CS 6354: SMT

28 September 2016

To read more...

This day's papers:

Tullsen et al, "Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor"
Alverson et al, "The Tera Computer System"

Supplementary Reading:

Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, Section 3.12
Kongetira et al, "Niagara: A 32-Way Multithreaded Sparc Processor"
Shin and Lipasti, *Modern Processor Design*, Section 11.4.4

Definition: Thread

stream of program execution

own registers

own program counter (current instruction pointer)

may or may not share memory

appears to execute at same time as other threads

Multithreading

```
thread_one_func(int offset) {
    for (int i = 0; i < N / 2; ++i)
        sum1 += array[i];
}
thread_two_func() {
    for (int i = N / 2; i < N; ++i)
        sum2 += array[i];
}
compute_sum() {
    thread_one = thread_create(thread_one_func);
    thread_two = thread_create(thread_two_func);
    wait_for_thread(thread_one);
    wait_for_thread(thread_two);
    sum = sum1 + sum2;
}
```

OS context switches

```
sum1 += array[offset + 0];
if (0 < N / 2) goto done1;
...
sum1 += array[offset + 1940];
if (1940 < N / 2) goto done1;
```

timer interrupt/exception

copy registers to memory
OS runs
load registers from memory

return from interrupt/exception

```
sum2 += array[offset + N/2 + 0];
if (0 < N / 2) goto done2;
...
sum2 += array[offset + N/2 + 1849];
if (1849 + N / 2 < N) goto done2;
```

threads state AKA context

externally visible:

- program counter (current instruction)
- (program-visible) registers
- (address of page table)

maybe shared between threads: memory

threads may or may not be in seperate programs

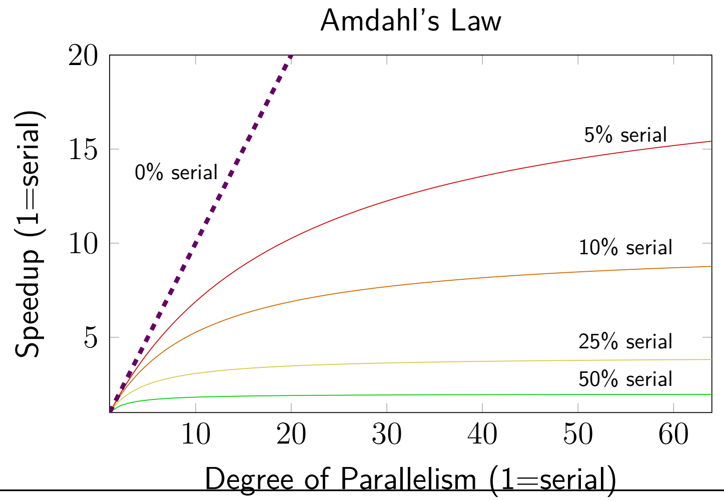
two approaches

Exploiting Choice	Tera
out-of-order	in-order
choose thread dynamically	round-robin between threads
many register name maps	many register files
schedule when ready	compiler-specified delays
reorder buffer	in-order completion imprecise exceptions
1-cycle data cache	70-cycle data memory

Tera: Is it usable?

minimum of nine threads to get full throughput

$\times 256$ CPUs $= 2304$ threads



Tera: the commercial version

Tera/Cray MTA (1997) — described in paper (took 7 years!)

Cray MTA-2 (2002)

Cray XMT (2009) — combines with conventional processors for I/O

not advertised anymore

a complaint

Why doesn't Tera paper compare to superscalar/out-of-order?

