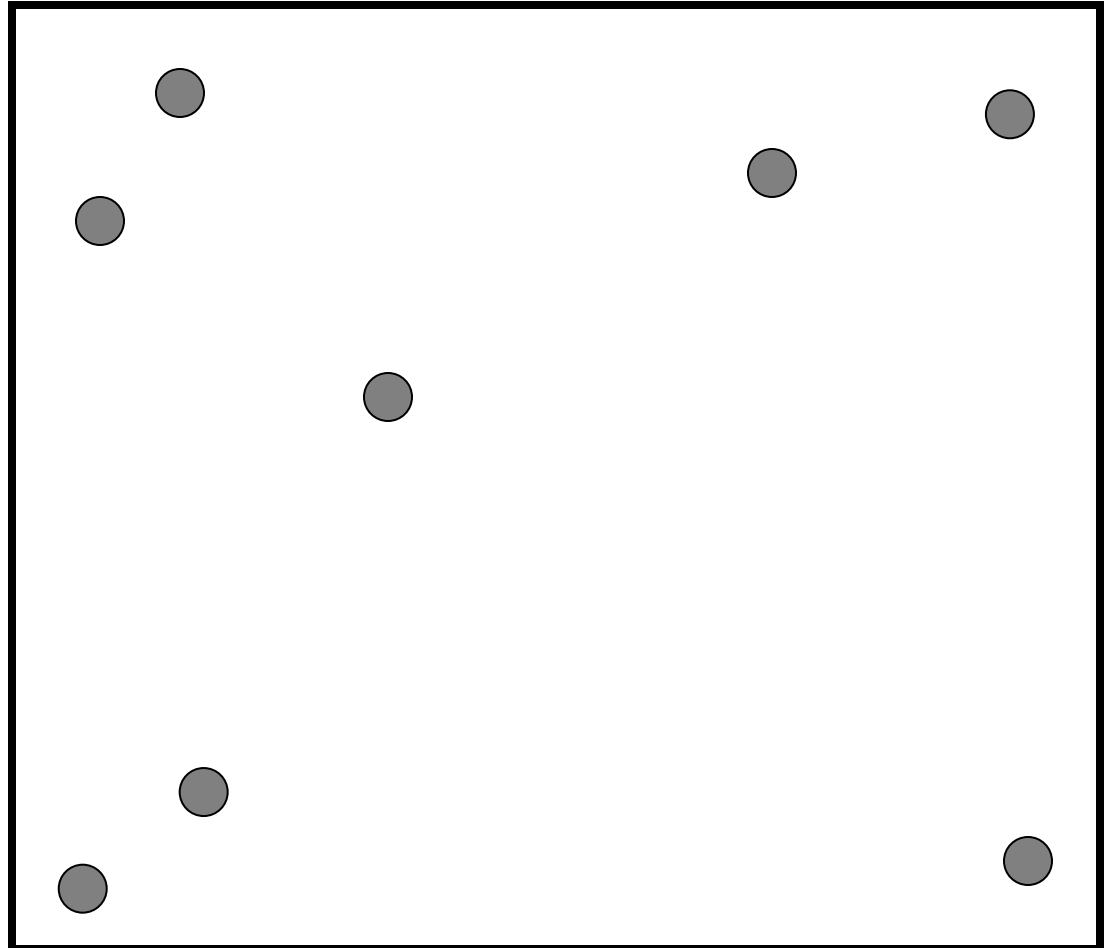
- Consider *performance optimization* in general terms
- Study a particular *performance evaluation* technique

Outline

- Performance optimization
 - Case Study
 - Principles
- Worst-case execution time
 - Motivation
 - Technique

