Towards Automatic Resource Management in Parallel Architectures

Per Stenström

Chalmers University of Technology
Sweden

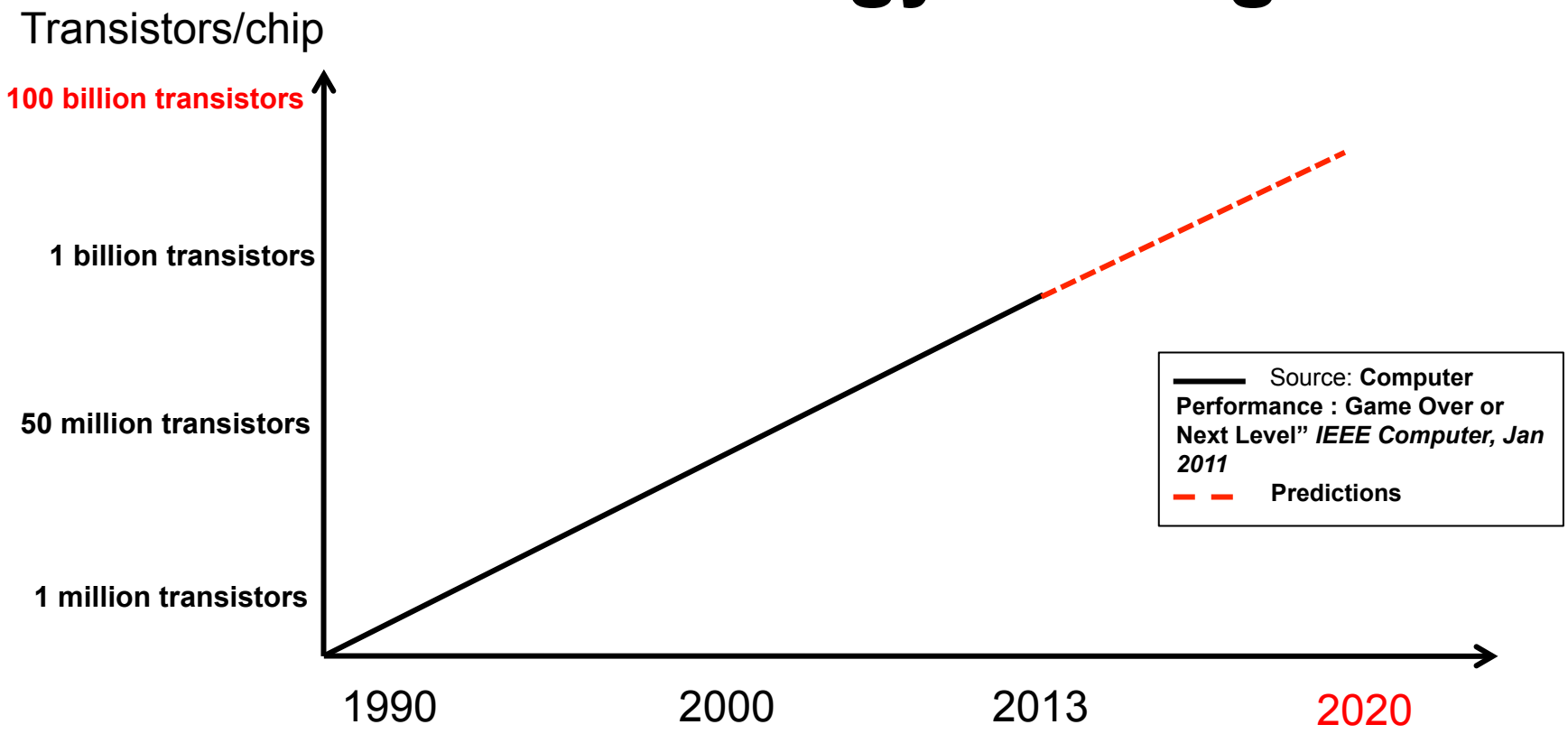
CHALMERS
Chalmers University of Technology

Keynote at PACT 2013 in Edinburgh, U.K., September 11, 2013.

Agenda

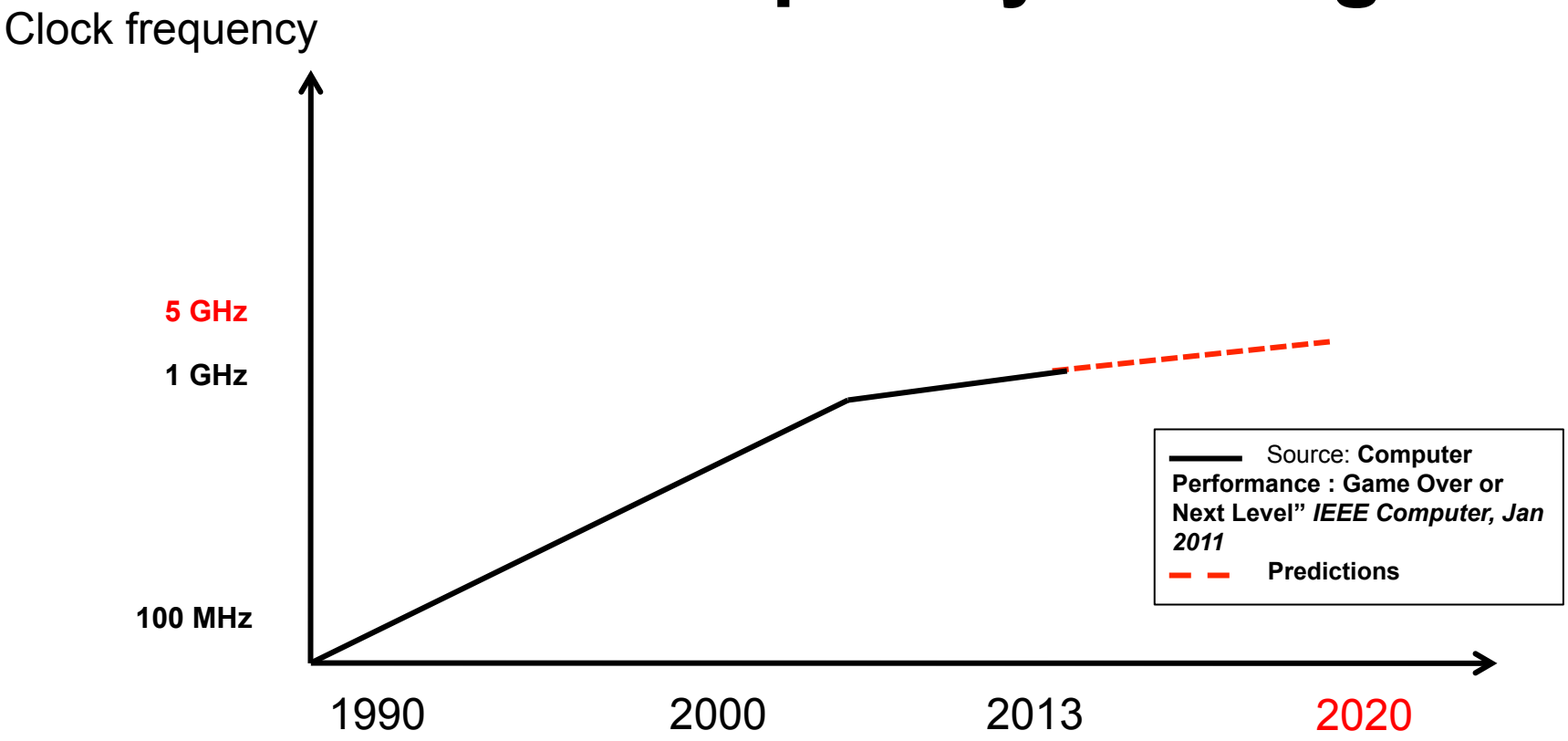
- Trends in parallel architectures
- Parallelism management (prog. model->arch.)
- Power management (prog. model->arch.)
- Value locality and cache management
- Concluding remarks

