

# High-Performance Scientific Computing

## Lecture 10: Parallel Performance

MATH-GA 2011 / CSCI-GA 2945 · November 14, 2012

# Today

Recent news

Tool of the day: Profilers

Multi-thread performance

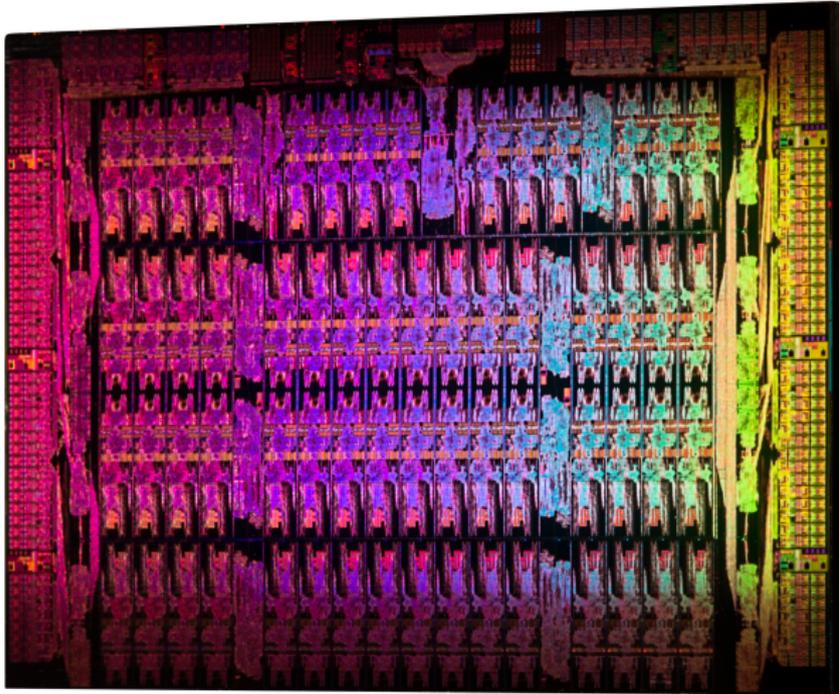
# Outline

Recent news

Tool of the day: Profilers

Multi-thread performance

# Xeon Phi



# Outline

Recent news

Tool of the day: Profilers

Multi-thread performance

# Profilers

Slow program execution:

- Poor memory access pattern
- Expensive processing  
(e.g. division, transcendental functions)
- Control overhead (branches, function calls)

**Desired Insight:**

- *Where is time spent?* (Source code location)
- *When?* (Execution History)
  - Call stack
- *What is the limiting factor?*

**Main Types of Profilers:**

- Exact, Sampling
- Hardware, Software



# Reflections on Profilers

Sampling	Exact
 Fast	 Slow
 Noisy (takes time to converge!)	 Exact

No free lunch. But: No exact machine-level profiler!

# Various profilers

List of profilers:

- Gprof: sampling, software, single-program
- Sysprof: sampling, software, system-wide
- Valgrind: exact, 'hardware', single-program
  - callgrind, cachegrind, reallly
- Perf: sampling, hardware, system-wide

Demo time

# Making sense of Perf sample counts

What do Perf sample counts mean?

Individually: not much!

→ Ratios make sense!

What kind of ratios?

- $(\text{Events in Routine 1}) / (\text{Events in Routine 2})$
- $(\text{Events in Line 1}) / (\text{Events in Line 2})$
- $(\text{Count of Event 1 in X}) / (\text{Count of Event 2 in X})$



**Always ask:** Sample count sufficiently converged?

## Perf: Examples

- **instructions / cycles**

Instructions per clock, target  $> 1$  (seen)

- **L1-dcache-load-misses / instructions**

L1 miss rate, target: small, location understood (demo)

- **LLC-load-misses / instructions**

L2 miss rate, target: small

- **stalled-cycles-frontend / cycles**

Instruction fetch stalls. Should never happen—means CPU could not predict where code is going. ( $\rightarrow$  pipeline stall)

- **stalled-cycles-backend / cycles**

Execution units (ALU/FPU/Load-store) is waiting for data/computation/...