Case Study: The N-body Problem

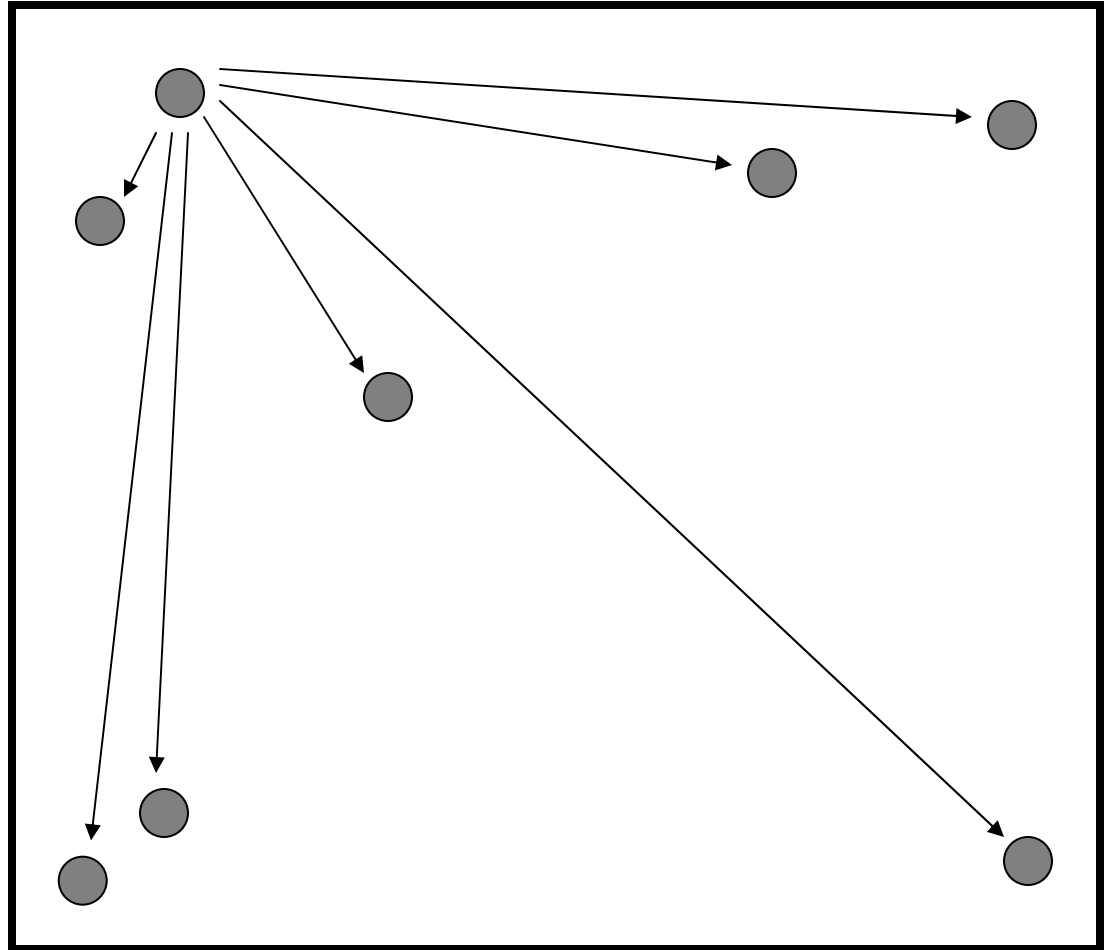
- Given masses in a gravitational field with initial positions and velocities: simulate interactions
- Each object could be a planet, e.g.



Resource: Bentley's Programming Pearls, 2nd edition

Direct Implementation

- At each (small) time step, compute each objects movement
- Key point: each object influenced by all the others
- Complexity: $O(n^2)$, i.e., slow