Technology Scaling



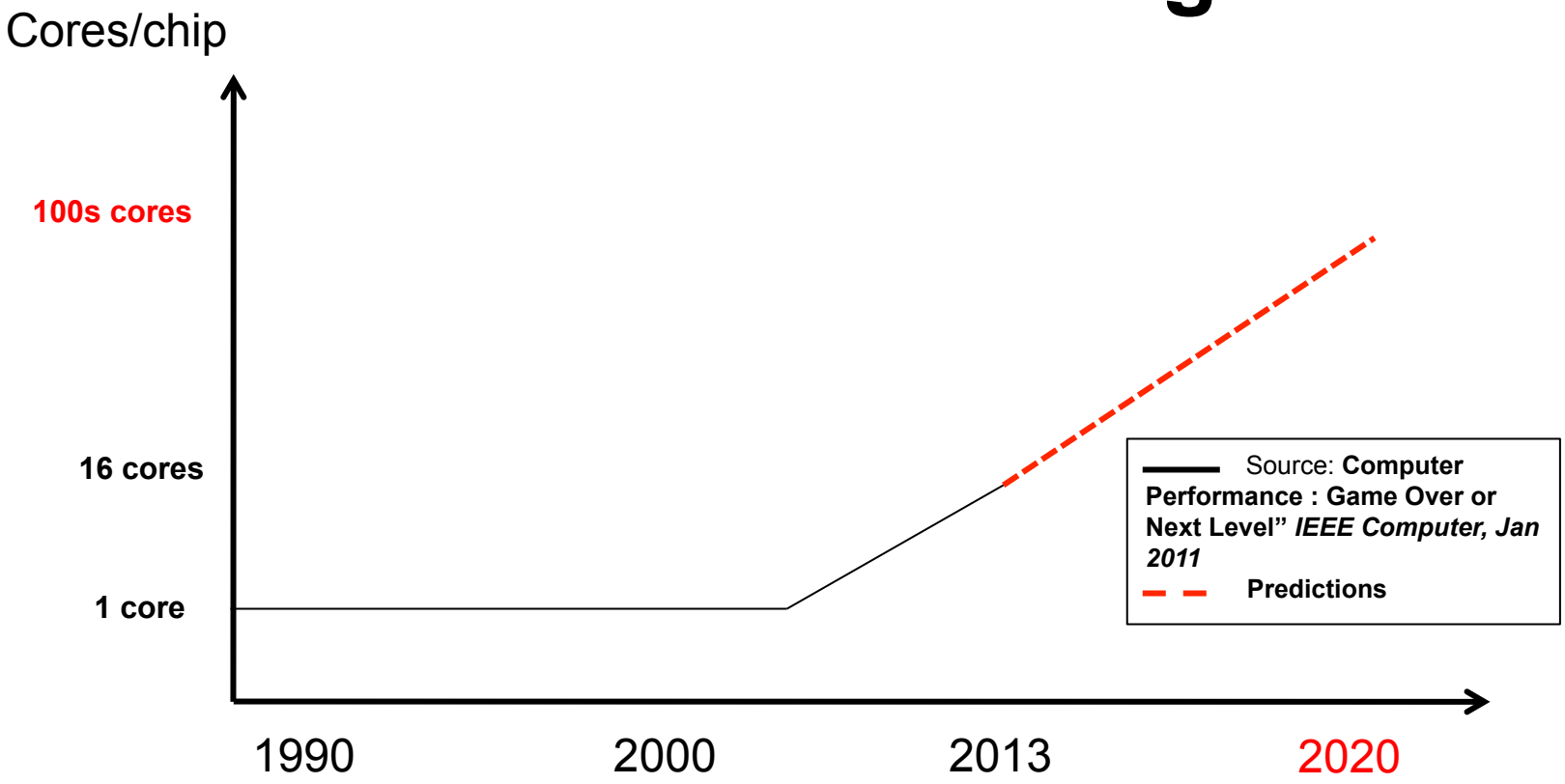
- Good news: Technology scaling will continue (for a while)

Clock Frequency Scaling



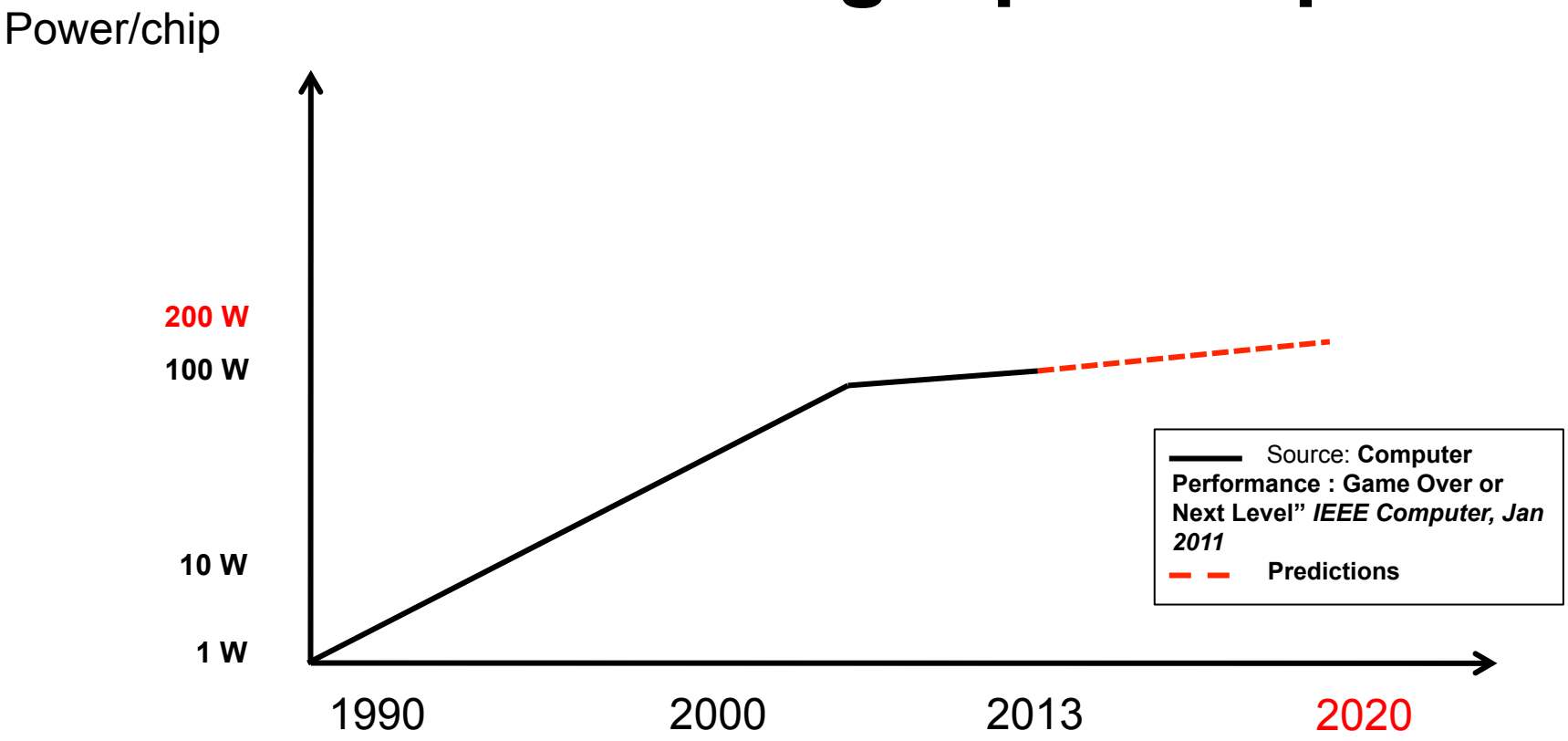
Bad news: Clock frequency will increase slowly at best

