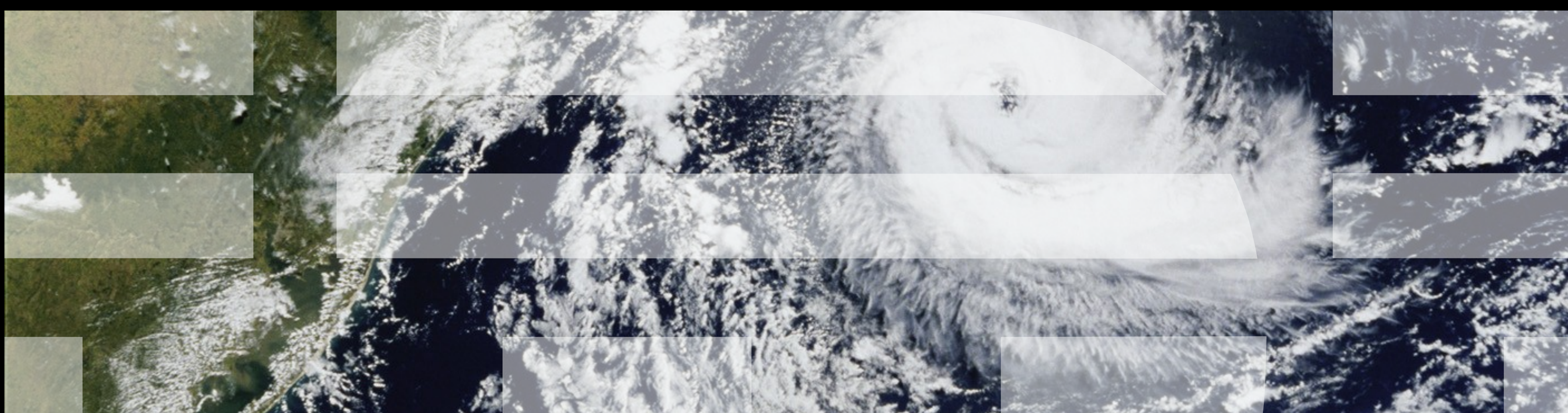
Paul E. McKenney, IBM Distinguished Engineer, CTO Linux

October 24, 2010

When Do Real Time Systems Need Multiple CPUs?

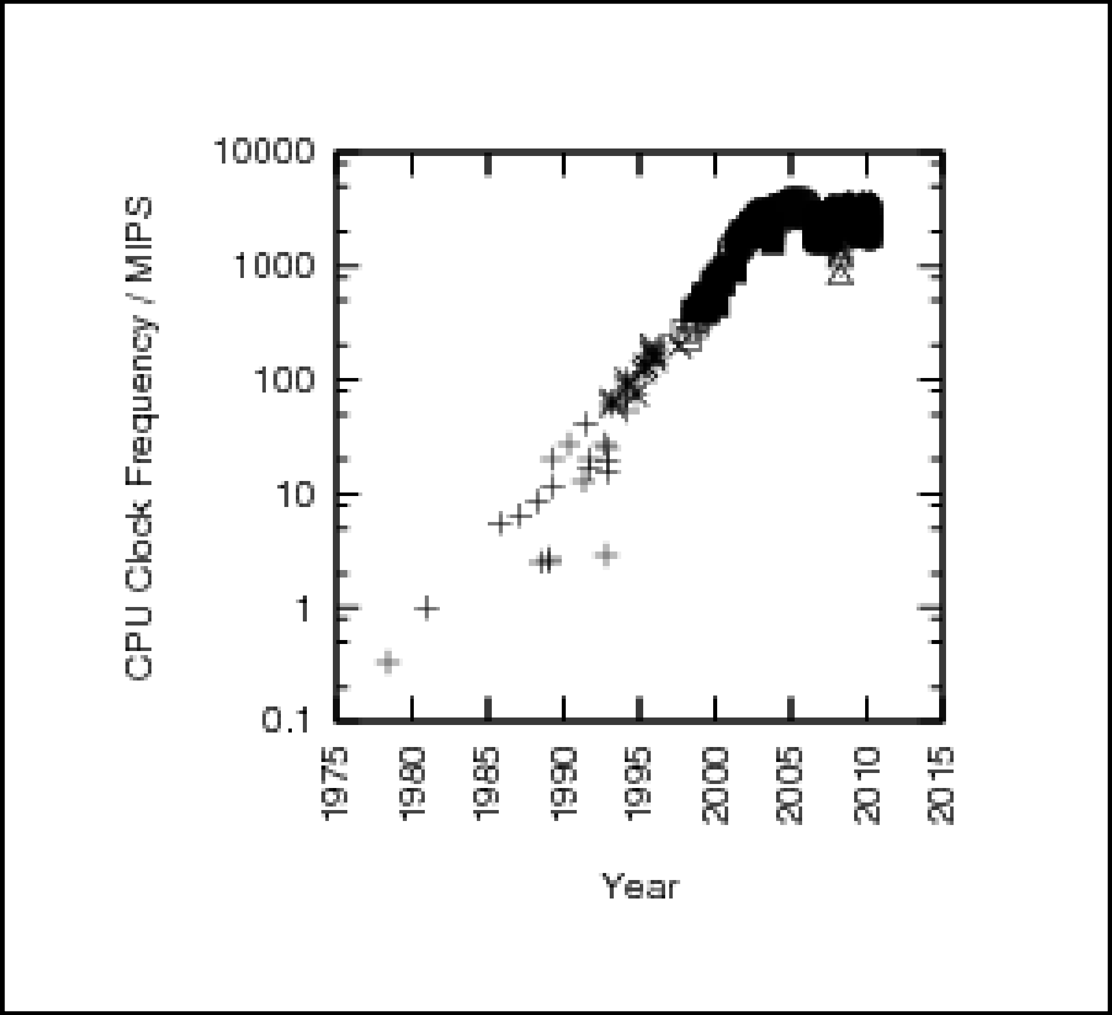


Overview

- SMP Real Time Systems: Inevitable?
- Very Brief Overview of Parallelization
- Two Basic Modes of Control-Loop Parallelism
- Evaluation
- “Real Time Theory Depression” and How to Fight It
- What to do with Leftover CPUs?
- Summary

SMP Real Time Systems: Inevitable?

SMP Inevitability: The Party Line



Real-World Evidence for SMP Inevitability...

- Multi-core ARM CPUs: a few tens of dollars per chip
- SMP support in -rt patchset for the Linux kernel
- SMP real-time systems in use, including financial military applications

More Real-World Evidence for SMP Inevitability...

- Multi-core ARM CPUs: a few tens of dollars per chip
- SMP support in -rt patchset for the Linux kernel
- SMP real-time systems in use, including financial military applications

- But is SMP real time the right answer in all cases?

SMP Real Time Systems: The Case Against

- Most software (especially real-time software is still single-threaded
- Many algorithms and workloads lack high-quality parallel implementations
- Parallel implementations often larger and more complex than their single-threaded counterparts
- Parallel implementations more difficult to validate than their single-threaded counterparts
- RT theory still tied to uniprocessor models and algorithms

- Parallel hardware is here. Parallel software? Not so much...
- Need a reason for RT parallelism: default answer is single-threaded