- 1960s: IBM, Control Data Corp. machines
- 1988: Motorola MC88100
- 1989: Intel i960CA
- Tera paper
- 1990: AMD 29000
- 1992: DEC Alpha 21064
- 1993: Pentium
- 1994: MIPS R8000

thread state — running superscalar

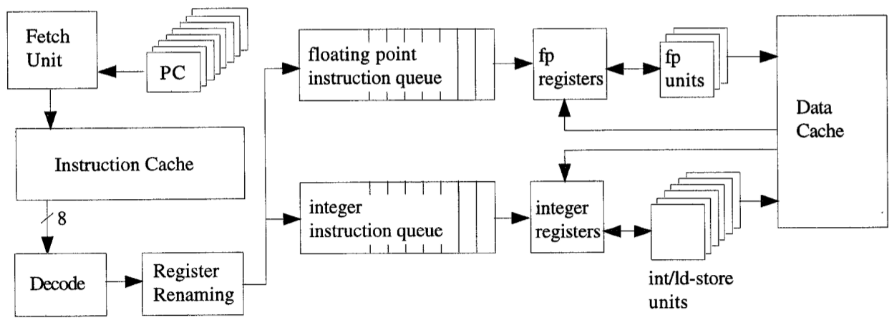


Figure 1: **Our base simultaneous multithreading hardware architecture.**

thread state AKA context

externally visible:

- program counter (current instruction)
- (program-visible) registers

internal:

- queued instructions
- reorder buffer
- program counters
- branch prediction info
- physical register values
- register map

modern SMT systems

most Intel desktop/laptop chips — 2 threads/core
- 2nd gen. Pentium 4 (“NetBurst”) (2002)

Oracle SPARC T5 (2013) — 8 threads/core
- SPARC T1 (2005) — 4 threads/core

IBM POWER8 (2013) — 8 threads/core
- POWER5 (2004) — 2 threads/core

running two threads

no context switch

duplicate thread state

shared resources

caches

instruction queues

functional units (adders, multipliers, etc.)

load/store queue

physical registers

duplicated resources

program counters

return address stack (branch prediction)

register maps

reorder buffer???

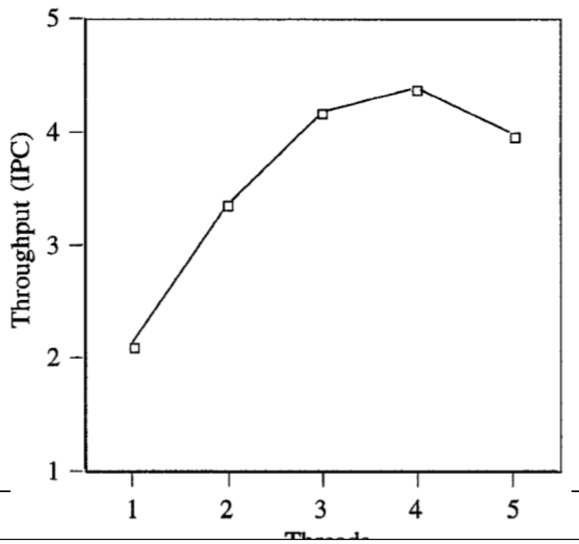
thread ids added to resources

branch target buffer — phantom branches

8-issue processor??

maximum throughput: 8 instructions/cycle

actual throughput: approx. 4.5



what workloads benefit?

two floating point intensive threads?
- how many floating point adders?

two intensive integer threads?
- how many integer ALUs?

two cache-bound threads?
- how many cache accesses per cycle?

two branch-heavy threads?

one intuition

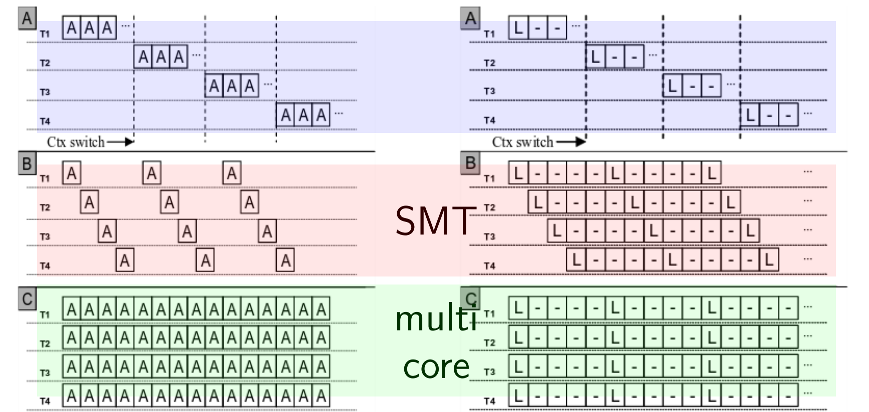


Figure 1. CPU-bound workload. Threads T1 through T4 contend for ALU on each cycle. Systems A and B perform comparably, because they each have one ALU. System C has an ALU per each of its four processors, and it outperforms A and B by a factor of four.