Multicore Scaling



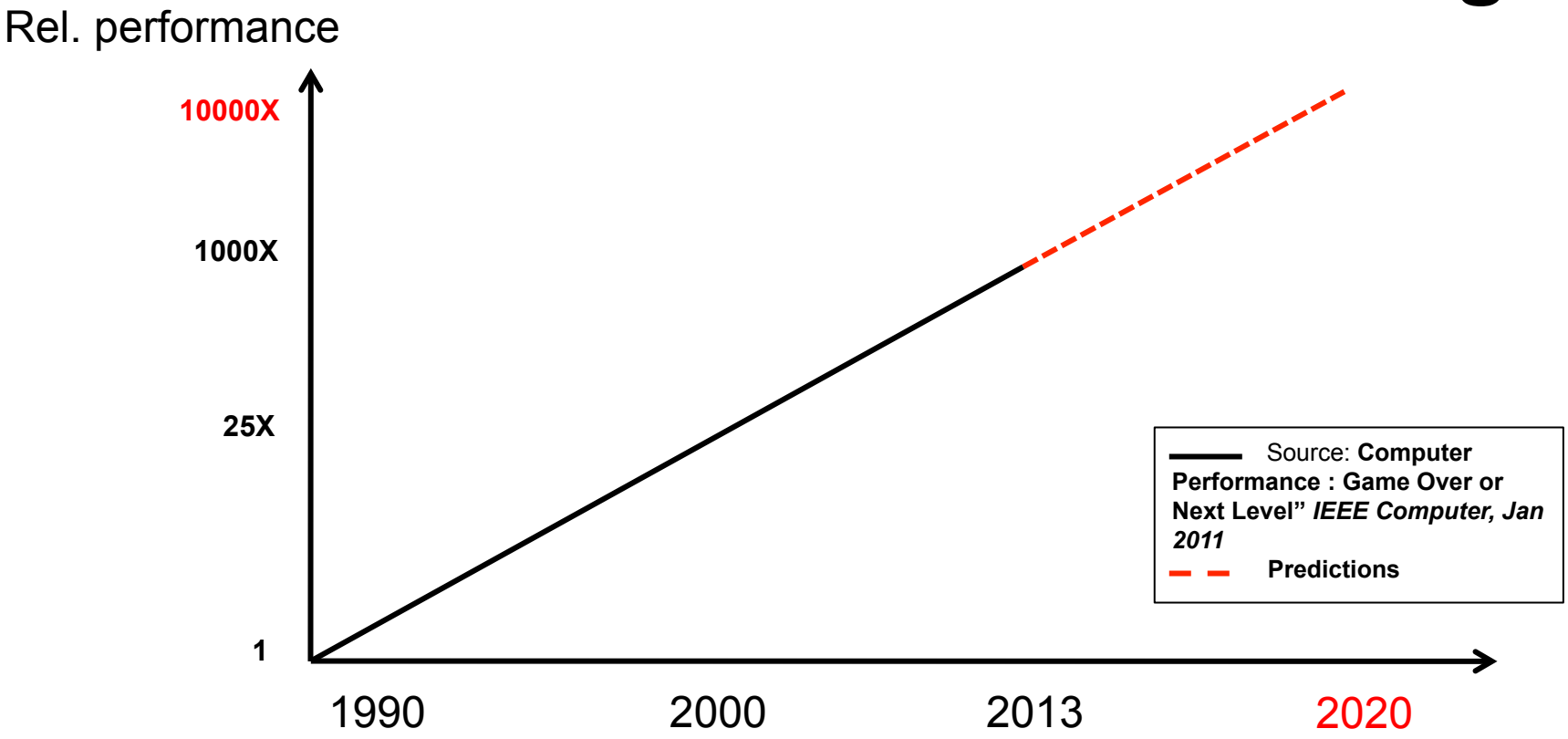
By 2020, several hundreds of powerful cores/chip

Power Budget per Chip



- Bad news: Power budget will increase slowly at best
- **Power budget**: <1W/core!

Performance/Mem. BW Scaling



Bad news: Off-chip memory bandwidth must scale linearly with performance

Trends (summary)

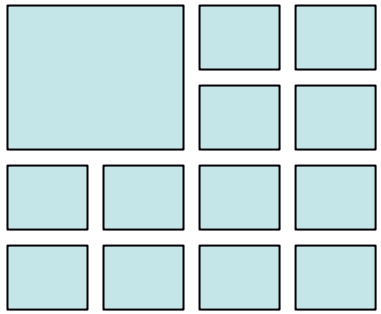
- Technology scaling will **continue** (for some time)
- Clock-frequency scaling has **slowed down**
- Power budget growth has **slowed down**
- Memory bandwidth growth has **slowed down**

The Road Forward

- **P**arallelism (any form) is our only hope
- **P**ower efficiency is a first-order concern
  - Using compute and memory resources efficiently is key

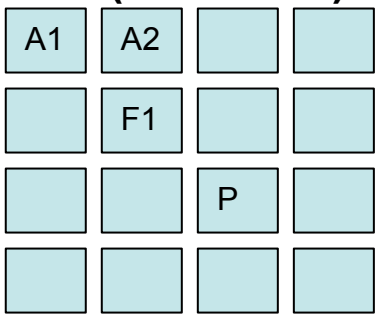
-> **Heterogeneous multicore architectures**

Capability heterogeneous (single ISA)



e.g, ARM big/LITTLE

Functionally heterogeneous (multi ISA)



Accelerators, GPU etc.

Challenges Ahead

Significant enhancements of

- Programmability and
- Power efficiency

are needed

(My) thesis:

Both can be addressed with

aggressive resource management

”under the hood”

Vision: HW/SW Interface in the Multicore Era

Productivity layer (concurrency ”agnostic” for productivity programmers)

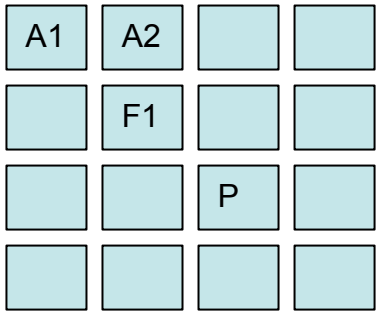
Efficiency layer (concurrency ”aware” for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

New primitives (HW/SW)

Parallelism/Power mngmt (HW/SW)



Architectural support for programmability

Agenda

- ✓ Trends in parallel architectures
- Parallelism management (prog. model->arch.)
- Power management (prog. model->arch.)
- Value locality and cache management
- Concluding remarks

Programmability

The Four Hard Steps in Parallel Programming

1. Decomposition
2. Assignment
3. Orchestration
4. Mapping

Goal: Expose concurrency

Goal: Bundle concurrent tasks into parallel threads

Goal: Map implementation of parallel program to architecture

Goal: Map design to programming model primitives (e.g OpenMP, Cilk, etc)

What are the Difficulties?

Process	Goal	Difficulties
1. Decomposition	Expose concurrency	Respect dependences

What are the Difficulties?

Process	Goal	Difficulties
1. Decomposition	Expose concurrency	Respect dependences
2. Assigment	Bundle concurrent tasks into parallel threads/tasks	Load balancing vs locality and communication

What are the Difficulties?

Process	Goal	Difficulties
1. Decomposition	Expose concurrency	Respect dependences
2. Assigment	Bundle concurrent tasks into parallel threads	Load balancing vs locality and communication
3. Orchestration	Map design to programming model	Factor in thread management & synchronization overhead

What are the Difficulties?

Process	Goal	Difficulties
1. Decomposition	Expose concurrency	Respect dependences
2. Assigment	Bundle concurrent tasks into parallel threads	Load balancing vs locality and communication
3. Orchestration	Map design to programming model	Factor in thread/ task management & synchronization overhead
4. Mapping	Map threads/tasks to architecture	Factor in topology (locality and comm.)

Correctness issues

Efficiency issues

Aim at a moving architecture target

Vision: Parallel Programming

”Productivity” programmers: No parallelism concerns

”Efficiency” programmers

1a. Express concurrency

1b. Enforce dependences

2. Assignment

3. Orchestration

4. Mapping

“UNDER THE HOOD”
Compiler & runtime support **with substantial architecture support**

Vision: HW/SW Interface for Parallelism Management

Productivity layer (concurrency ”agnostic” for productivity programmers)

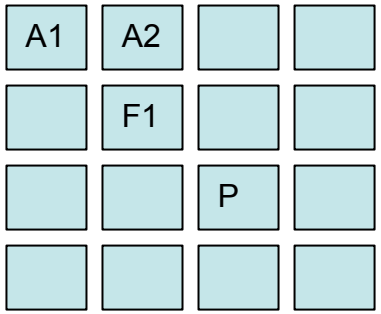
Efficiency layer (concurrency ”aware” for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

New primitives (HW/SW)

Parallelism/Power mngmt (HW/SW)