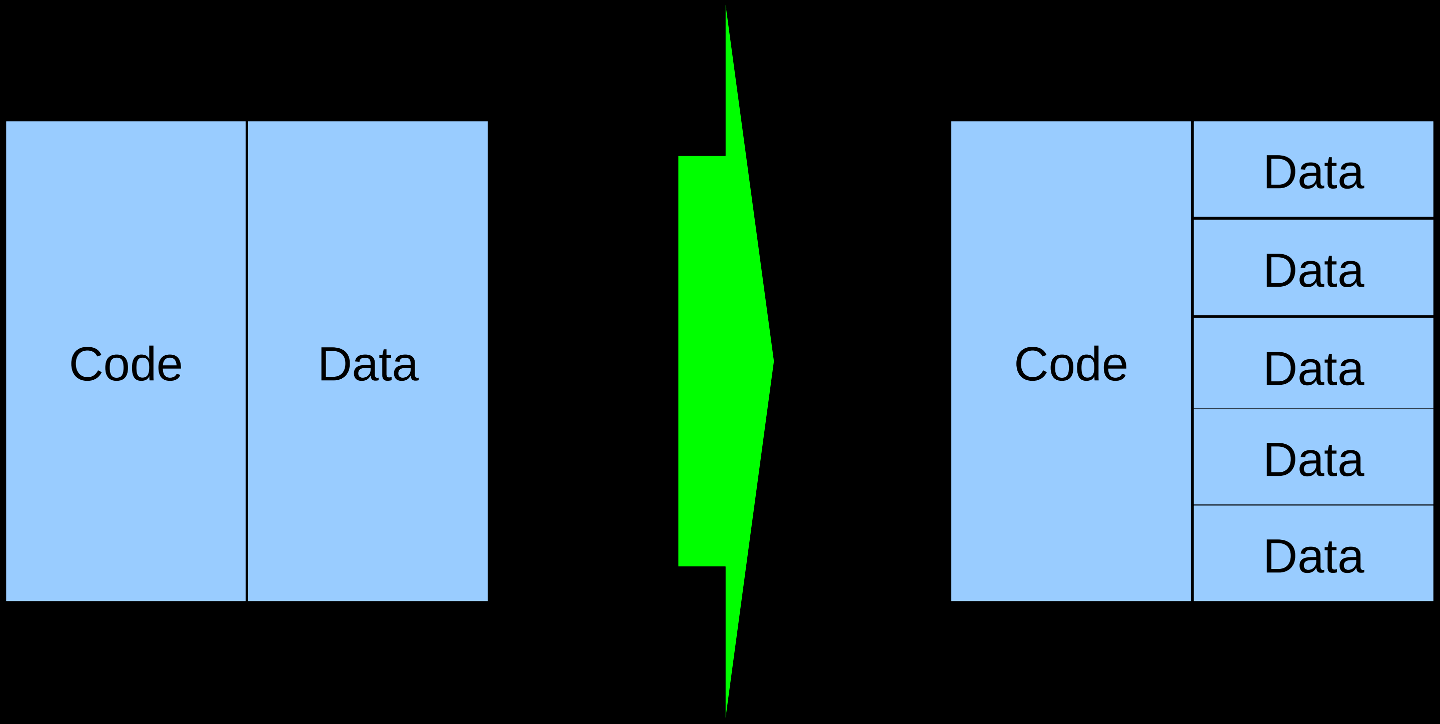
SMP Real Time Systems: The Case Against

- Most software (especially real-time software is still single-threaded
- Many algorithms and workloads lack high-quality parallel implementations
- Parallel implementations often larger and more complex than their single-threaded counterparts
- Parallel implementations more difficult to validate than their single-threaded counterparts
- RT theory still tied to uniprocessor models and algorithms

- Parallel hardware is here. Parallel software? Not so much...
- Need a reason for RT parallelism: default answer is single-threaded
- Blindly replicating UP RT in an SMP environment: not a winning strategy!

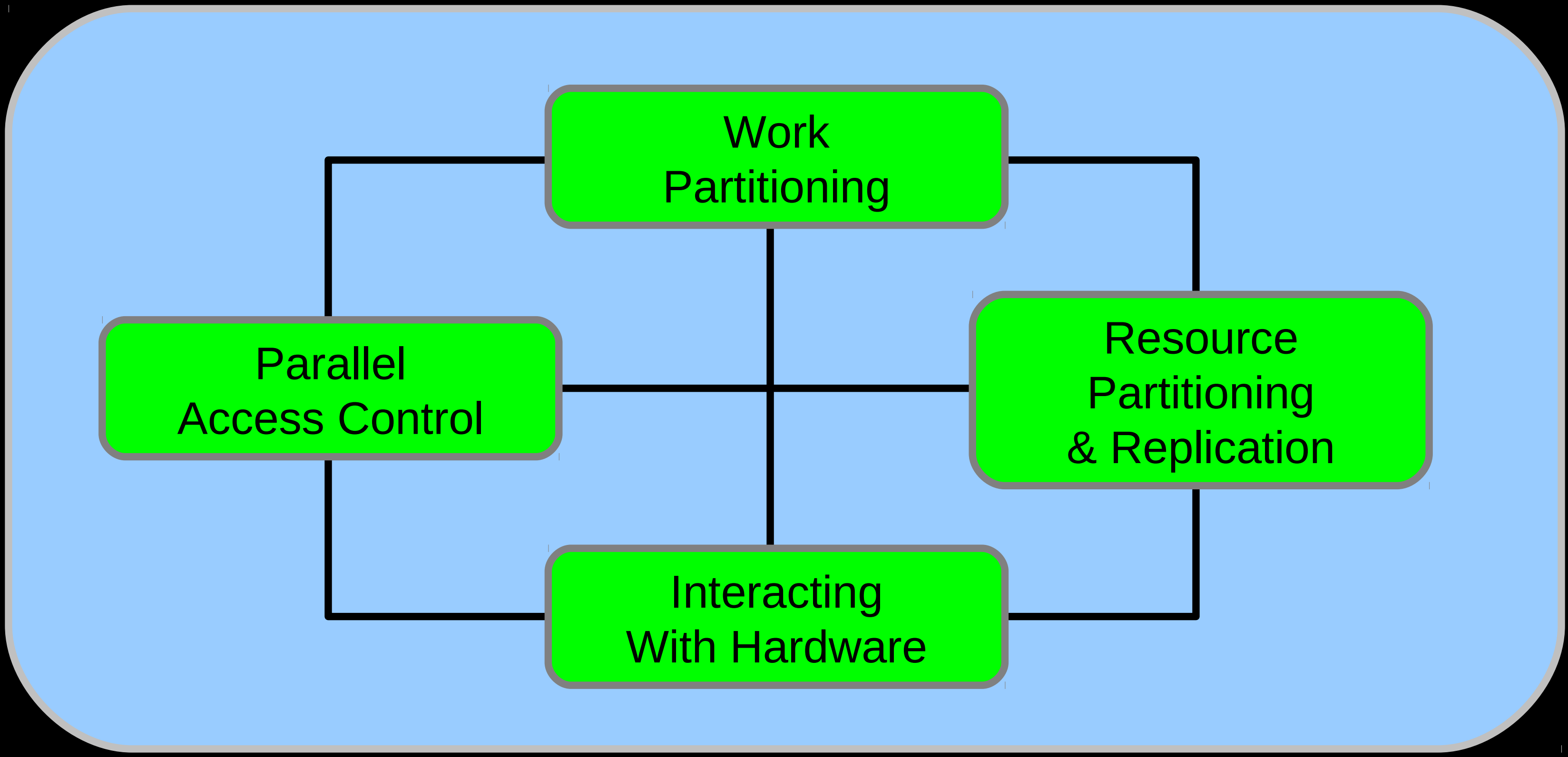
Very Brief Overview of Parallelization

Parallelization: First, Partition the Data!



Just a quick overview: there are full textbooks on this topic, for example:
http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html