## Perf: Examples

- **instructions / cycles**

Instructions per clock, target  $> 1$  (seen)

- **L1-dcache-load-misses / instructions**

L1 miss rate, target: small, location understood (demo)

- **LLC-load-misses / instructions**

L2 miss rate, target: small

- **stalled-cycles-frontend / cycles**

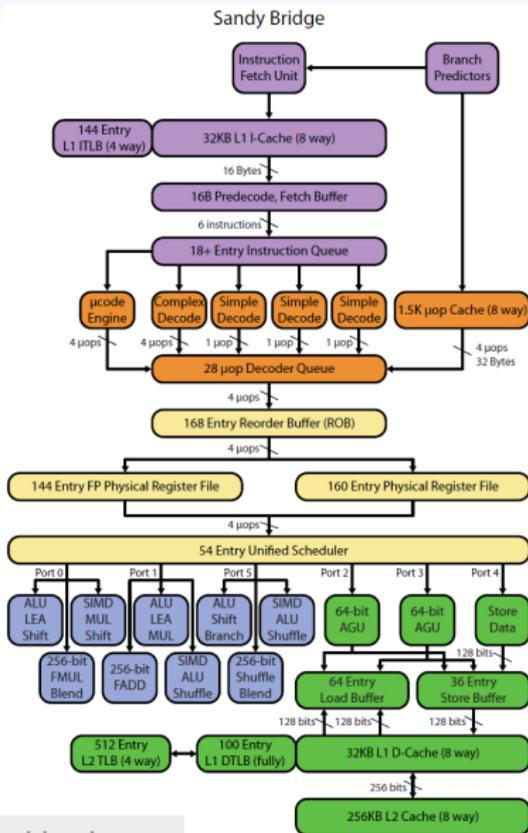
Instruction fetch stalls. Should never happen—means CPU could not predict where code is going. ( $\rightarrow$  pipeline stall)

- **stalled-cycles-backend / cycles**

Execution units (ALU, FP, ...) stalled  
data/computation

Front end? Back end?

# Front end and Back end



# Learning about PMU events

- [Intel Optimization Manual](#) (no.)
  - [Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3B](#)  
(yes!)
- [AMD Optimization Manual](#) (no.)
  - [AMD BIOS and Kernel Developers' guide for Family 15h processors](#)  
(yes!)

Latter contain event descriptions.

Former contain advice on what ratios to use.

# Perf low-level hw event demo

# Outline

Recent news

Tool of the day: Profilers

Multi-thread performance

- Memory-related

- Non-memory-related

# Multi-thread performance

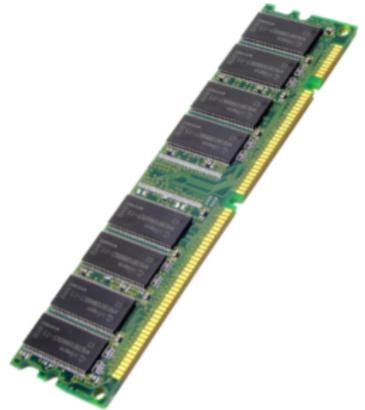
Difference to single-thread?

# Multi-thread performance

Difference to single-thread?

**Memory System** is (about) the only shared resource.

All 'interesting' performance behavior of multiple threads has to do with that.



# Outline

Recent news

Tool of the day: Profilers

**Multi-thread performance**

Memory-related

Non-memory-related

# Threads v. caches demo

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

- Modified** most recent correct copy, memory stale. No other copies.
- Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.
- Exclusive** most recent, correct copy, memory fresh. No other copies.
- Shared** most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale.
- Invalid** no valid copy of the data.

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

**Modified** most recent correct copy, memory stale. No other copies.

**Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.

**Exclusive** most recent, correct copy, memory fresh. No other copies.

**Shared** most recent, correct copy. Other CPUs may hold copies in O and S state. Memory may be stale

**Invalid** no

What states are safe to write? (in my and someone else's cache)

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

**Modified** most recent correct copy, memory stale. No other copies.

**Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.

**Exclusive** most recent, correct copy, memory fresh. No other copies.

**Shared** m  
co

**Invalid** no

What states are safe to write? (in my and someone else's cache)

(and transitions to what state?)