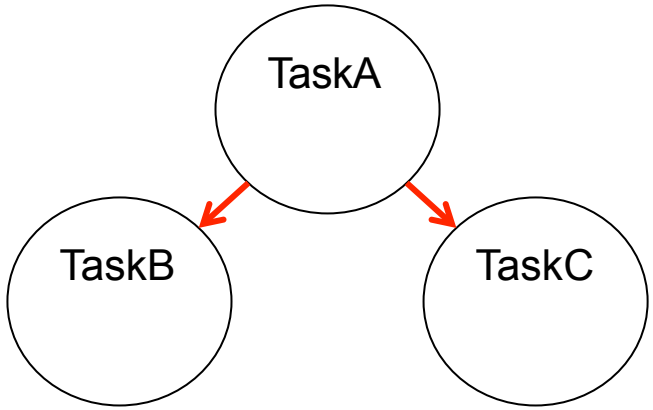


Architectural support for programmability

Programming Models

Process	Goal	Difficulties	Programming Models
1. Decomposition	Expose concurrency	Respect dependences	StarSS (OpenMP 4.0)
2. Assigment	Bundle concurrent tasks into parallel threads	Load balancing vs locality and communication	Cilk, OpenMP 3.0, TBB, CUDA, OpenCL,…
3. Orchestration	Map design to programming model	Factor in thread management & synchronization overhead	OpenMP <3.0
4. Mapping	Map threads to architecture	Factor in topology (locality and comm.)	Pthreads

Task-based Dataflow Prog. Models




```
#pragma css task output(a)
void TaskA( float a[M][M]);

#pragma css task input(a)
void TaskB( float a[M][M]);

#pragma css task input(a)
void TaskC( float a[M][M]);
```

- Programmer annotations for task dependences
- Annotations used by run-time for scheduling
- Dataflow task graph constructed dynamially

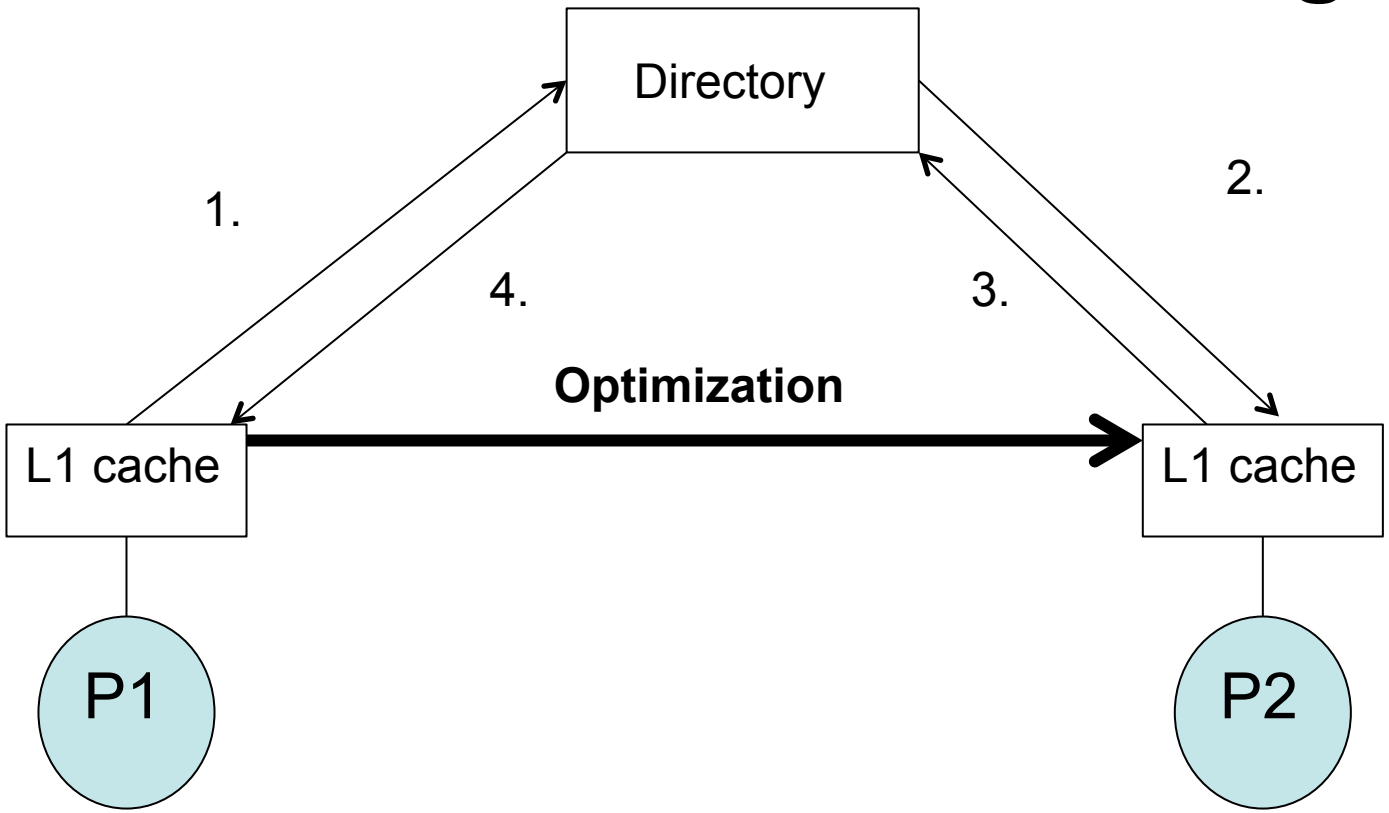
Important: Conveys semantic information to run-time for efficient scheduling

Cache Coherence Optimizations

- **Programmability**. Simplifies porting of legacy software by providing a monolithic memory view
- **Efficiency.** Several concerns:
  - **Latency.** Indirection of requests through a directory
  - **Energy.** Inefficient handling of coarse-grain sharing behavior ⇨ useless traffic
  - **Resources.** Scalability concerns for metadata storage

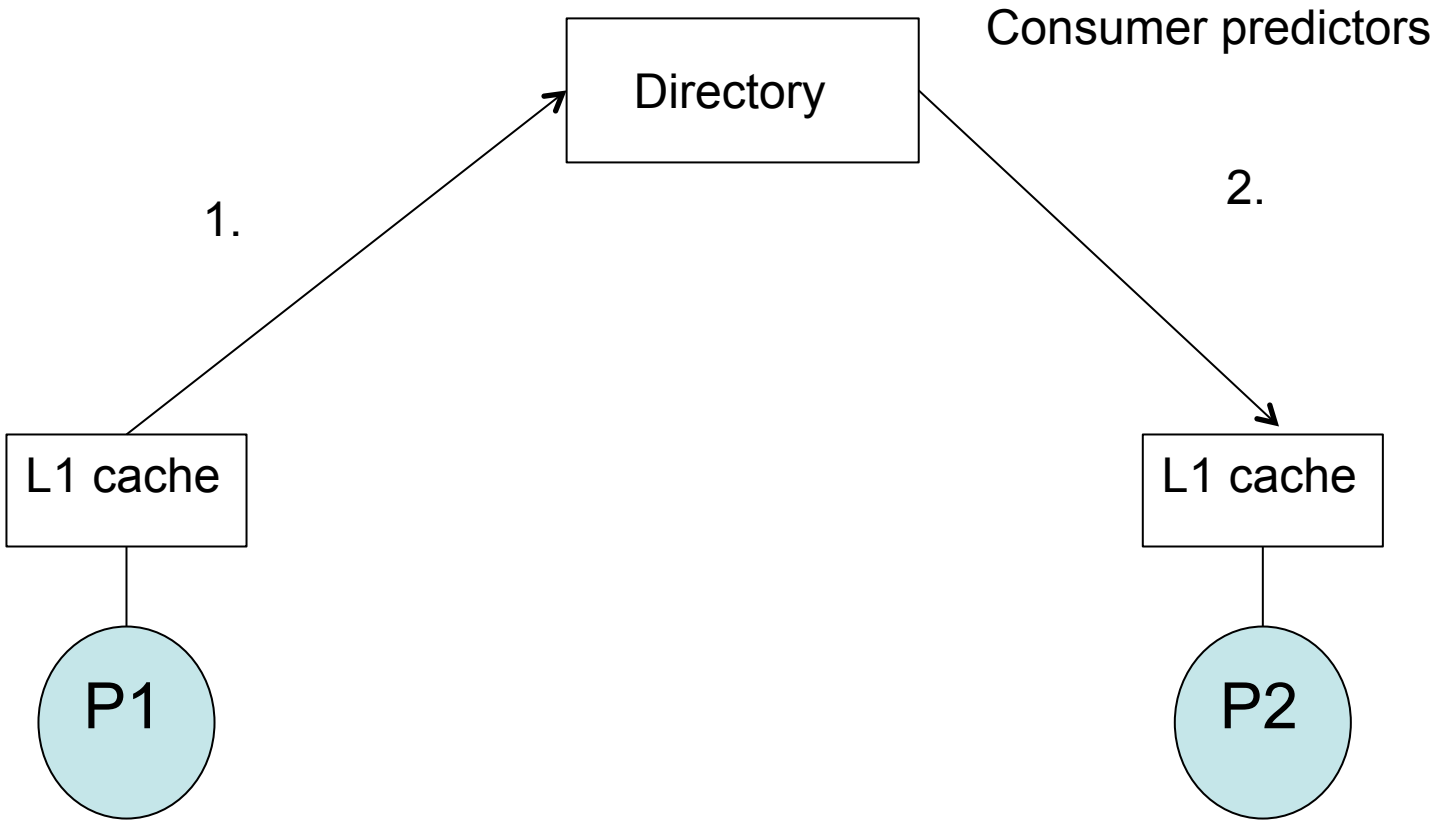
Active research area so mitigation is underway