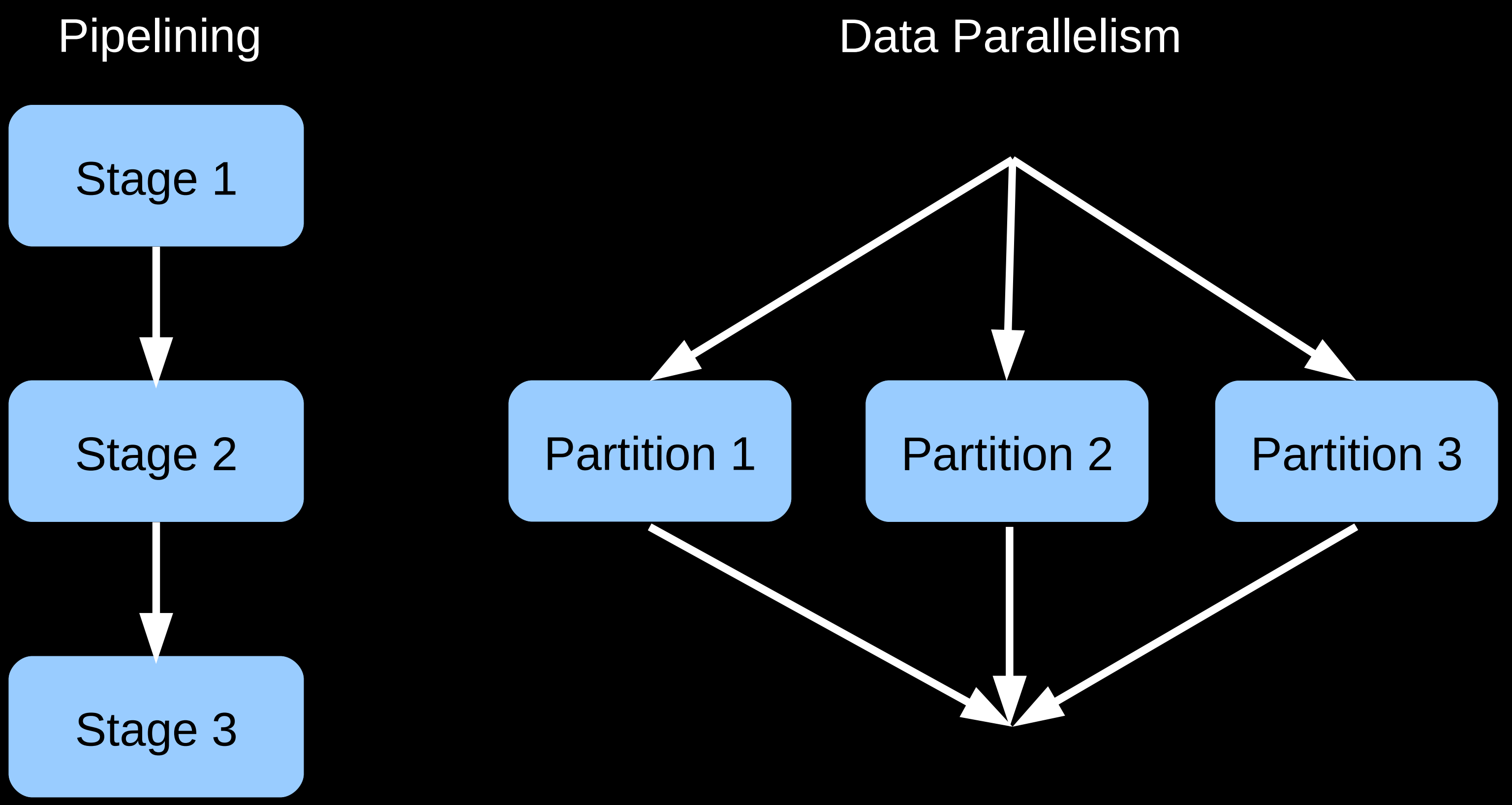
Parallelization: General Process



Data-parallel approach: first partition resources, then partition work, and only then worry about parallel access control. Lather, rinse, and repeat.

Two Basic Modes of Control-Loop Parallelism

Two Basic Modes of Parallelism



Which to use? And when?

Evaluation

Test With Randomly Chosen Synthetic Workload

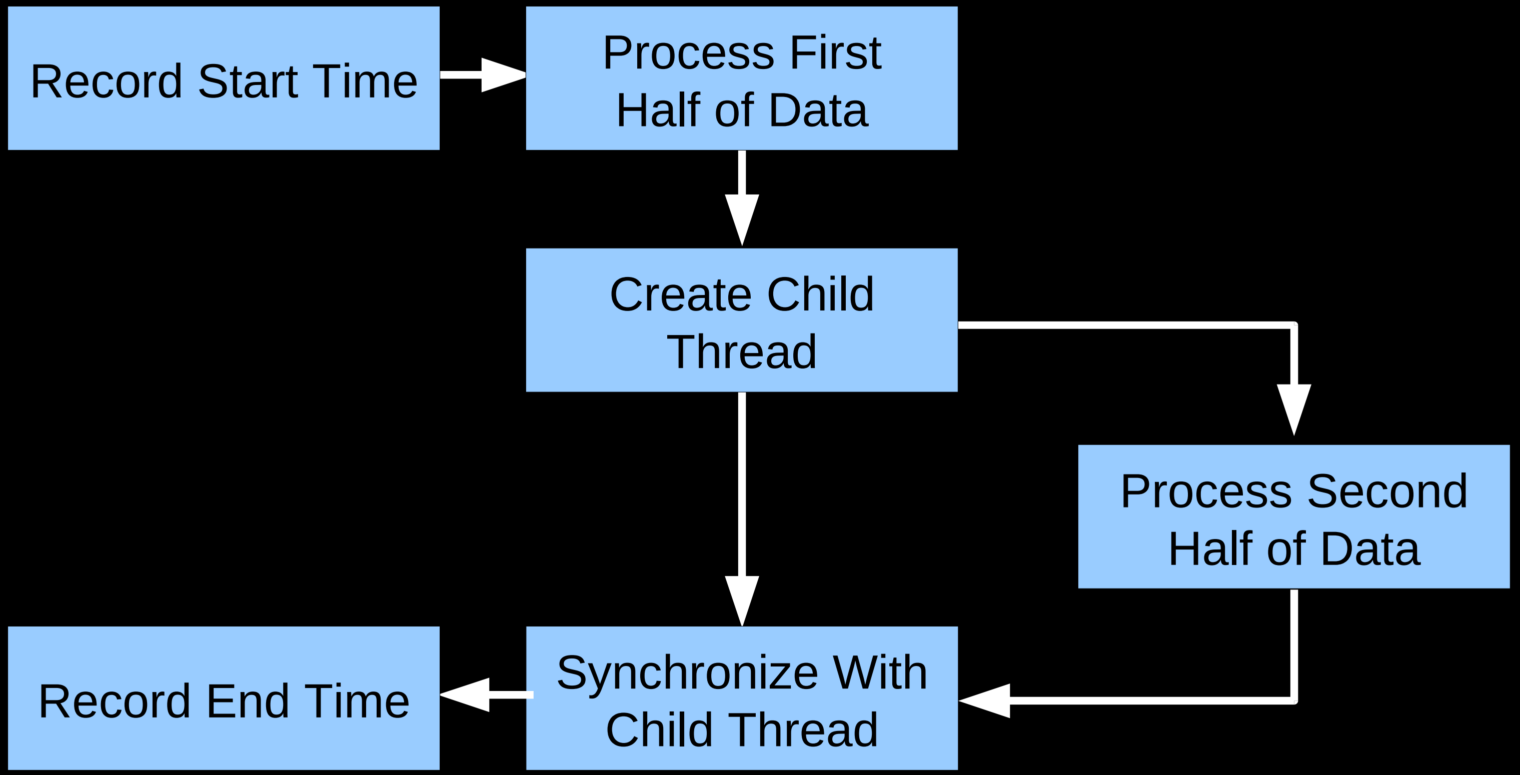
```
void mung(int *x, int n)
{
        int i;

        for (i = 0; i < n; i++)
                x[i] = 10 + x[i] / 10;
}
```