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

**Modified** most recent correct copy, memory stale. No other copies.

**Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly stale) memory on evict.

**Exclusive** m

co

What states are safe to write? (in my and someone else's cache)

**Shared** m

co

(and transitions to what state?)

**Invalid** no

What states did the sums array see?

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

**Modified** most recent correct copy, memory stale. No other copies.

**Owned** most recent, correct copy. Other CPUs may hold copy in S state. Responsible for updating (possibly

**Exclusive** m  
co  
st  
What states are safe to write? (in my and someone else's cache)

**Shared** m  
co  
(and transitions to what state?)

**Invalid** no  
What states did the sums array see?

How do memory fences fit into this picture?

# Cache coherency

*Example:* “MOESI” protocol (e.g. AMD). A cache line holds. . .

**Modified** most recent correct copy, memory stale. No other copies.

**Owned** most recent correct copy. Other CPUs may hold

co  
st

What states are safe to write? (in my and someone else's cache)

**Exclusive** m

co

(and transitions to what state?)

**Shared** m

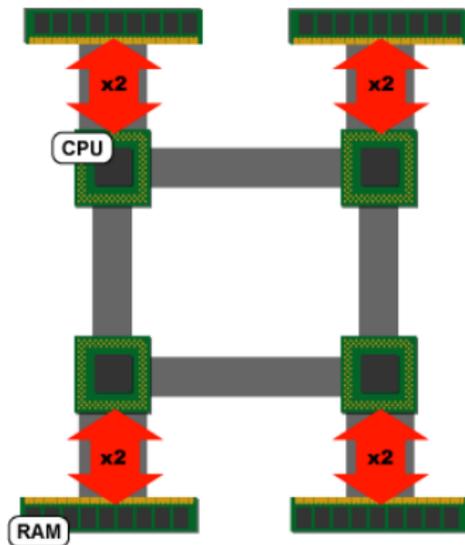
co

What states did the sums array see?

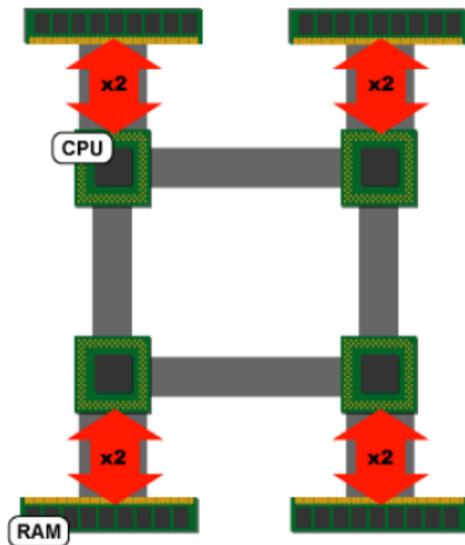
**Invalid** no

How do memory fences fit into this picture?  
None of this is instantaneous → queued!

# Multiple sockets?



# Multiple sockets?



“NUMA”

# Contention/throughput demo



# NUMA results

'crunchy3' at Courant

```
sequential core 0 -> core 0 : BW 4189.87 MB/s
sequential core 1 -> core 0 : BW 2409.1 MB/s
sequential core 2 -> core 0 : BW 2495.61 MB/s
sequential core 3 -> core 0 : BW 2474.62 MB/s
sequential core 4 -> core 0 : BW 4244.45 MB/s
sequential core 5 -> core 0 : BW 2378.34 MB/s
....
sequential core 29 -> core 0 : BW 2048.68 MB/s
sequential core 30 -> core 0 : BW 2087.6 MB/s
sequential core 31 -> core 0 : BW 2014.68 MB/s
```

# NUMA results

'crunchy3' at Courant

```
all -contention core 0 -> core 0 : BW 1081.85 MB/s
all -contention core 1 -> core 0 : BW 299.177 MB/s
all -contention core 2 -> core 0 : BW 298.853 MB/s
all -contention core 3 -> core 0 : BW 263.735 MB/s
all -contention core 4 -> core 0 : BW 1081.93 MB/s
all -contention core 5 -> core 0 : BW 299.177 MB/s
....
all -contention core 27 -> core 0 : BW 202.49 MB/s
all -contention core 28 -> core 0 : BW 434.295 MB/s
all -contention core 29 -> core 0 : BW 233.309 MB/s
all -contention core 30 -> core 0 : BW 233.169 MB/s
all -contention core 31 -> core 0 : BW 202.526 MB/s
```