Latency/Traffic Overhead in Producer/Consumer Sharing



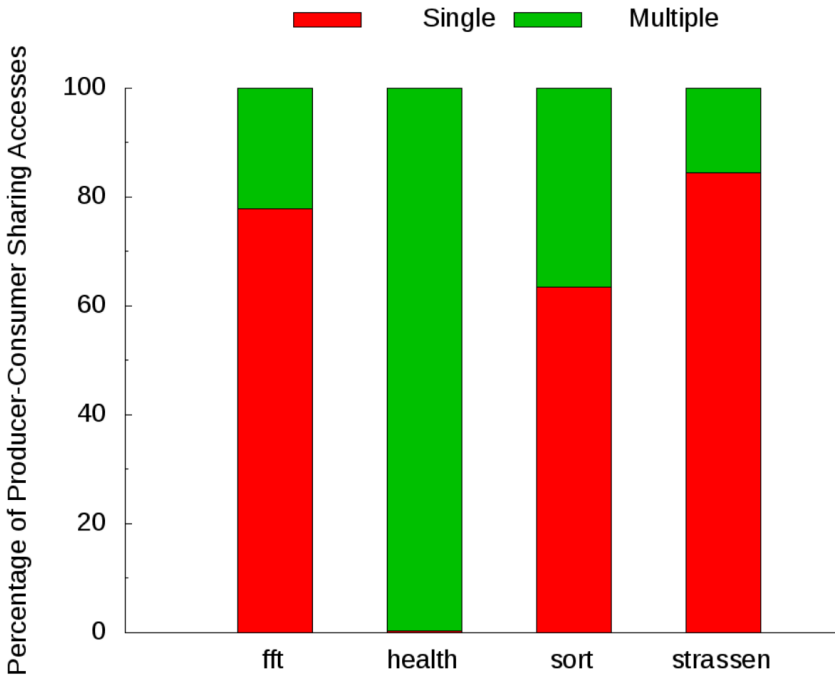
Producer – P1; Consumer – P2

Consumer Prediction



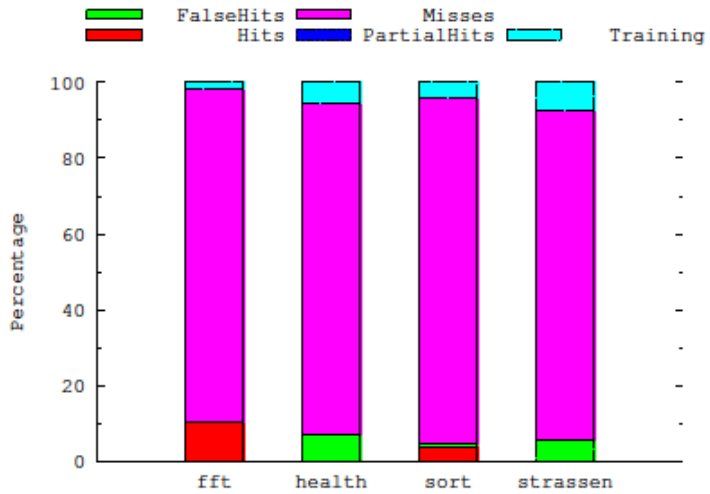
Producer – P1; Consumer – P2

Consumer Prediction Accuracy 1(2)



- Several prod-cons interactions needed to train predictors
- Only few interactions FFT, Sort, and Strassen

Consumer Prediction Accuracy 2(2)



- Low prediction accuracy (<15%)
- **Problem:** Task-based run-time systems reschedule tasks to improve locality or to load-balance (task stealing)

Approach: Use semantic information from the scheduler:
Cooperative coherence prediction

Other Possible Optimizations

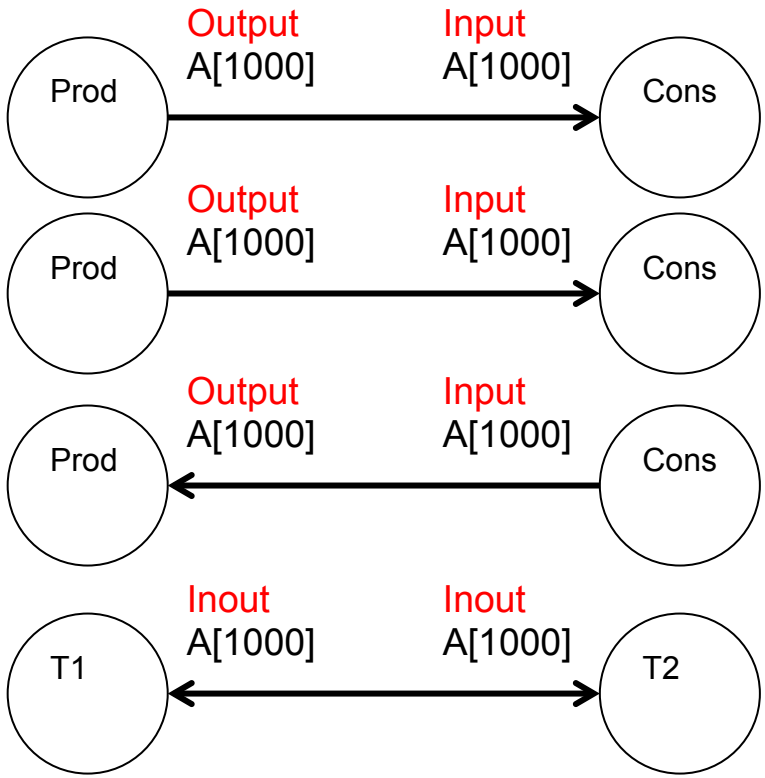
Dependency annotations allow for optimizations with high accuracy (like in message passing)

Bulk data transfer

Forwarding

Prefetching

Migratory sharing optimization



Vision: HW/SW Interface in the Multicore Era

Productivity layer (concurrency ”agnostic” for productivity programmers)

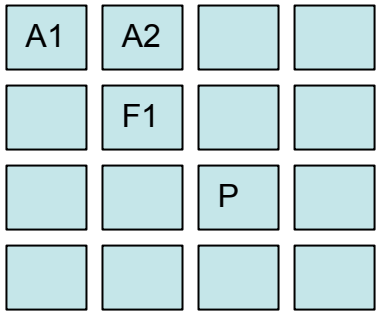
Efficiency layer (concurrency ”aware” for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

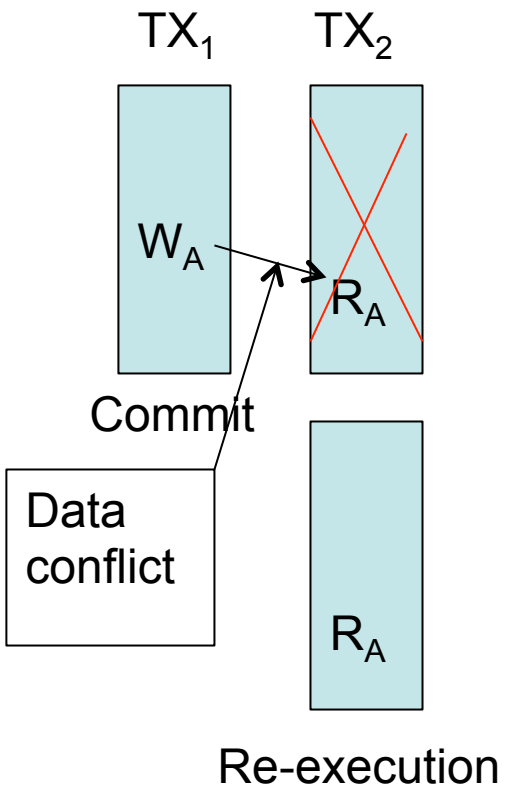
New primitives (HW/SW)