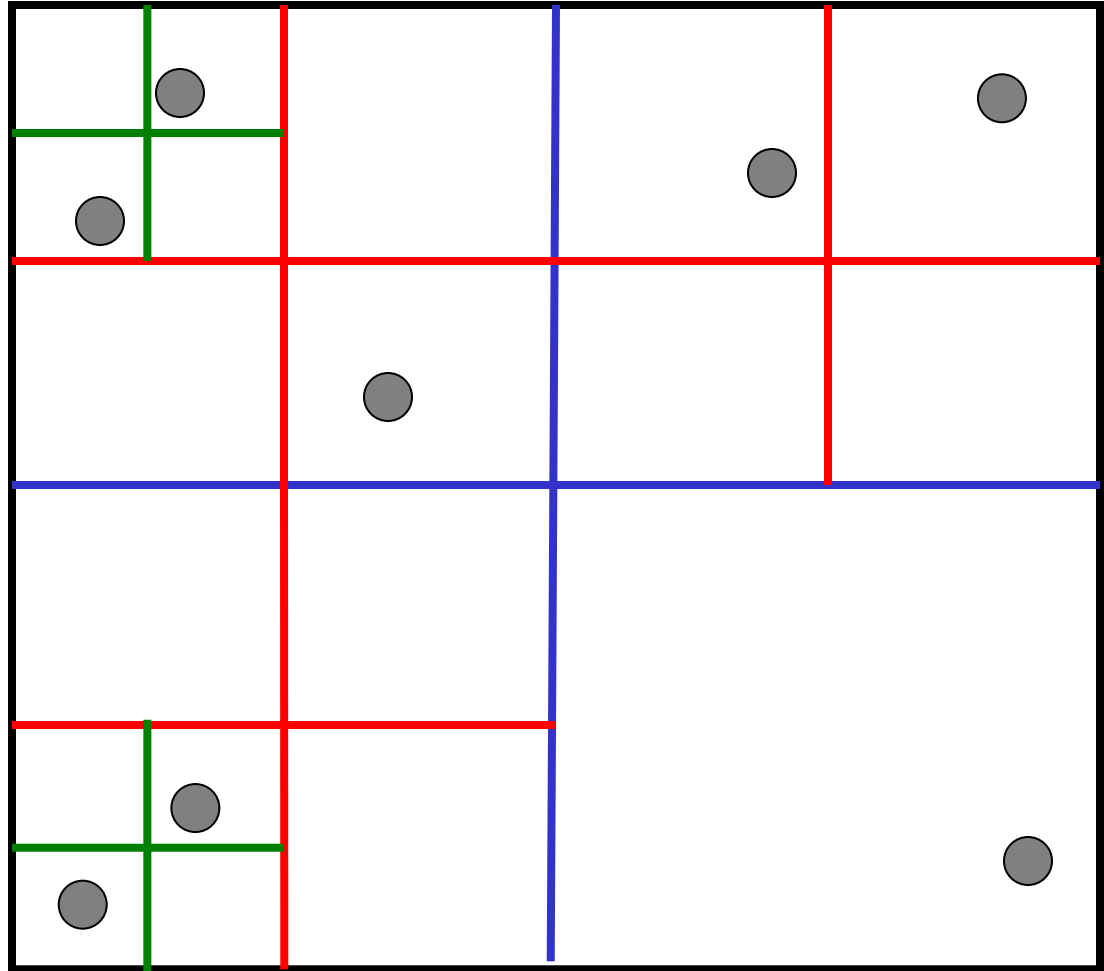


Appel's Optimization

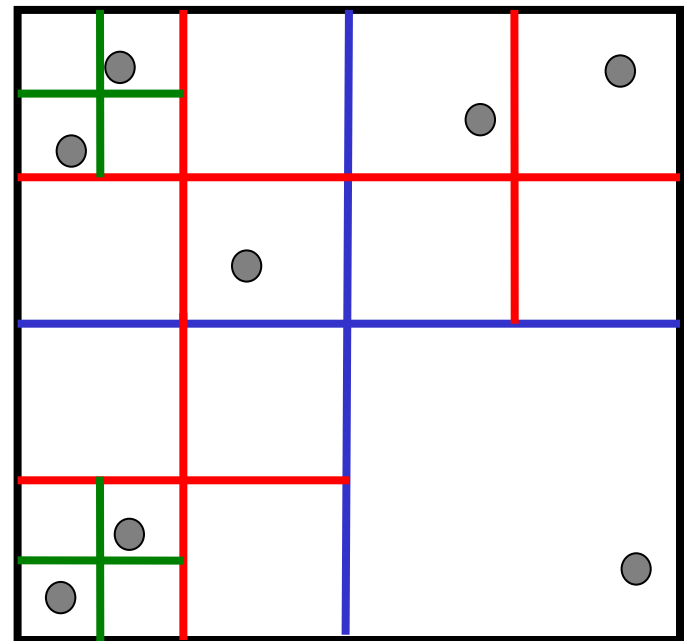
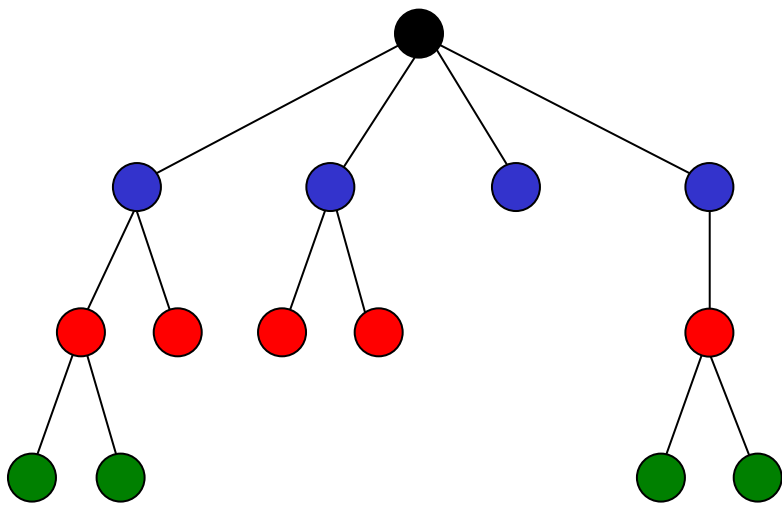
- In a 1985 paper, Andrew Appel described how he reduced the runtime for the N-body problem from one *year* to one *day*
 - “An efficient program for many-body simulations,” *SLAM Journal on Scientific and Statistical Computing*, 6, 1, pp. 85-103. (Jan '85)
- His optimizations targeted several design levels

Appel's Algorithmic Approach

- Construct a quadtree (for 2d)
- Treat *distant* clusters as a single point
- Each subsquare can record the center of mass and total mass of the particles it contains



Quadtree Representation



- Leaves are objects
- Each non-leaf is a cluster
- Given a few approximations, each object is influenced by the clusters above it, $O(n \log n)$

System-wide Contributions

Design Level	Speedup	Change
Alg and Data Structure	12	Binary tree reduces $O(n^2)$ to $O(n \log n)$
Alg Tuning	2	Larger time steps
Data Structure	2	Tree-specific D.S.
System-ind Code Tune	2	Use single-precision integers
System-dep Code Tune	2.5	Hand tuned critical function
Hardware	2	F.P. accelerator
Total	400	