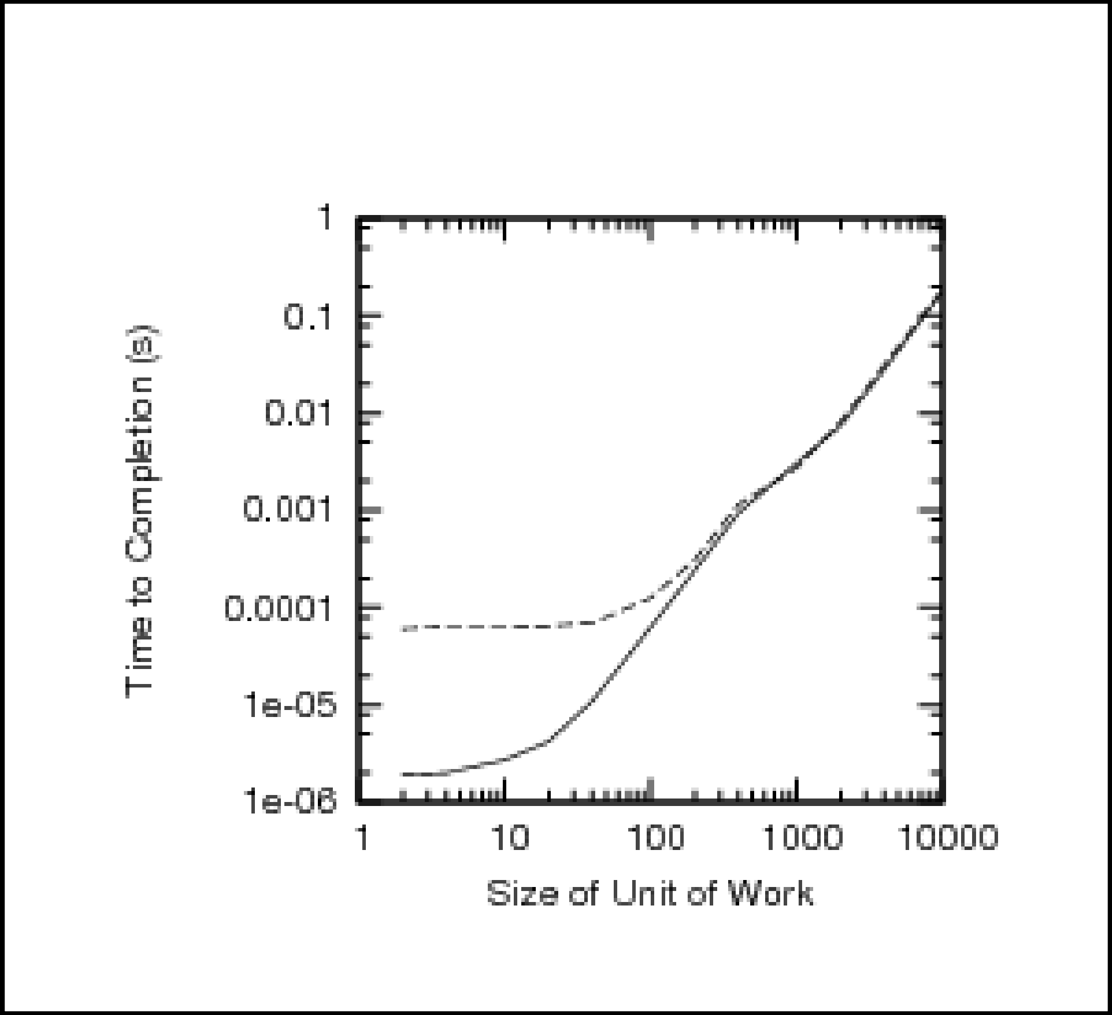
Pipelining Test Setup

- User-mode tests
- Synchronization via pthread_mutex_t
- Overhead of pthread_create() and pthread_join() counted against pipelining
- Flow of control:
  - Record start time
  - Process the first half of the data
  - Create a child thread using pthread_create()
  - Child processes second half of the data
  - Use pthread_join() to synchronize with child thread
  - Record end time

Pipelining Parallel Control Flow



Latency Results for Pipelining: Not Good!!!

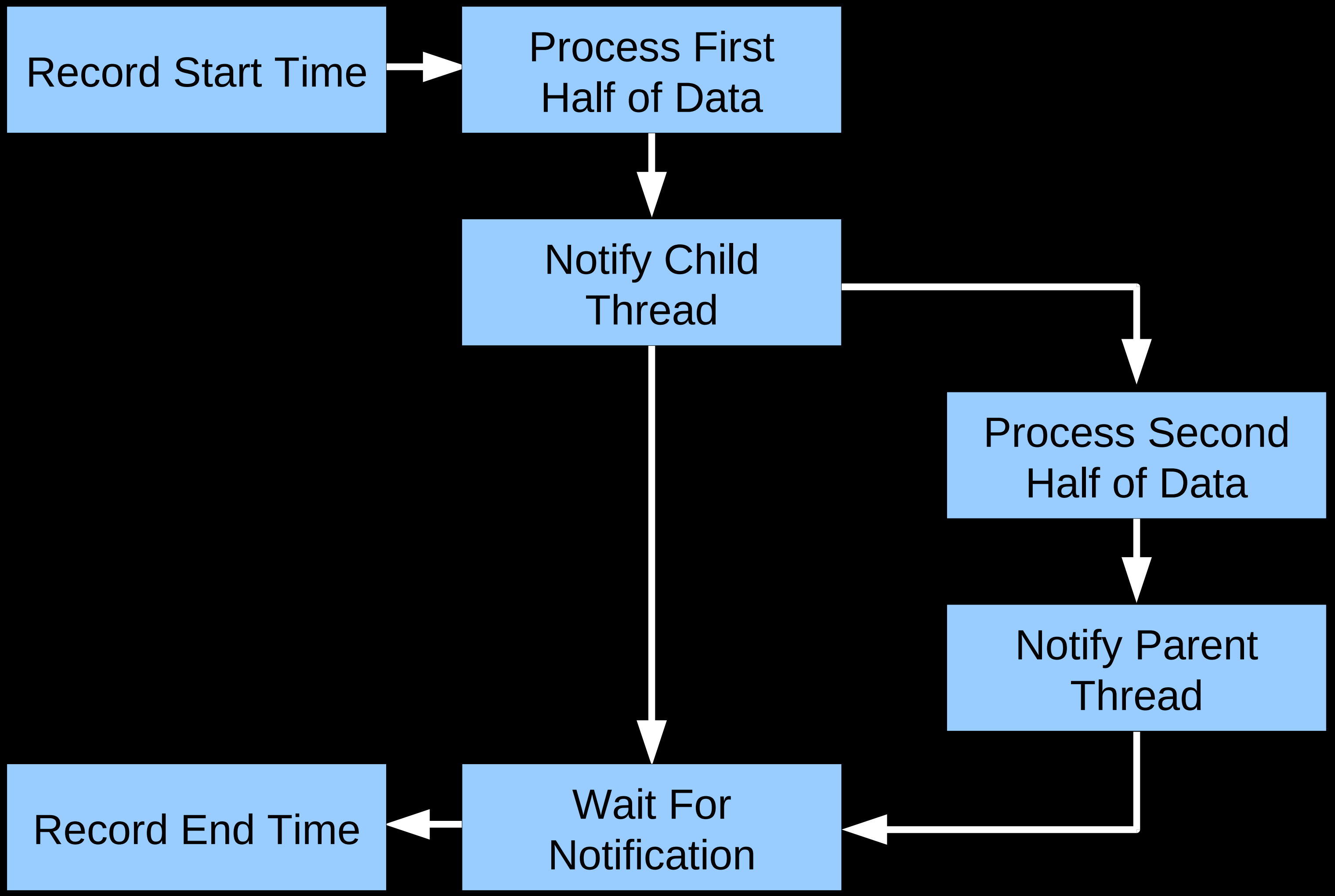


Always Faster To Run a Single Thread!!!

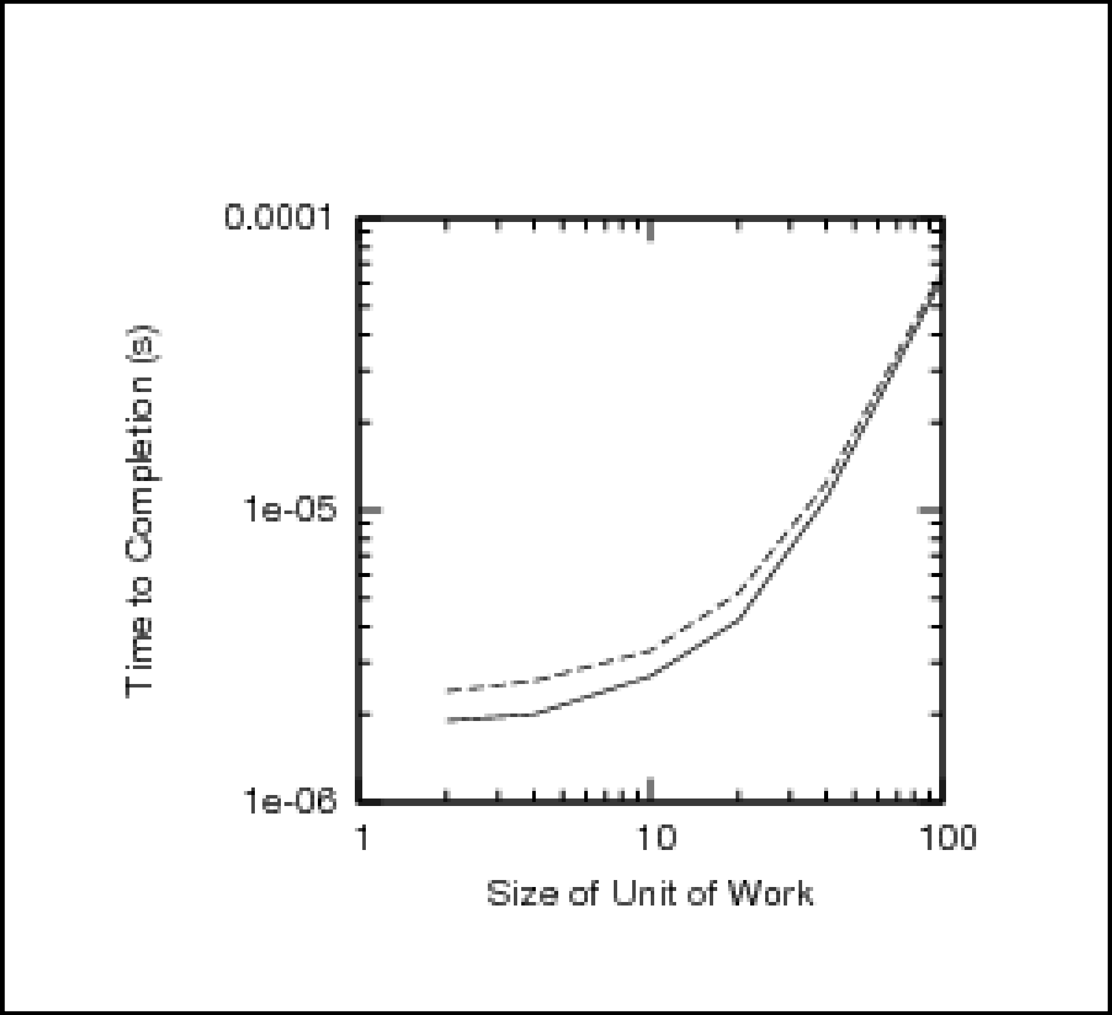
Pipelining Test Setup: Pre-Existing Threads