# NUMA results

'crunchy3' at Courant

two-contention core 0 → core 0 : BW 3306.11 MB/s

two-contention core 1 → core 0 : BW 2199.7 MB/s

two-contention core 0 → core 0 : BW 3257.56 MB/s

two-contention core 19 → core 0 : BW 1885.03 MB/s

# NUMA? Do I need to care?

Large multi-core machines *are* NUMA.

Also: Easy, can use OpenMP → popular

What happens if you ignore NUMA?

- What happens at `malloc`?
- What happens at 'first touch'?
- What happens if you don't pin-to-core?

# Outline

Recent news

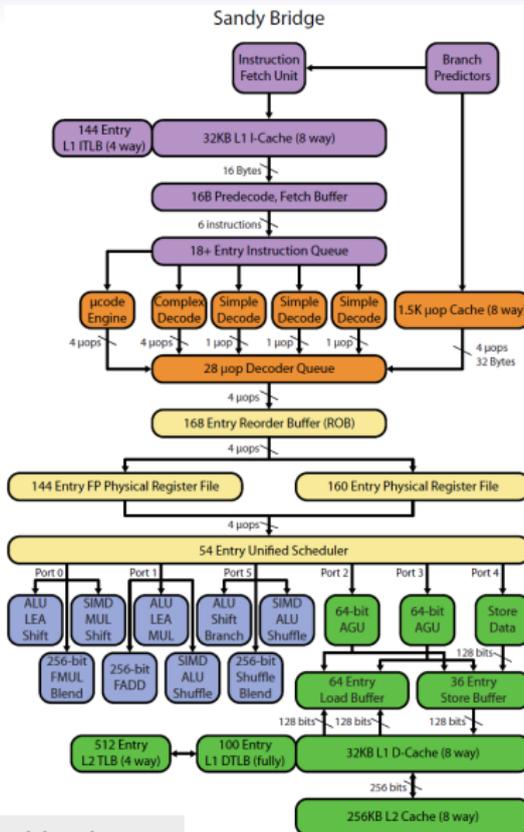
Tool of the day: Profilers

**Multi-thread performance**

Memory-related

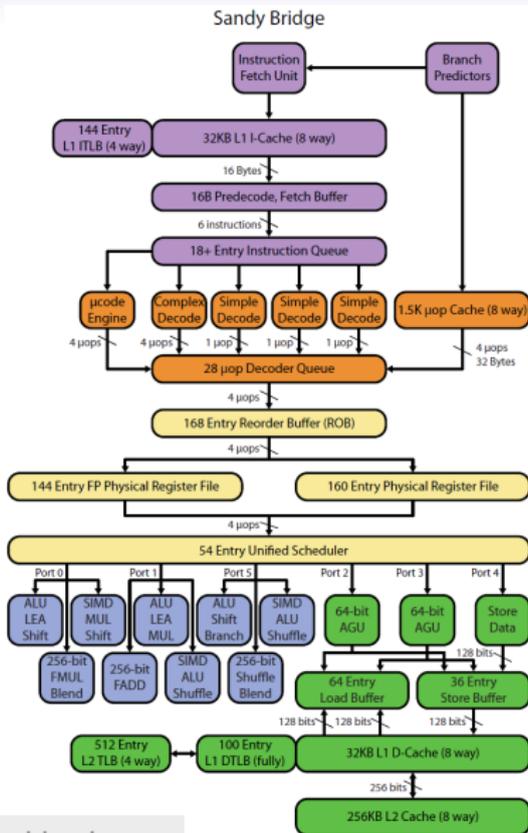
Non-memory-related

# Recap: superscalar architecture



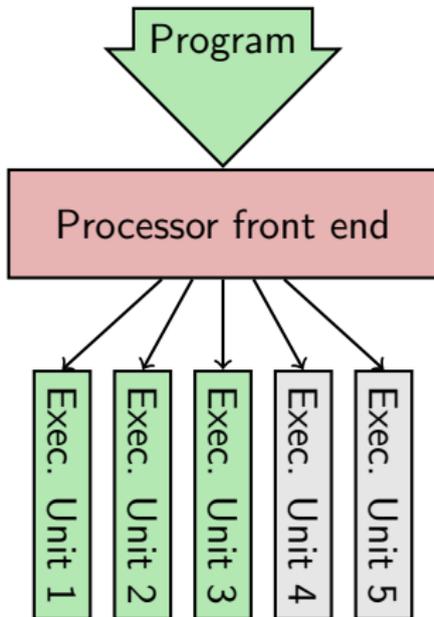