Parallelism/Power mngmt (HW/SW)



Architectural support for programmability

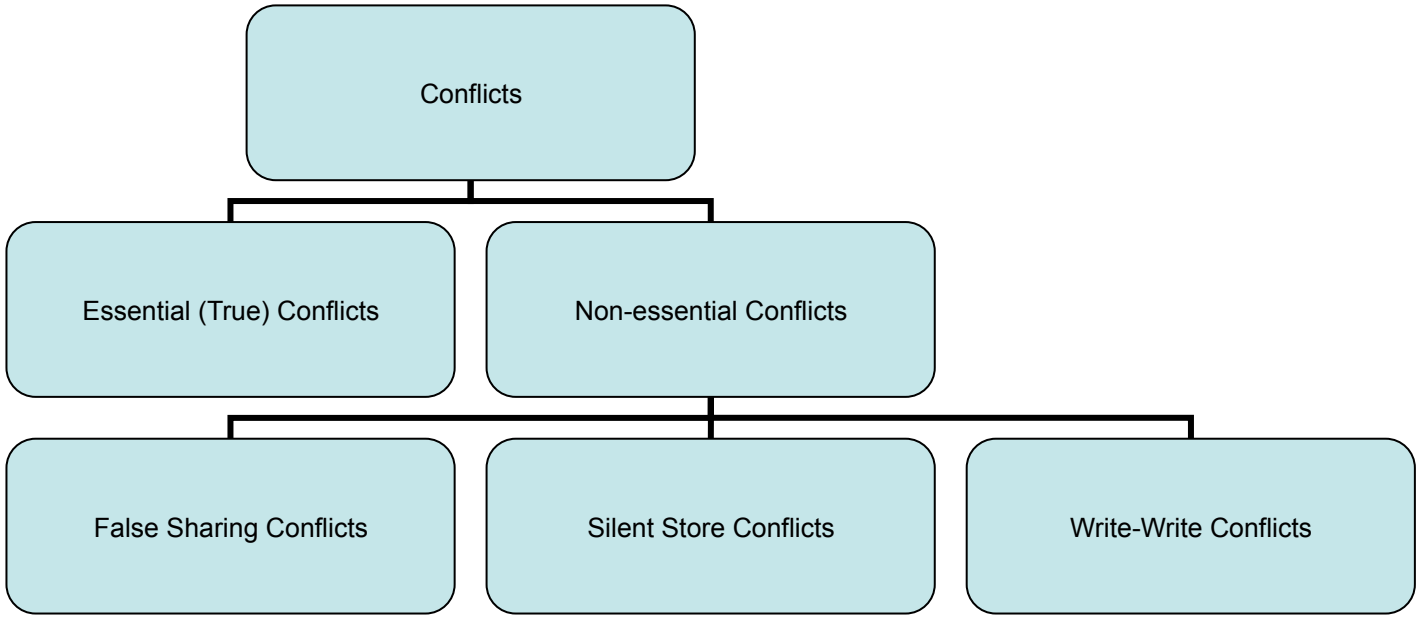
Transactional Memory (TM)



- **Transactional memory semantics:**
  - Atomicity, consistency, and isolation
  - Tx_begin/Tx_end primitives
- Allow for concurrency inside critical sections
- Software implementations too slow
- Hardware implementations complex but have been adopted (IBM Bluegene, Intel Haswell)
- 100s of papers in the open literature; design space fairly well understood

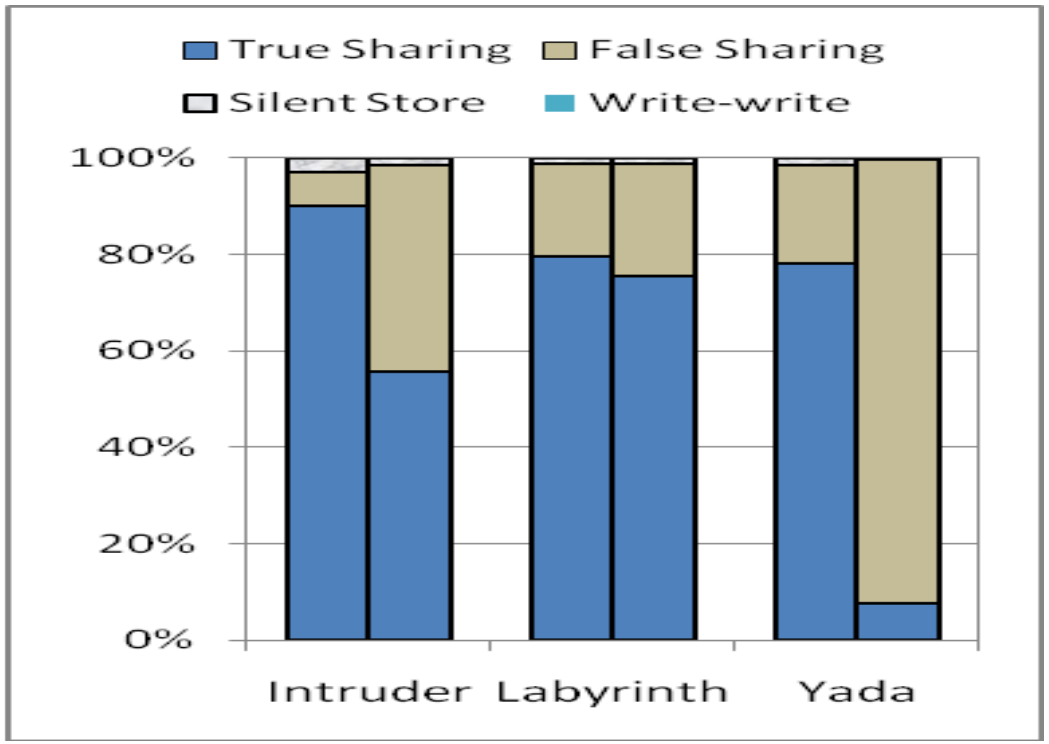
Is the TM abstraction a good idea?

Root Causes of Conflicts in TM



- Essential conflicts stem from inherent communication
- Non-essential conflicts are artifactual and can be avoided

Impact of Data Conflicts



- True and false conflicts dominate
- Silent store conflicts are rare and no write-write conflicts

Agenda

- ✓ Trends in parallel architectures
- ✓ Parallelism management (prog. model->arch.)
- Power management (prog. model->arch.)
- Value locality and cache management
- Concluding remarks

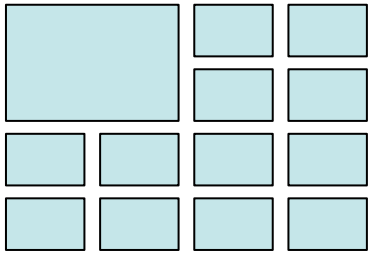
Power Management

- Tasks have different QoS levels

Problem: No way to express it, yet

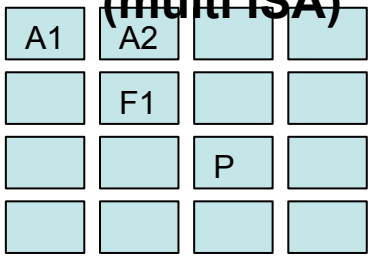
- If tasks had explicit deadlines and known running times as a function of architectural resources, we could do significantly better in power management
- Need advancement across layers: (prog model, compiler, run-time, architecture)

Capability heterogeneous (single ISA)



e.g, ARM big/LITTLE

Functionally heterogeneous (multi ISA)



Accelerators, GPU etc.