- User-mode tests
- Synchronization via pthread_mutex_t
- Create threads at initialization:
  - Overhead of pthread_create() and pthread_join() *not* counted against pipelining
- Lock threads down to specific CPUs
- Downstream thread spins waiting for work from upstream thread

Pipelining Parallel Control Flow: Pre-existing Threads



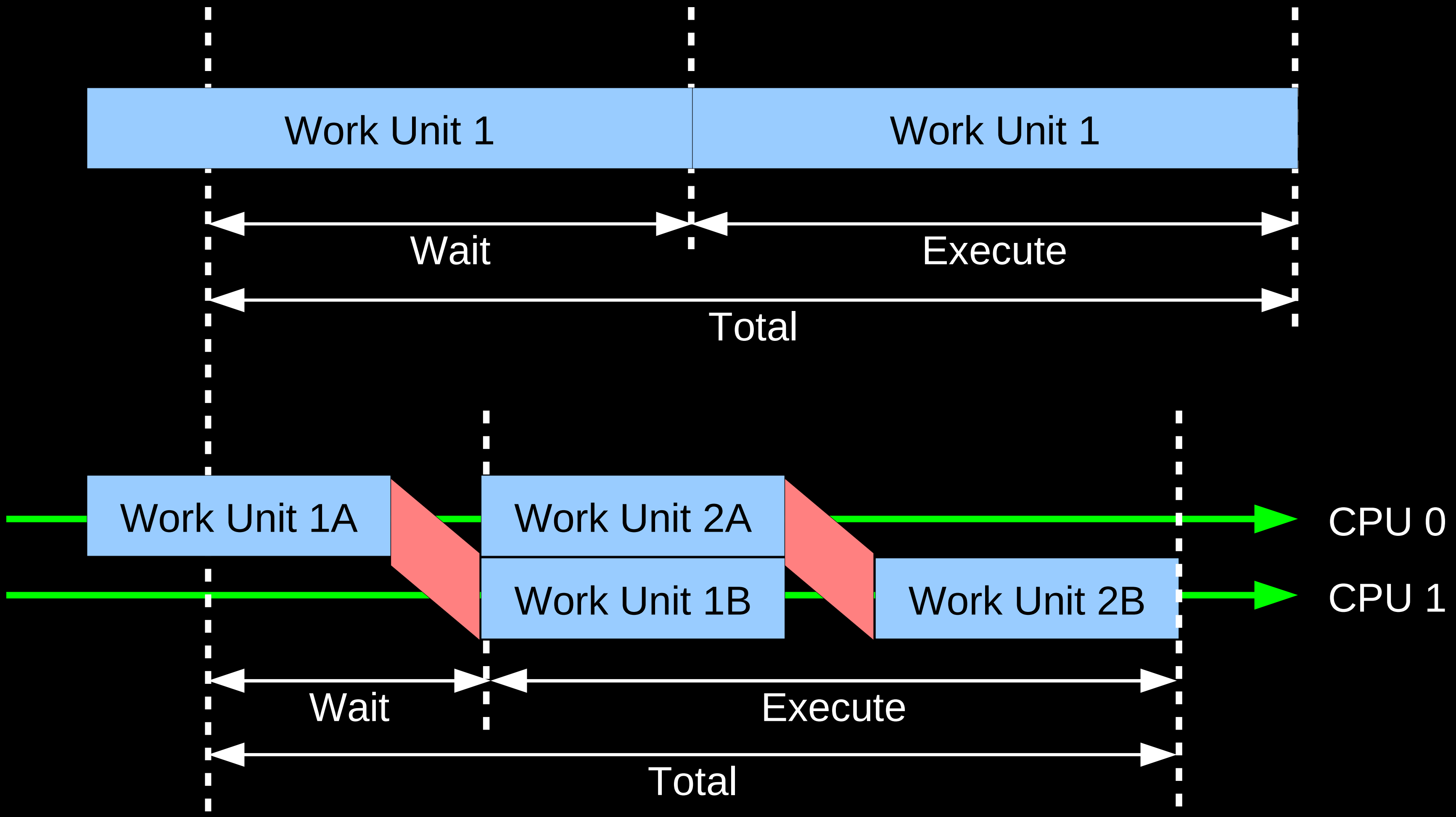
Latency Results for Pipelining With Pre-Existing Threads...



Well, it isn't quite as bad as before, but...

Why Bother With Parallel Pipelines???

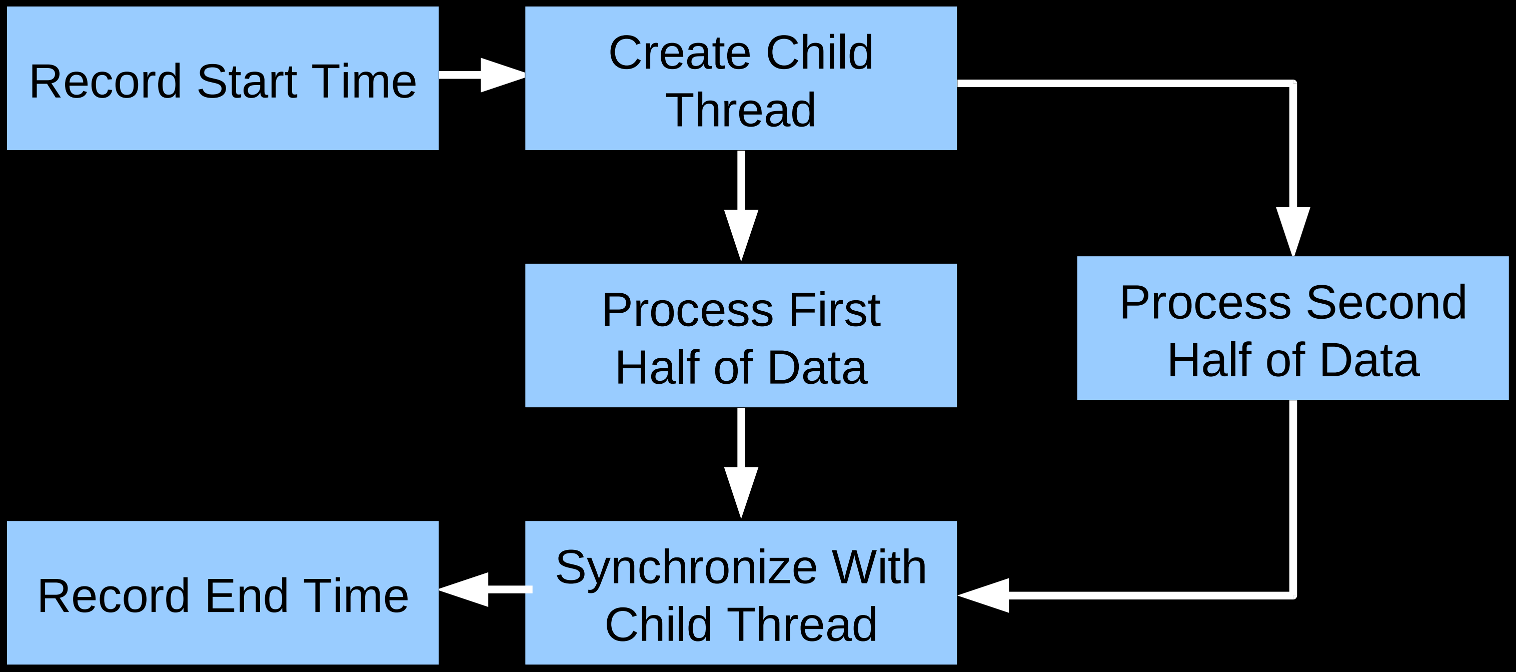
Good Use of Parallel Pipelines: Overlap Successive Work Units