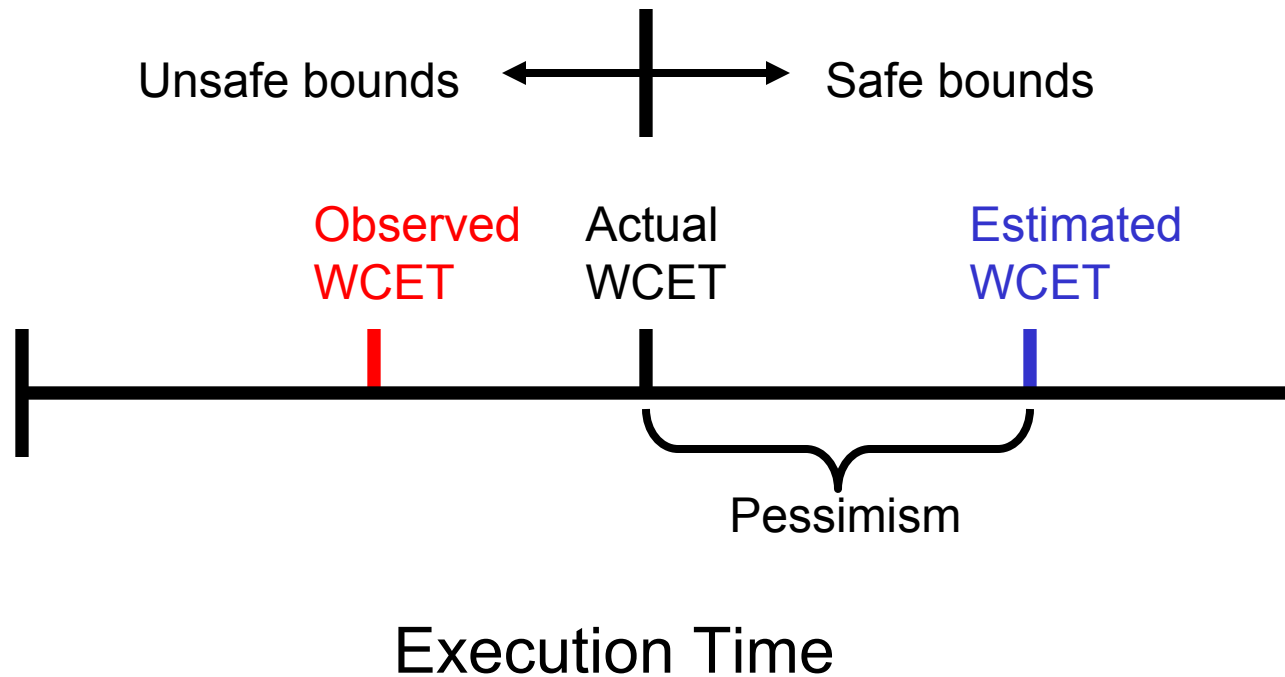
Principles

- Dramatic performance available across all design levels in the system
- Improvements in algorithms have high impact, and can lead to additional improvements
 - The tree-based organization created opportunities
- If you need a little speedup: choose the right level
- If you need a large speedup: work at multiple levels
- Optimization comes at a cost
 - Code is more complex (1200 lines vs. 40)
 - Appel spent months on this

Evaluation: Worst-Case Execution Time (WCET)

- Networks are real time systems
 - Implementations must be both *functionally* and *temporally* correct
 - If service time exceeds inter-arrival time, the system will be unstable
- Minimum reliable performance *can* be stated in terms of worst-case conditions
 - Best effort routers
 - Beyond best effort: differentiated services

WCET: Safety & Tightness



WCET on Network Processors

- Challenges:
 - Implementations are software
 - Programs run on multithreaded multiprocessors
- Related real-time systems work
 - Simple systems studied recently
 - No work on *multithreading*

Question: How do you find a safe & tight worst-case bound?