Vision: HW/SW Interface in the Multicore Era

Productivity layer (concurrency ”agnostic” for productivity programmers)

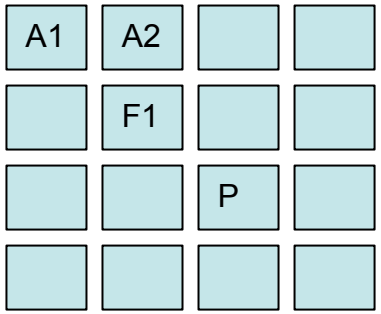
Efficiency layer (concurrency ”aware” for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

New primitives (HW/SW)

Parallelism/Power mngmt (HW/SW)



Architectural support for programmability

Agenda

- ✓ Trends in parallel architectures
- ✓ Parallelism management (prog. model->arch.)

- Value locality and cache management
- Concluding remarks

Agenda

- ✓ Trends in parallel architectures
- ✓ Parallelism management (prog. model->arch.)
- ✓ Power management (prog. model->arch.)
- Value locality and cache management
- Concluding remarks

Memory Hierachies are Inefficient

Performance

- Processor/memory speed-gap is increasing
- Off-chip memory bandwidth does not scale

Power

- Significant portion is spent in mem. hierarchy

Resource usage

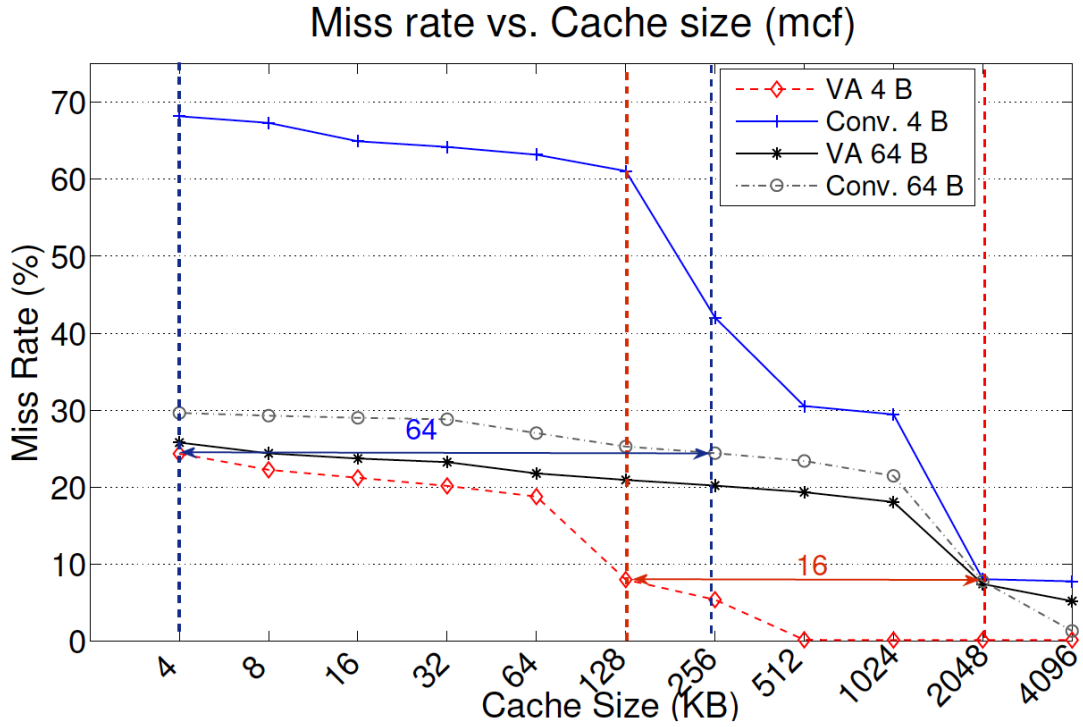
- Cache (memory) resources are used inefficiently

Major source of inefficiency: value replication

Value Locality

Observation:

– A value is typically replicated across many locations



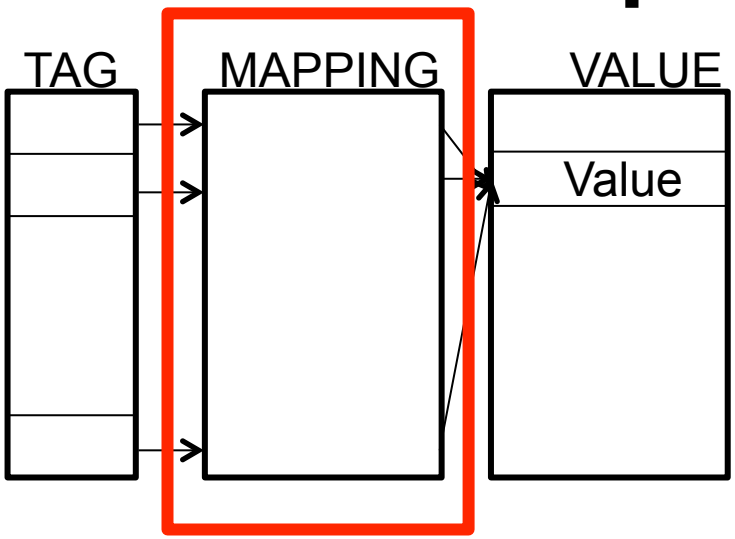
Potential compression ratio: ~ 32X

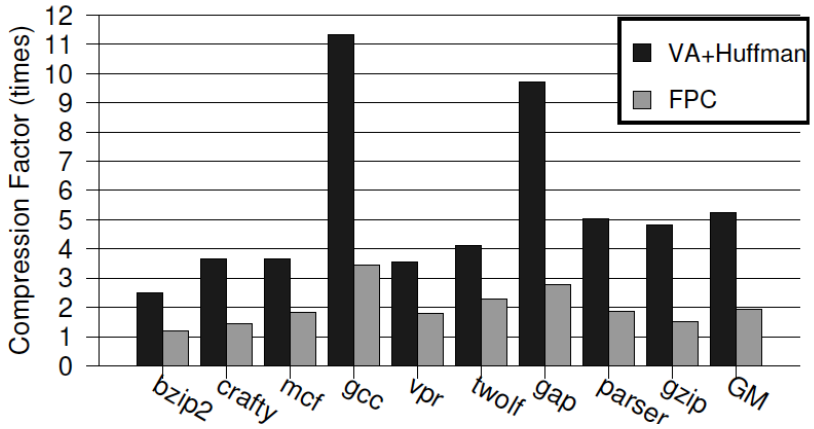
Columbus and Huffman (unfairly) got Credit for Viking Discoveries