David Kanter / Realworldtech.com

# Recap: superscalar architecture

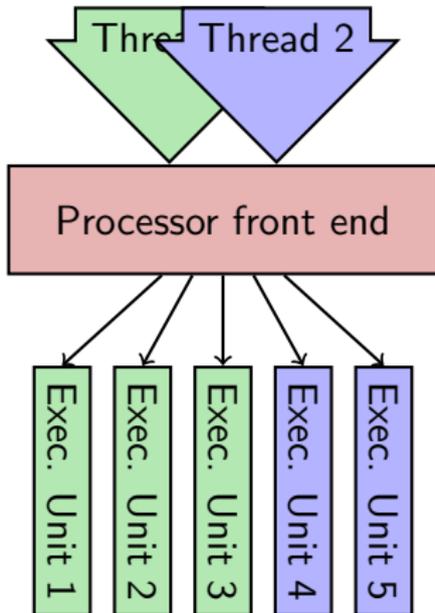


David Kanter / [Realworldtech.com](http://Realworldtech.com)

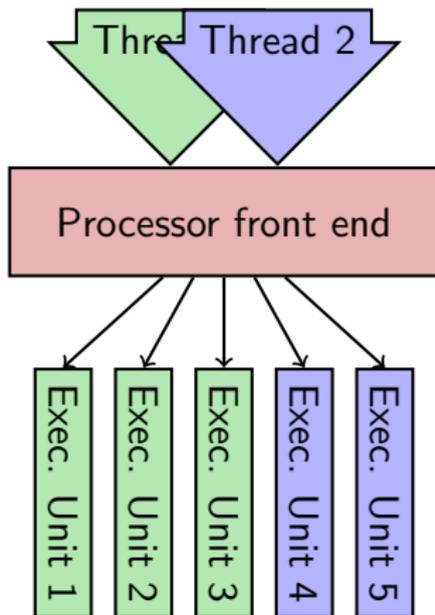
# SMT / "Hyperthreading"



# SMT / "Hyperthreading"

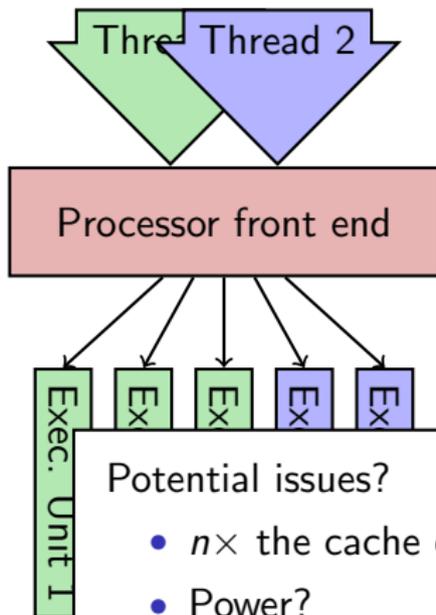


# SMT / "Hyperthreading"



Potential issues?

# SMT / "Hyperthreading"



Potential issues?

- $n \times$  the cache demand!
- Power?

→ Some people just turn it off and manage their own ILP.

# Locks

Locks are not slow

Lock *contention* is slow

# Locks

Locks are not slow

Lock *contention* is slow

Demo, also → HW2

Questions?

?

# Image Credits

- Clock: [sxc.hu/cema](http://sxc.hu/cema)
- Bar chart: [sxc.hu/miamiamia](http://sxc.hu/miamiamia)