A Method for Multithreaded WCET Estimation

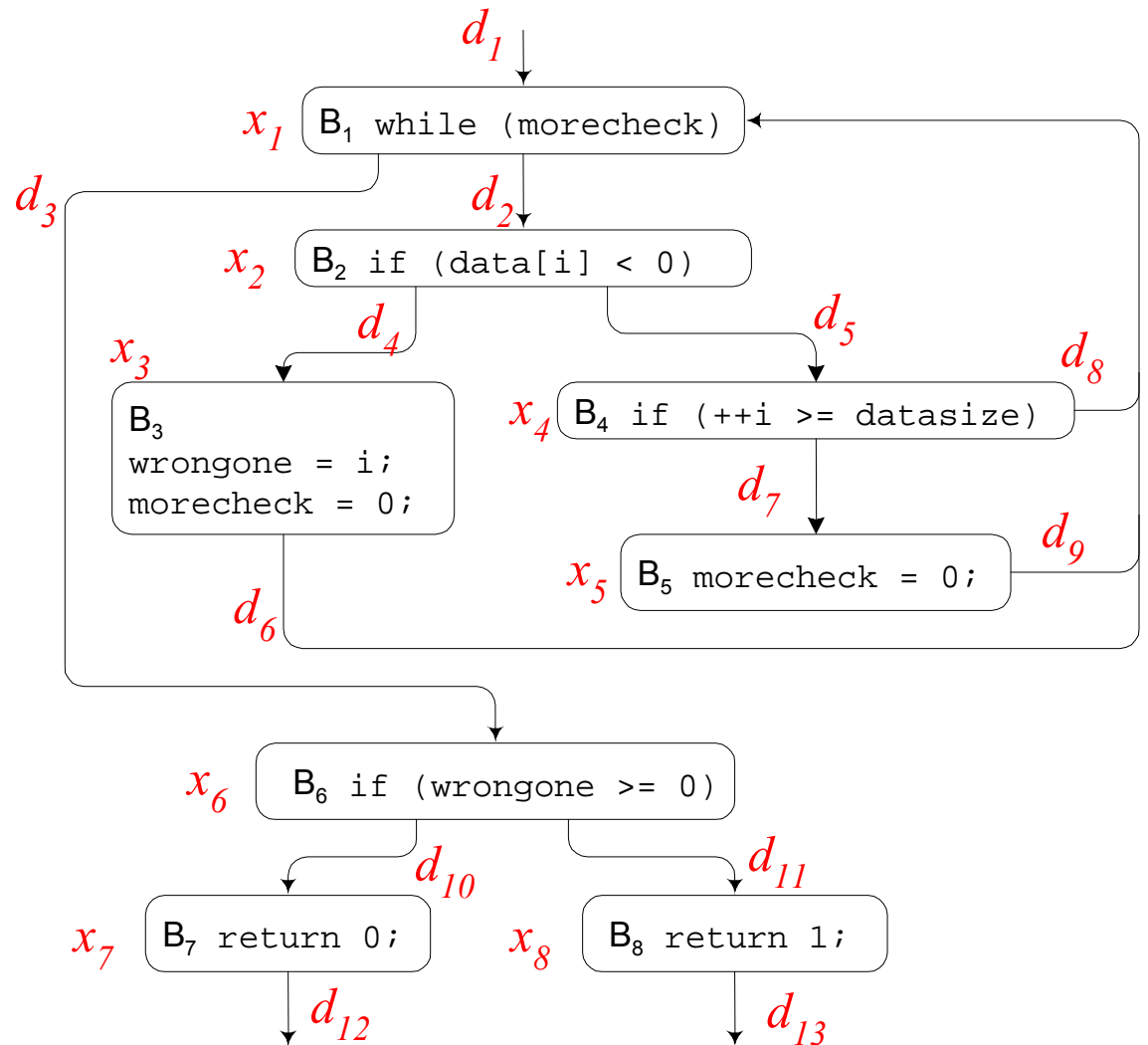
- Basic idea:
 - Use integer linear programming techniques to find the most expensive of all paths through a program's control-flow graph (CFG).
- Decidability restrictions on programs:
 - No unbounded loops
 - No dynamic data structures
 - No recursion
- Based on Implicit Path Enumeration (IPET)
 - Work by Steven Li, *et al.* from Princeton

The Problem

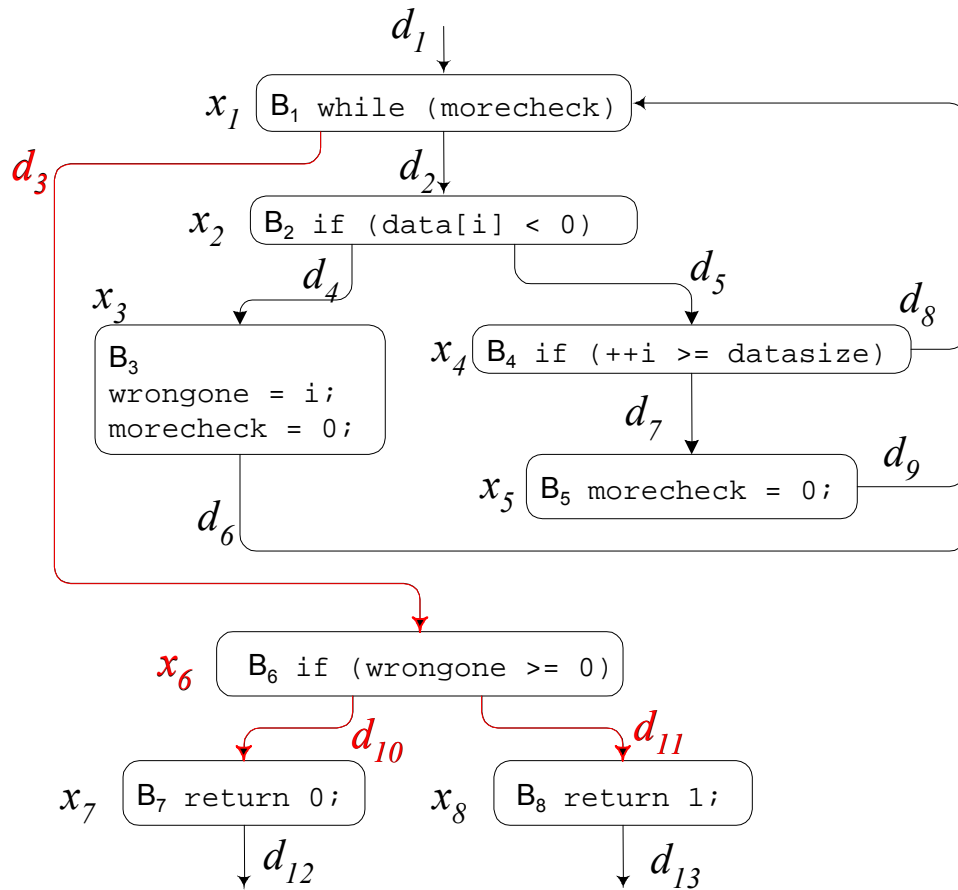
- Question 1
 - How long will one thread take to complete?
 - Assuming a simple, single issue processor
- *Question 2*
 - *How long will a group of threads take to complete?*
 - *Assuming a single issue processor with 0-cycle context switches*
- Only considering worst-case throughput here