=

=

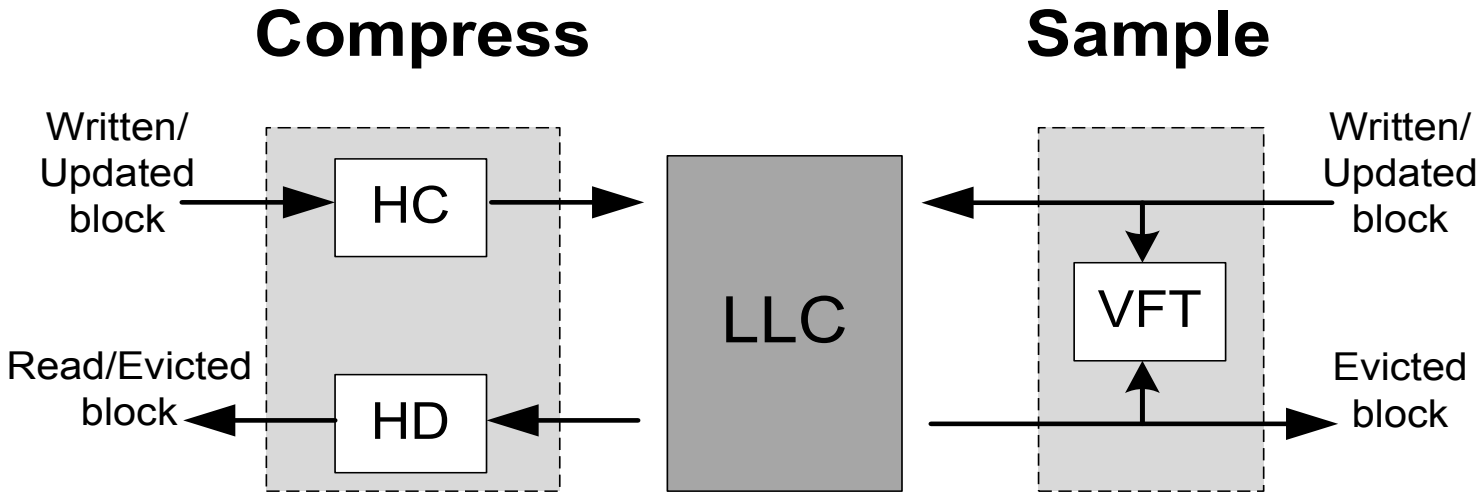
Potential of Huffman Cache Compression





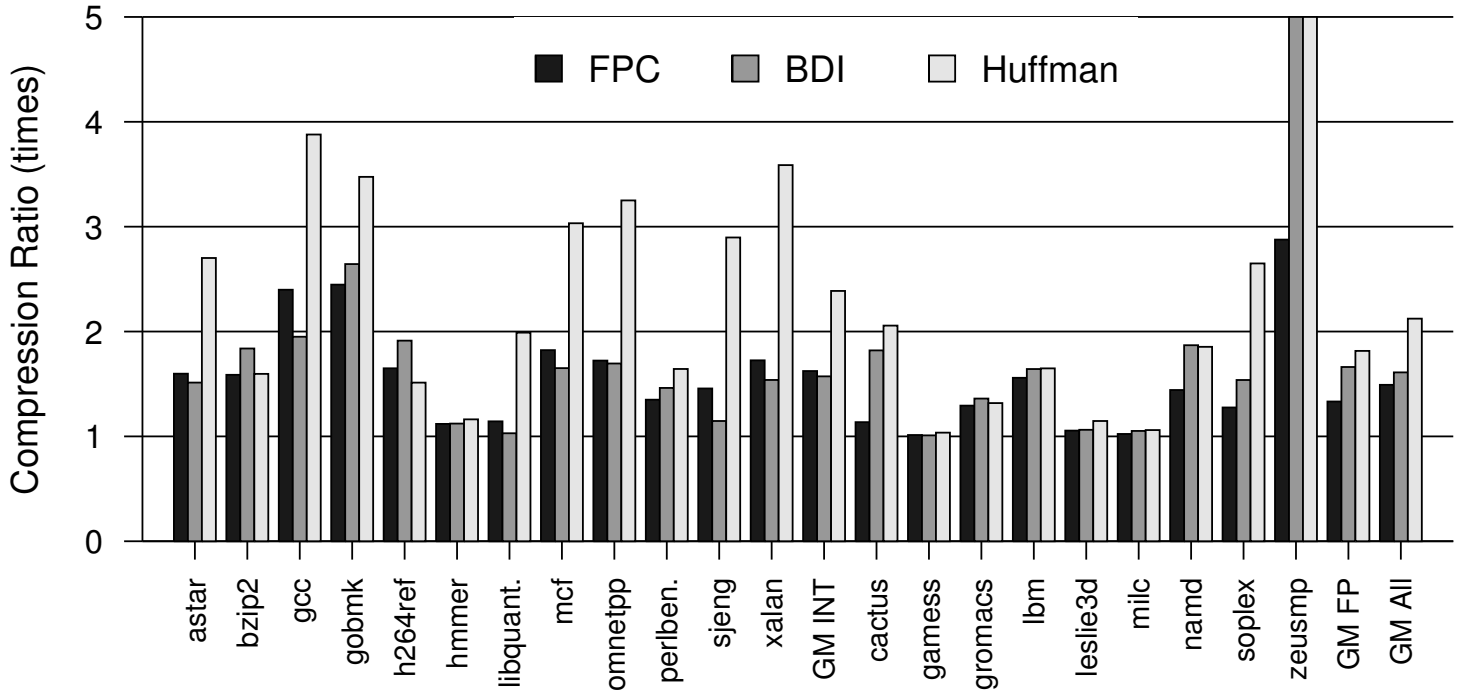
- Compression Factor (CF) = data store(conv)/ mapping
- > 5X on average

Overview



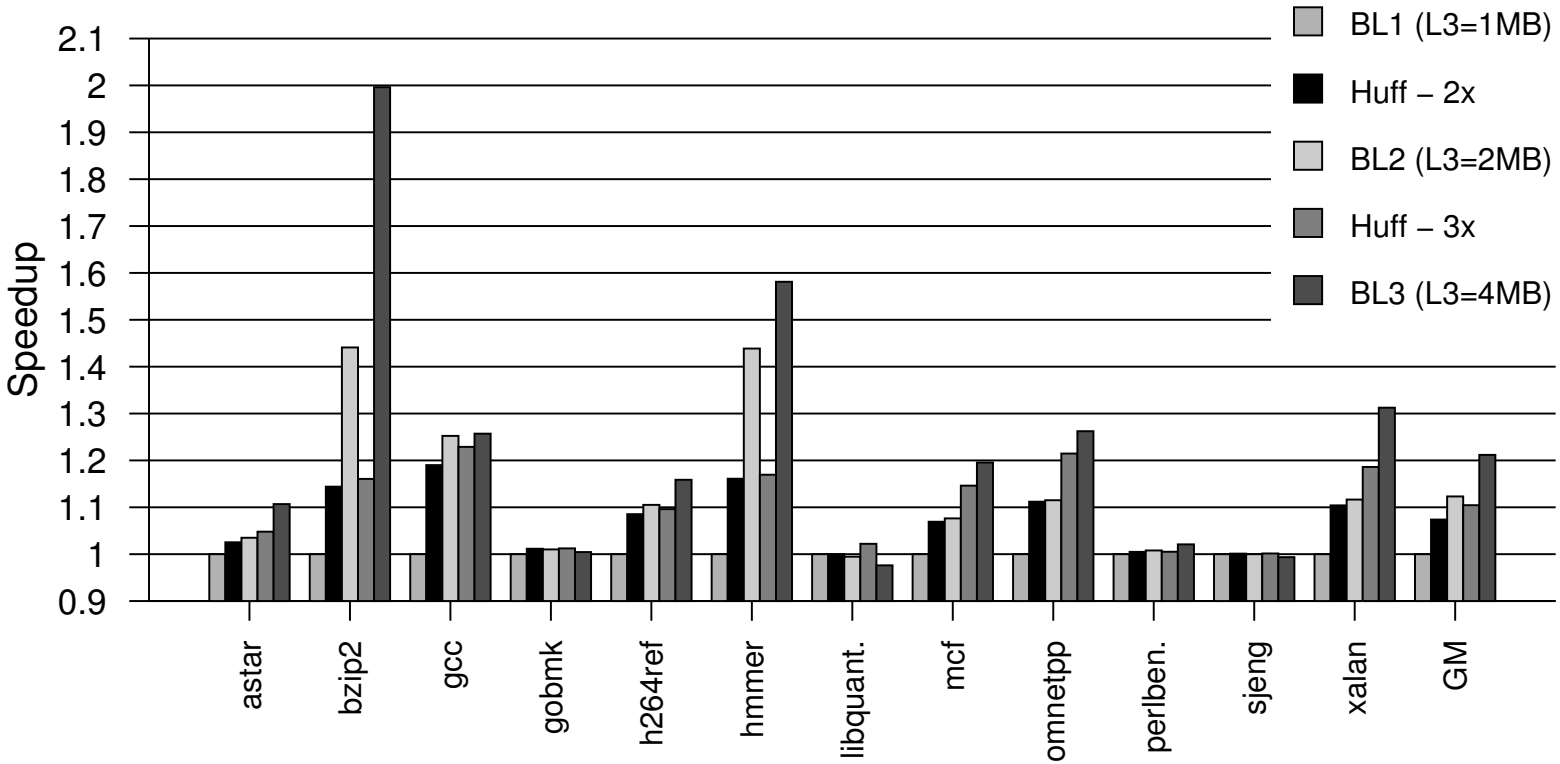
- Applied to last-level cache (LLC)
- Two main processes
  - Sample: To establish value frequency
  - Compress: Apply Huffman coding to cache content

Compression Ratio



- FPC and BDI yield around 1.5X compression
- Huffman yields a compression ratio of 2.2X

Speedup



– Huff – 3X does always better than BL1 and almost as well as BL2 (2X larger cache)

Agenda

- ✓ Trends in parallel architectures
- ✓ Parallelism management (prog. model->arch.)
- ✓ Value locality and cache management
- Concluding remarks

Concluding Remarks

- Big challenges ahead
  - Programmability
  - Power management
- Linking information across layers is key to
  - Enhance programmability
  - Use resources effectively

Thank you!

Questions?