Figure 2. Memory-bound workload. Systems B and C outperform System A by a factor of four, because they are able to overlap memory access latencies for the four threads.

When running the memory-bound workload, System

Figure from Fedorova et al, "Chip multithreading systems need a new operating system scheduler", 2004

variable gains

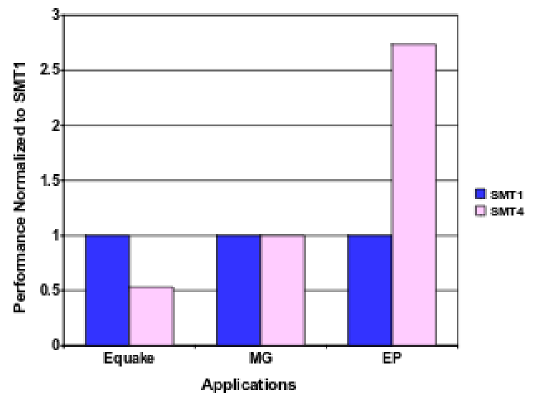


Fig. 1: Comparison of performance with SMT1 vs. SMT4 for 3 applications on an 8-core POWER7 system. Each application is run alone in a separate experiment. The application uses 8 threads under SMT1 and 32 threads under SMT4.

Figure from Funston, et al, "An SMT-Selection Metric to Improve Multithreaded Applications' Performance", IPDPS 2012

added complexity?

huge number of registers — slower regfile
- Exploiting Choice: useful for single thread

more complex interrupt logic
- Tera: imprecise arithmetic exceptions
- Tera: in-order completion

fetch/branch logic
- Tera: fetch logic = issue logic

removed complexity?

Tera: no data cache
- just have more parallelism!

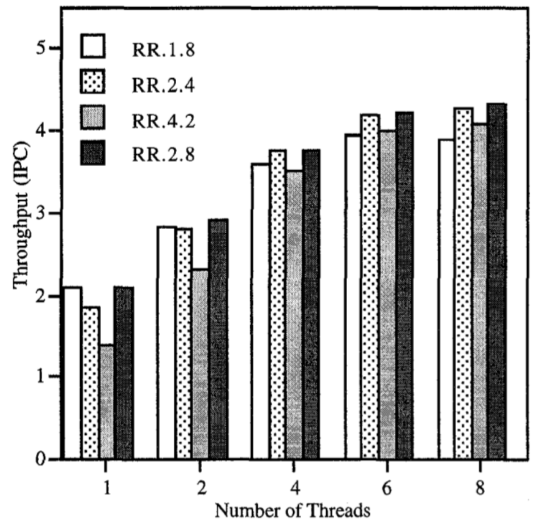
hide long-latency instructions
- instead of better branch prediction
- instead of faster ALUs

round-robin variants

baseline (1.8)
- cycle 1: 8 from thread 1
- cycle 2: 8 from thread 2
- cycle 3: 8 from thread 1
- cycle 4: 8 form thread 2

multiple threads at a time (2.4)
- cycle 1: 4 from thread 1, 4 from thread 2
- cycle 2: 4 from thread 1, 4 from thread 2
- ...