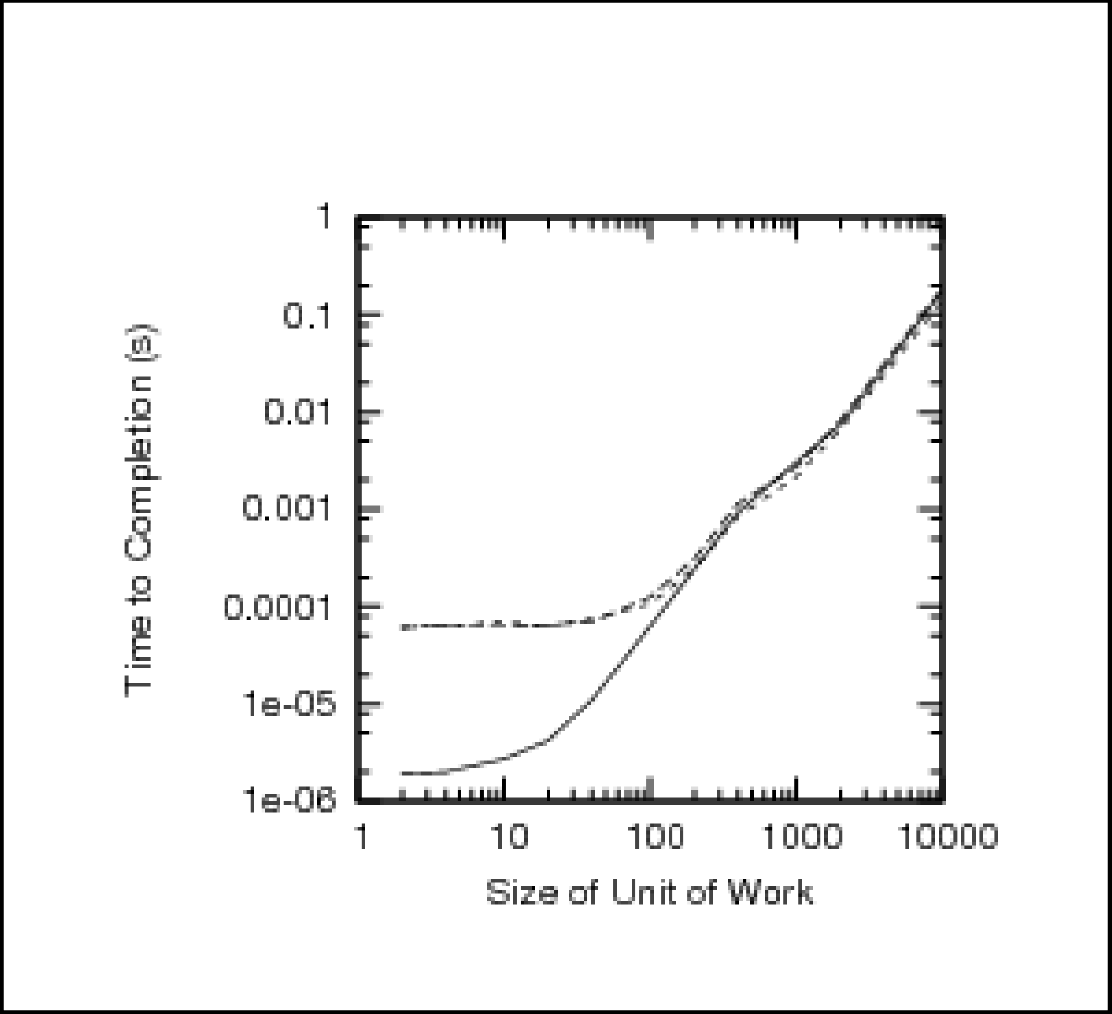
Data Parallel Test Setup

- User-mode tests
- Synchronization via pthread_mutex_t
- Overhead of pthread_create() and pthread_join() counted against pipelining

Data Parallel Control Flow



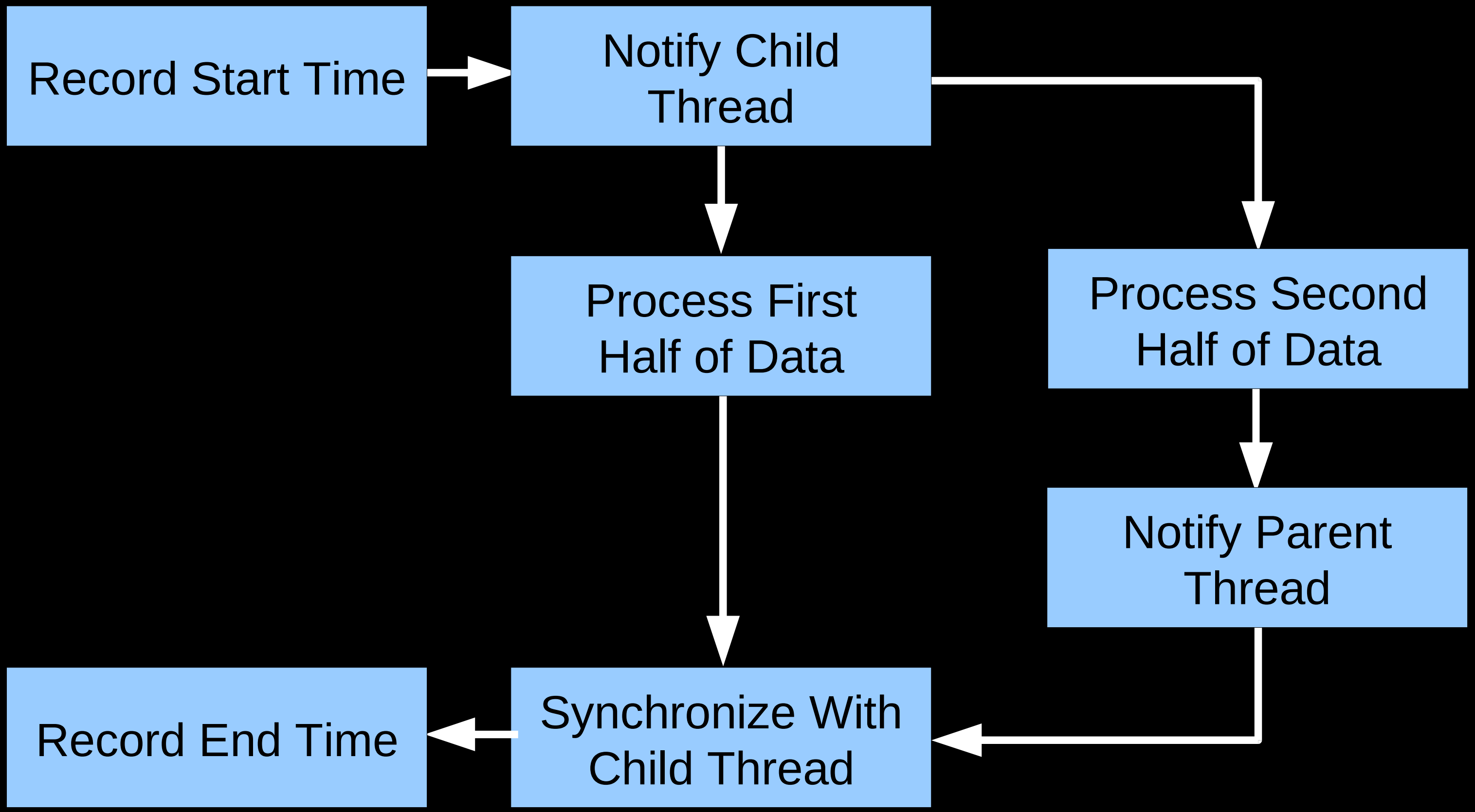
Latency Results for Data Parallelism: Not Great, But OK...