Example

```
/*
morecheck = 1;
i = 0; datasize = 10;
wrongone = -1;
*/
while (morecheck) {
  if (data[i] < 0) {
    wrongone = i;
    morecheck = 0;
  }
  else
    if (++i >= datasize)
      morecheck = 0;
}
if (wrongone >= 0)
  return 0;
else
  return 1;
```



Structural constraints



$$d_1 = 1$$

$$x_1 = d_1 + d_6 + d_8 + d_9 = d_2 + d_3$$

$$x_2 = d_2 = d_4 + d_5$$

$$x_3 = d_4 = d_6$$

$$x_4 = d_5 = d_7 + d_8$$

$$x_5 = d_7 = d_9$$

$$x_6 = d_3 = d_{10} + d_{11}$$

$$x_7 = d_{10} = d_{12}$$

$$x_8 = d_{11} = d_{13}$$

- Flow in = flow out

Problem Statement

- Assume each basic block has a constant cost c_i
 - Based on expected latency of each operation
- The constraints bound the feasible x_i values.
- Solve for WCET by maximizing the sum:

$$\text{Program Execution Time} = \sum_{i=1}^N c_i x_i$$

Subject to the structural and functional constraints

Results

- Implementation
 - Targets the Intel IXP1200 NP
 - Takes assembly code as input
 - Publicly available LP solver
 - Logic to construct CFG, extract constraints and interface to LP solver and rendering engine
- Experiments
 - Compare WCET bound to simulated results over ‘worst-case’ input

Program	Description	IXP μ -words	Basic Blocks
<code>reverse_array</code>	Reverses an array of N elements	60	13
<code>chk_data</code>	Checks array for negative values	70	17
<code>sort</code>	Bubblesort (reverse sorted input)	103	23

Results for 1 Thread

Program	Simulated WCET	Estimated WCET
reverse_array	659	699
chk_data	530	568
sort	134794	274002

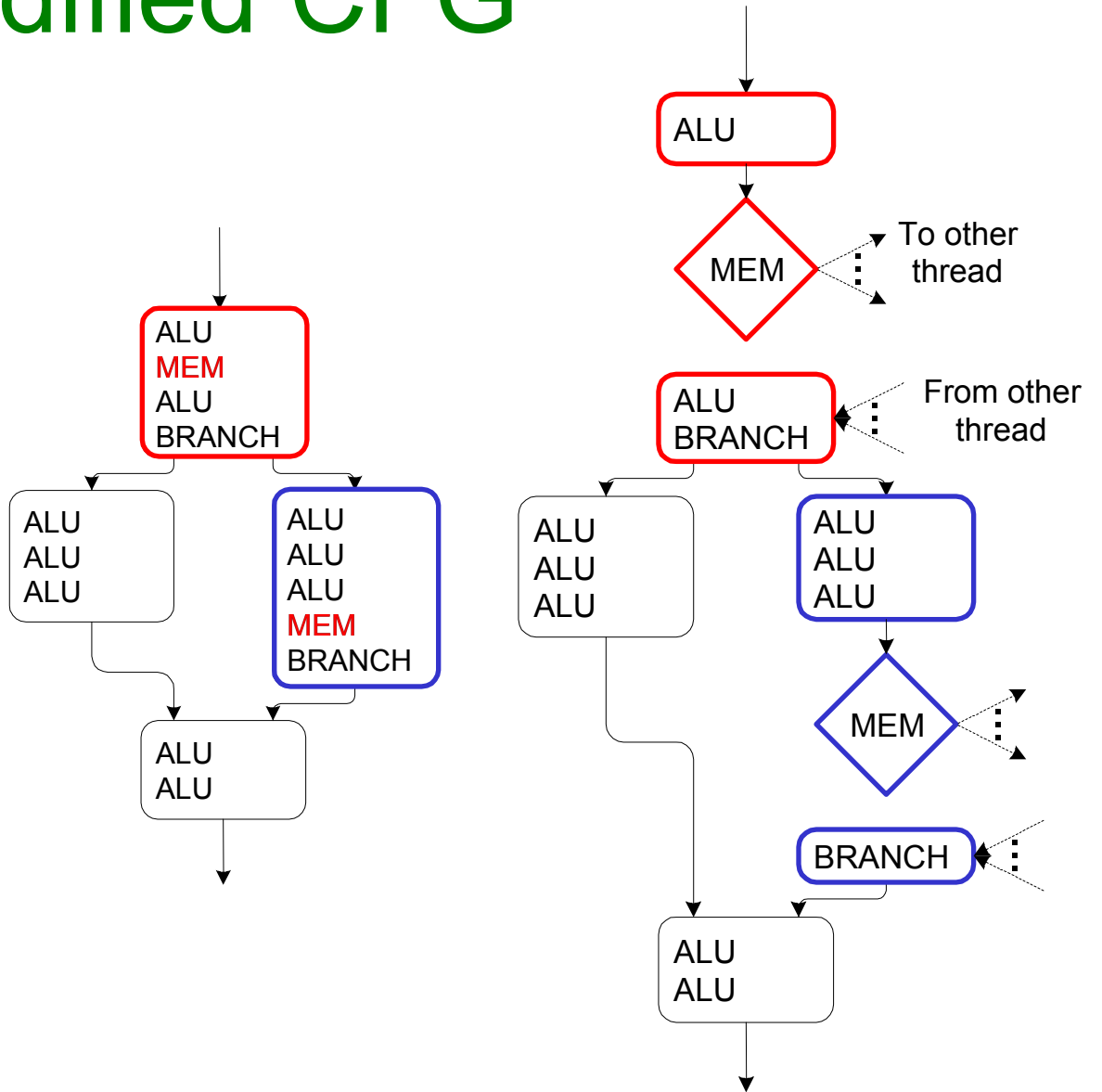
- First two apps are bounded tightly
- sort is within a factor of 2 due to pessimistic loop bounds
 - We used the ‘easy’ nested loop bound; a tighter functional constraint, if found, could improve this
 - Bubblesort theoretical worst case: $\frac{n(n-1)}{2}$