round-robin performance



priority-based fetch

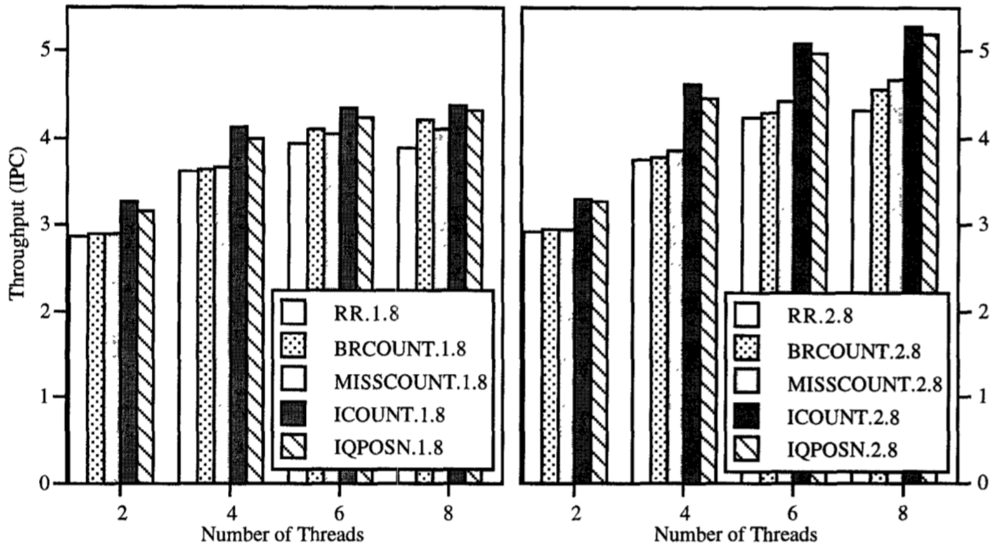
fetch more for faster/more starved threads

less unresolved branches

less cache misses

less pending instructions

priority-based fetch



Tera: thread creation

CREATE instruction

no OS intervention

OS can later move each thread between processors

Exploiting Choice: thread creation

not specified

Intel mechanism: each thread looks like processor

same as multiple processors

"logical processor/core"

Tera: hypertorus

16x16x16 version of:

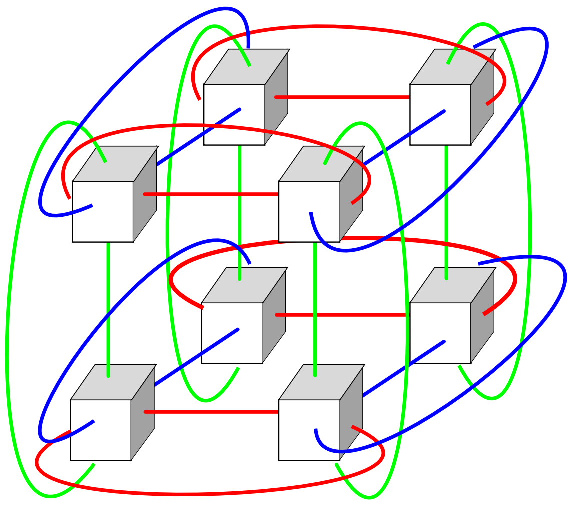


Image: Wikimedia Commons user おむこさん志望

Tera: Synchronization

no caches — single copy of all data

complex commands to memory:

- read
- write
- read/write when ready
- fetch and add

FMA: optimization or benchmark cheat?

Fused Multiply-Add $A = B \times C + D$

single instruction/functional unit use

gives 2 floating point operations/cycle/functional unit

really helps dense matrix math

Next week: multiple processors

C.mmp — one of the earliest multiprocessor

T3E — supercomputer from the 90s

Some weird terminology in C.mmp

not something you are expected to know:

C.mmp deals with core memory (1950s-1970s)

tiny metal rings, magnetized to store a bit

read:

1. set magnetization direction to '0'
2. triggers signal if old direction was '1'
3. rewrite value to old direction

steps 1-2: access time

steps 1-3: cycle time

C.mmp distractions

lots of software issues that don't really concern multiprocessor

you can skim/skip these parts

things to think about when reading

challenges in making multiprocessor machine

design of the networks

how does one program these machines?

how does one coordinate between threads?

how well are threads isolated from each other?

what changes from the uniprocessor were required?