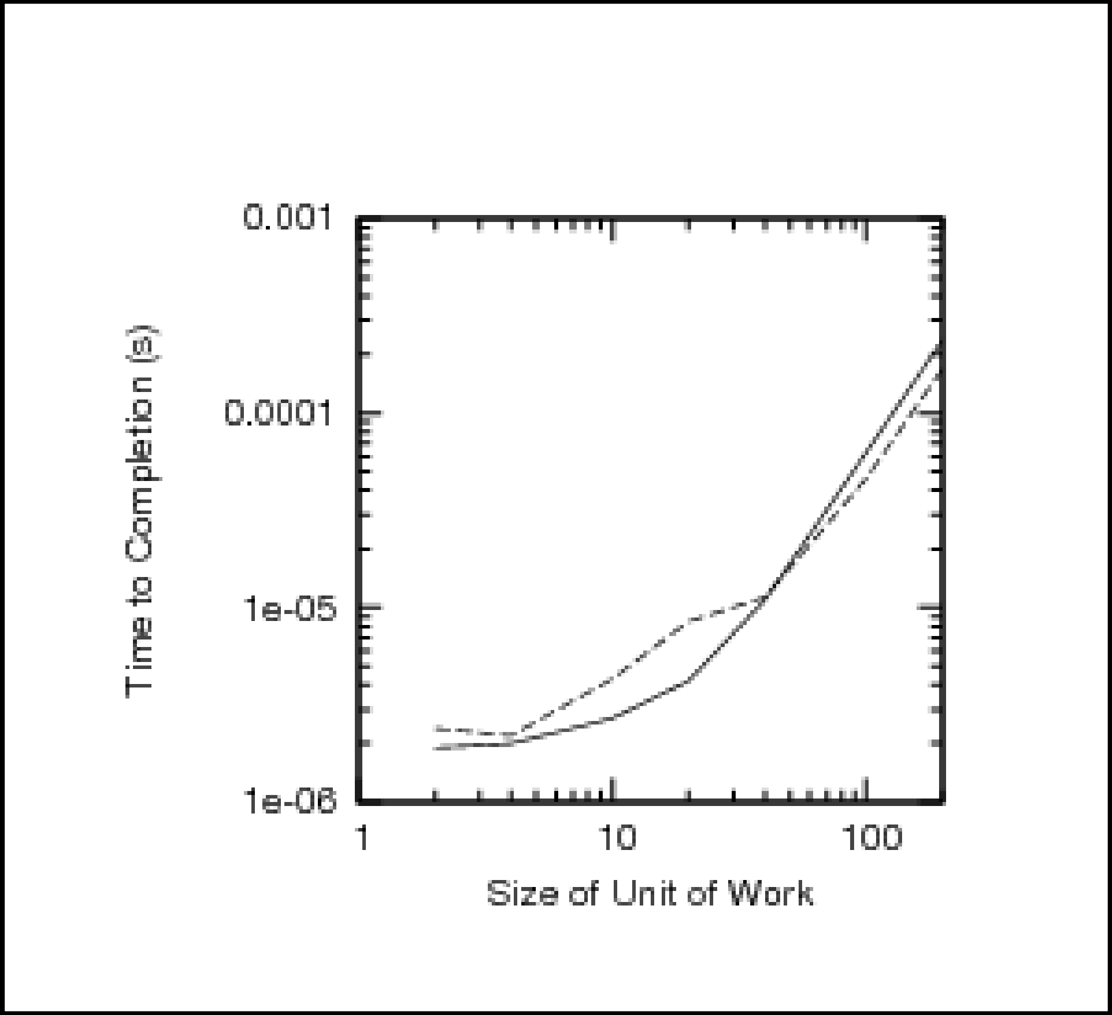
Data Parallel Test Setup: Pre-Existing Threads

- User-mode tests
- Synchronization via pthread_mutex_t
- Create threads at initialization:
  - Overhead of pthread_create() and pthread_join() *not* counted against pipelining
- Lock threads down to specific CPUs
- Downstream thread spins waiting for work from upstream thread

Data Parallel Control Flow: Pre-Existing Threads



Latency Results for Pipelining With Pre-Existing Threads...



Semi-respectable speedup! What can be achieved?

“Real Time Theory Depression” and How to Fight It

When In Doubt, Normalize!!!

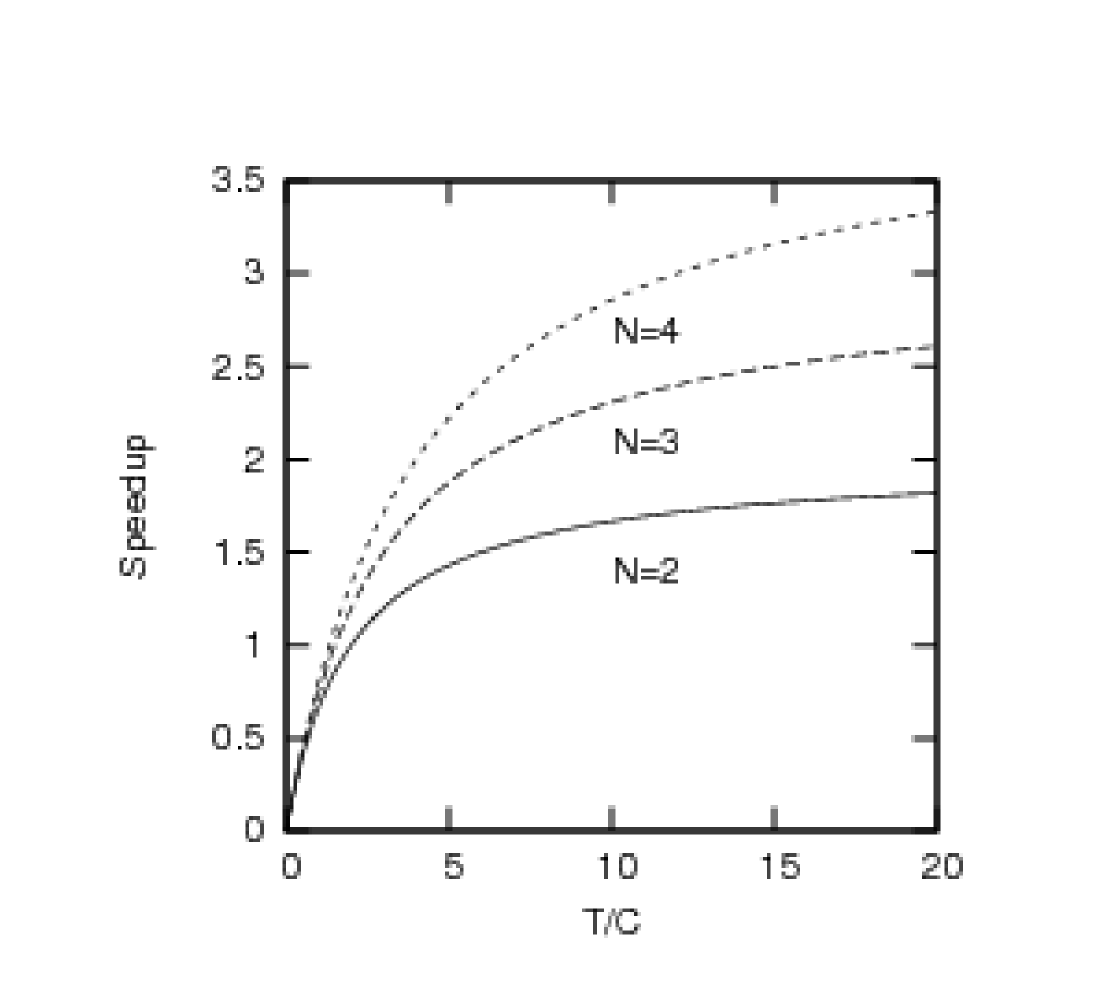
- T: Time required to complete unit of work in single-threaded environment
- C: Communications overhead (of all kinds) incurred in SMP environment
- N: Number of CPUs/threads
- S: Speedup: sequential time divided by SMP time (yes, can be less than 1!)

$$S = \frac{T}{\frac{T}{N} + C}$$

- Plot S against T/C...

Theoretical Limits For Data Parallelism

$$S = \frac{N \frac{T}{C}}{\frac{T}{C} + N}$$



Murphy Strikes Again!!! (And CS Theory is Depressing!)

Suppose That You Need a Specific Speedup

- Solve prior expression for T/C:

$$\frac{T}{C}=S\frac{N}{N-S}$$

- Plug in values for S & N:
  - 40% speedup (S=1.4)
    - N=2: T/C>=4.7
    - N=3: T/C>=2.6
    - N=4: T/C>=2.2
  - 100% speedup (S=2.0)
    - N=2: T/C infinite
    - N=3: T/C>=6
    - N=4: T/C>=4
  - 200% speedup (S=3.0)
    - N=3: T/C infinite
    - N=4: T/C>=12
- The tighter your RT deadlines, the less helpful parallelism will be!!!