An Extension for Multithreading

- Assumptions based on Intel IXP1200:
 - Strict round robin-scheduling
 - Explicit yield control
 - Yield at all memory operations
 - A fixed yield latency
- Basic idea: retain the notion of flow and join threads at their yield operations
- Software model: a group of threads implements a single software-pipeline stage

Modified CFG

- Step 1: Add yield nodes
- Step 2: Add yield edges
- Step 3: Conserve yield flow



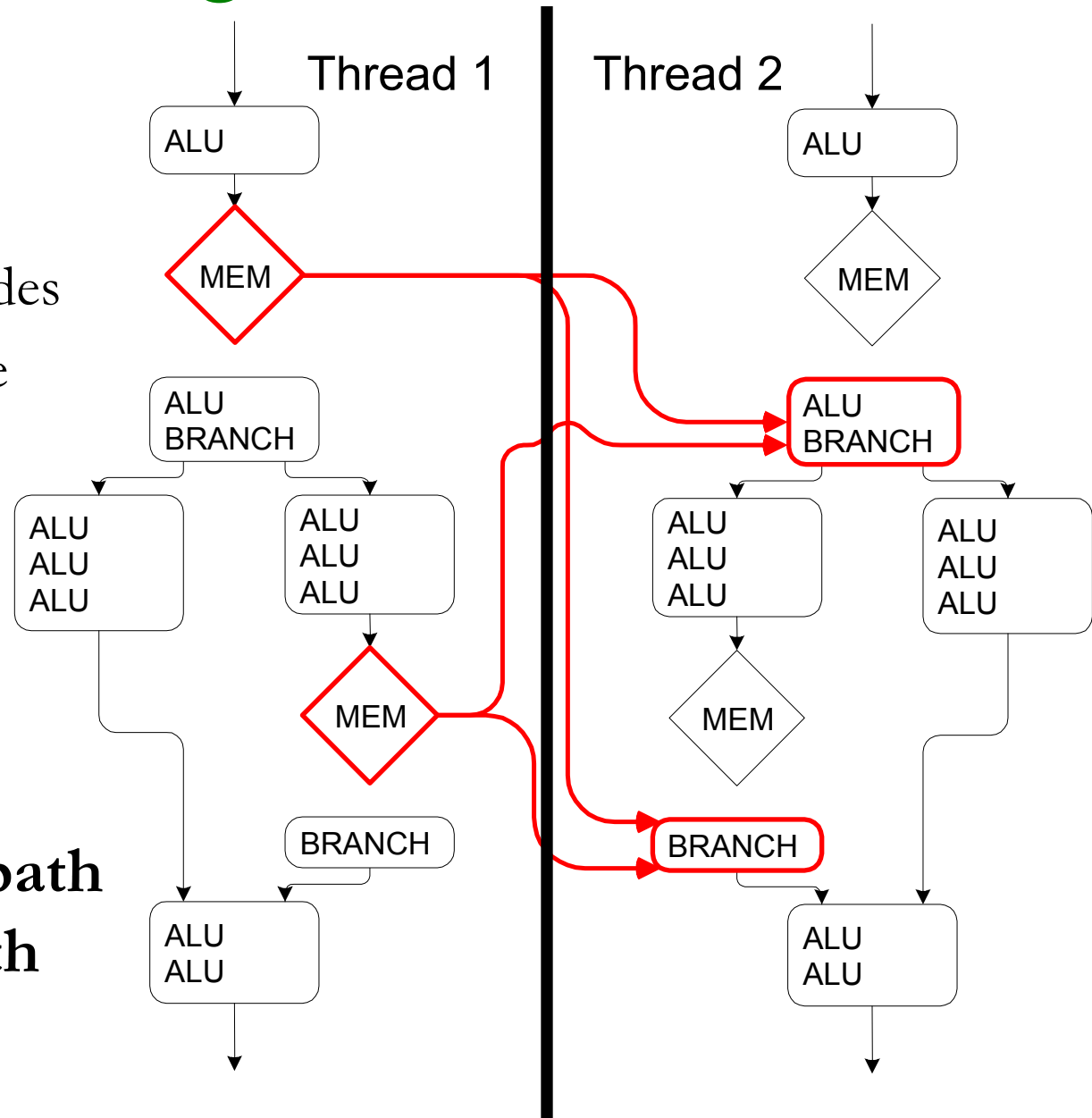
Yield Edge Placement

Yield Edges

- Begin at yield nodes
- End at yield node followers

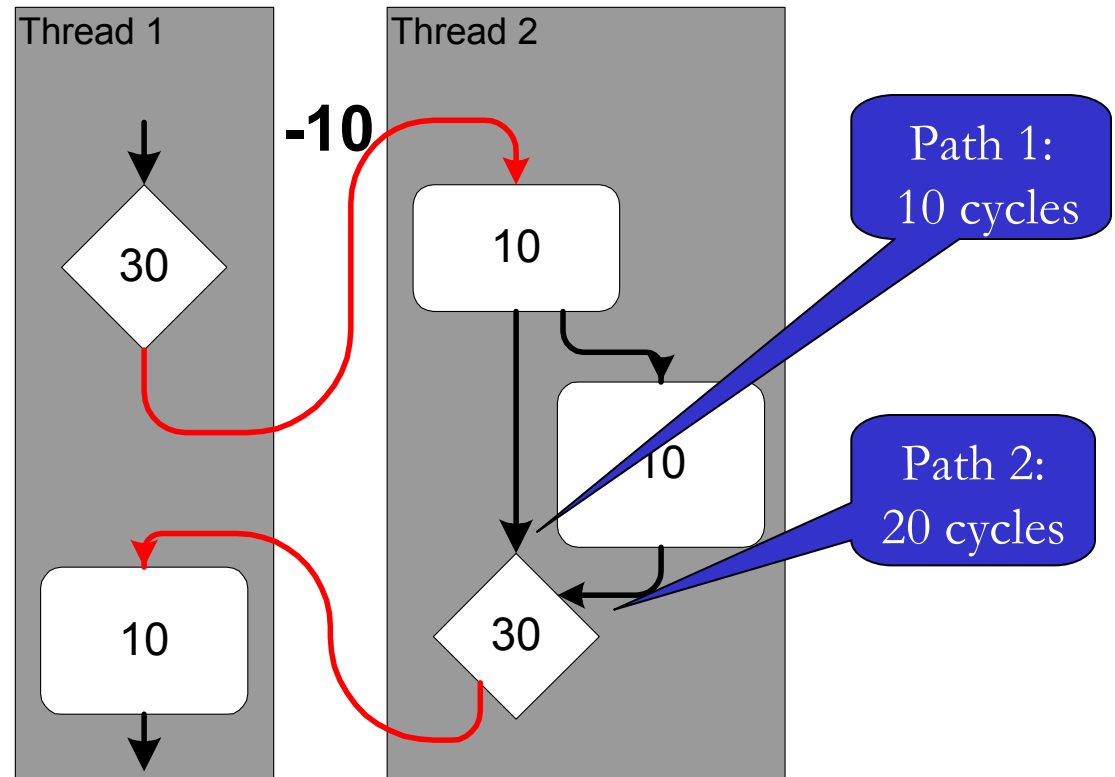
Return edges not shown

Most expensive path now includes both threads!



Costs & Yield Edge Values

- Yield edge values
 - Account for hidden latency
 - Min path to next yield



$$\text{Yield Edge Value} = -\min(\text{Min Dest Exec Cycles}, \text{Source Yield Latency})$$

New Problem Statement

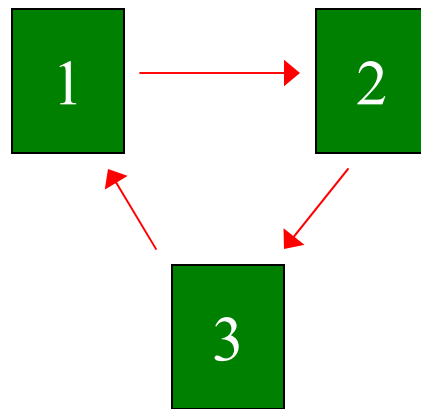
- New objective function:

$$\text{Program Execution Time} = \sum_{i=1}^{N'} c_i x_i + \sum_{j=1}^M k_j y_j$$

- Where,
 - k_j is the yield edge value for yield edge j
 - And y_j is its frequency

3 or More Threads

- Scales directly to more than 2 threads
- Yield edges are added only between threads that neighbor one another in the round-robin schedule
- Size complexity is quadratic, $O(m^2)$, where m is the number of yield operations per thread



Results for 4 Threads

Program	Simulated WCET	Estimated WCET
reverse_array	896	2204
chk_data	985	1808
sort	192789	817980

- Bounds nearly within a factor of 2
 - Except, again, for sort
- Pessimism due to:
 - No overlap between yield operations & apps are memory bound
 - Conservative yield edge values

Assignment

- Tuesday
 - Milestone 3: Implementation 2
 - Use the template in this presentation and post your milestone report to the newsgroup. Bring a hardcopy to class.
- Thursday
 - None