How Can You Fight Theoretical RT Parallel Depression???

- Apply parallelism at the highest possible level
  - The larger your units of work, the more benefit you will get from parallelization
- Use interleaving (crypto, compression, encoding)
  - Some difficulties applying to audio
  - Consider splitting the display for video: but too bad about existing standards...
- Ditch parallelism: hand-optimize sequential control loops
  - Real men will hand-code them in assembly
  - Real women will hand-code them in hexadecimal
- Ditch parallelism: hardware acceleration for standard transformations
- Ditch parallelism: FPGAs for non-standard transformations
  - Which won't necessarily be any easier than coding in parallel
  - But some workloads are better suited to FPGAs and vice versa

- And if the original sequential implementation was fast enough, why did you even bother reading this far???

How Can You Fight Theoretical RT Parallel Depression???

- Apply parallelism at the highest possible level
  – The larger your units of work, the more benefit you will get from parallelization
- Use interleaving (crypto, compression, encoding)
  – Some difficulties applying to audio
  – Consider splitting the display for video: but too bad about existing standards...
- Ditch parallelism: hand-optimize sequential control loops
  – Real men will hand-code them in assembly
  – Real women will hand-code them in hexadecimal
- Ditch parallelism: hardware acceleration for standard transformations
- Ditch parallelism: FPGAs for non-standard transformations
  – Which won't necessarily be any easier than coding in parallel
  – But some workloads are better suited to FPGAs and vice versa

- And if the original sequential implementation was fast enough, why did you even bother reading this far??? Ah yes, wasting those leftover CPUs...

How Can You Fight Theoretical RT Parallel Depression???

- Apply parallelism at the highest possible level
  – The larger your units of work, the more benefit you will get from parallelization
- Use interleaving (crypto, compression, encoding)
  – Some difficulties applying to audio
  – Consider splitting the display for video: but too bad about existing standards...
- Ditch parallelism: hand-optimize sequential control loops
  – Real men will hand-code them in assembly
  – Real women will hand-code them in hexadecimal
- Ditch parallelism: hardware acceleration for standard transformations
- Ditch parallelism: FPGAs for non-standard transformations
  – Which won't necessarily be any easier than coding in parallel
  – But some workloads are better suited to FPGAs and vice versa

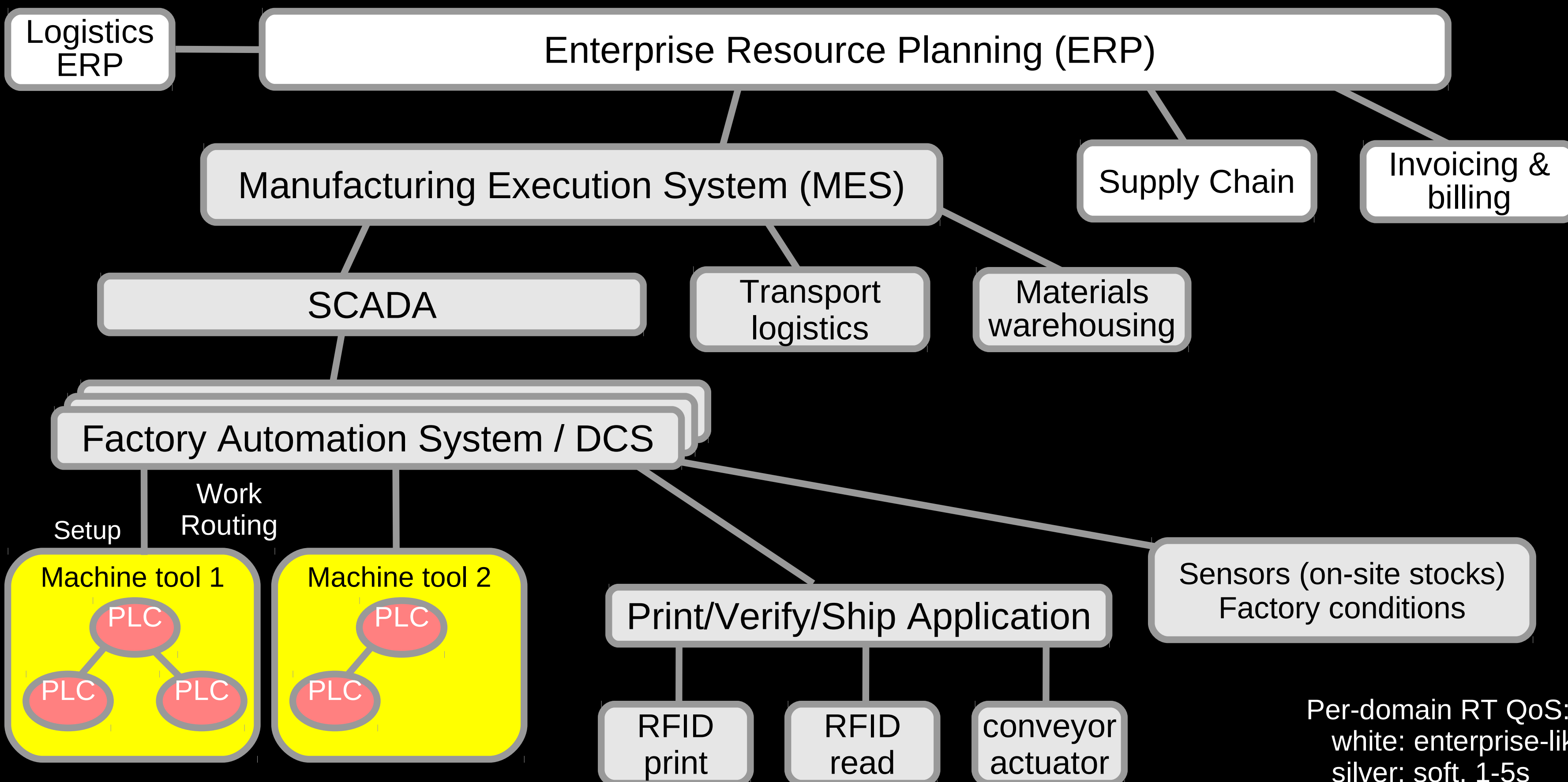
- And if the original sequential implementation was fast enough, why did you even bother reading this far??? Ah yes, wasting those leftover CPUs... Such a tragedy!!!

What to do with Leftover CPUs?

What To Do With Leftover CPUs???

- Get a system with fewer CPUs
- Power off the leftover CPUs
- Use leftover CPUs to run any needed UI or reporting
- For enterprise real time, run part of the enterprise portion of the application on the leftover CPUs
- These last two imply RT-to-non-RT communication...

Enterprise Real Time: RT Reflexes and Enterprise Processing



SCADA: supervisory/system control and data acquisition

Per-domain RT QoS:
- white: enterprise-like
- silver: soft, 1-5s
- gold: harder, <1s
- red: hard, sub-reflex

How To Do RT-To-Non-RT Communication???

- Messaging:
  - –Real-time implementations of linked queues
  - –User-mode equivalents of kfifo ring buffer
  - –Simple shared-memory “mailboxes”
  - –Numerous real-time messaging projects and products
- Lookups (read-mostly hash tables, lists, search trees):
  - –RCU!!!
- Other communications might use locking
  - –And you might want priority boosting...

Thread Placement Can Be Critical!!!

16-CPU 2.8GHz Intel X5550 (Nehalem) System

Operation	Cost (ns)	Ratio
Clock period	0.4	1
“Best-case” CAS	12.2	33.8
Best-case lock	25.6	71.2
Single cache miss	12.9	35.8
CAS cache miss	7.0	19.4
Single cache miss **(off-core)**	31.2	86.6
CAS cache miss **(off-core)**	31.2	86.5
Single cache miss **(off-socket)**	92.4	256.7
CAS cache miss **(off-socket)**	95.9	266.4

Summary

- SMP hardware is here – SMP software, not so much
- SMP for real time can make sense In control loops
  - Pipelining: reduce queuing delays
  - Data parallelism: reduce execution delays
  - However, the most aggressive control loop deadlines are hurt most by SMP communications overhead...
- Leftover CPUs have many uses
  - But don't be afraid to simply refuse to use them
- Thread placement is critically important
  - Something about the finite speed of light and atomic nature of matter – and lack of theory of SMP real time!

Questions?