
Fine-grain Multithreading: Sun Niagara, Cray MTA-2, Cray Threadstorm & Oracle T5

John Mellor-Crummey

**Department of Computer Science
Rice University**

johnmc@rice.edu

Context

- **Last class**
 - simultaneous multithreading for improving IPC
 - CMP for scalable parallelism
 - processors today combine the two
 - e.g., Power7, Intel Xeon Phi, Blue Gene/Q, ...
- **Today's focus**
 - architectures based on fine-grain multithreading
 - design points
 - Cray MTA-2 & Cray XMT: scalable high performance shared memory system for technical computing
 - sustained operations per second
 - Sun Niagara: commercial multithreaded server applications
 - sustained throughput of client requests
 - Oracle's SPARC T5

Conventional Parallel Programming Wisdom

For high performance ...

- **Place data near computation**
- **Avoid modifying shared data**
- **Access data in order and reuse**
- **Avoid indirection and linked data-structures**
- **Partition program into independent, balanced computations**
- **Avoid adaptive and dynamic computations**
- **Avoid synchronization**
- **Minimize inter-process communication**
- **Rule of thumb: stride 1 + heavy data reuse → performance**

John Feo, Cray Inc.

Throughput Computing

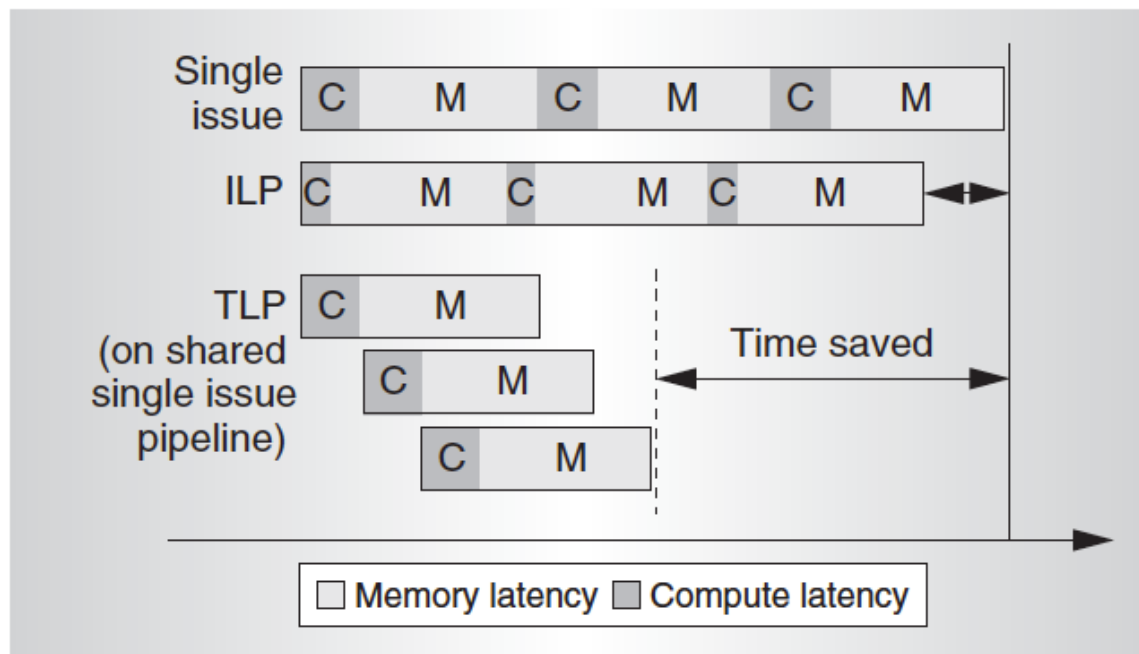
- For a single thread

- memory is the principal obstacle to high performance

- server workloads exhibit poor locality

- exploiting ILP provides only a modest reduction in running time

- conventional ILP processors have low utilization



- With many threads

- can find something to compute every cycle

- significantly higher throughput

- processor utilization is much higher

Figure credit: Niagara: A 32-Way Multithreaded SPARC Processor, P. Kongetira, K. Aingaran, and K. Olukotun, IEEE Micro, pp. 21-29, March-April 2005.

Fine-grain Multithreading

Use thread-level parallelism to hide latencies

- **Multiple active threads per processor**
 - thread = sequential ordered block of > 1 instructions
- **Overlap delays due to long latency operations in one thread with instruction execution of other threads**
 - interleave execution of multiple threads within the pipeline
- **Fine-grain multithreading requires HW support**
 - multiple thread contexts (registers, status word, PC)
 - choose ready instruction to execute from among multiple threads
 - context switch without any delay cycles
 - multiple outstanding memory requests per thread

A Multithreading Thought Question

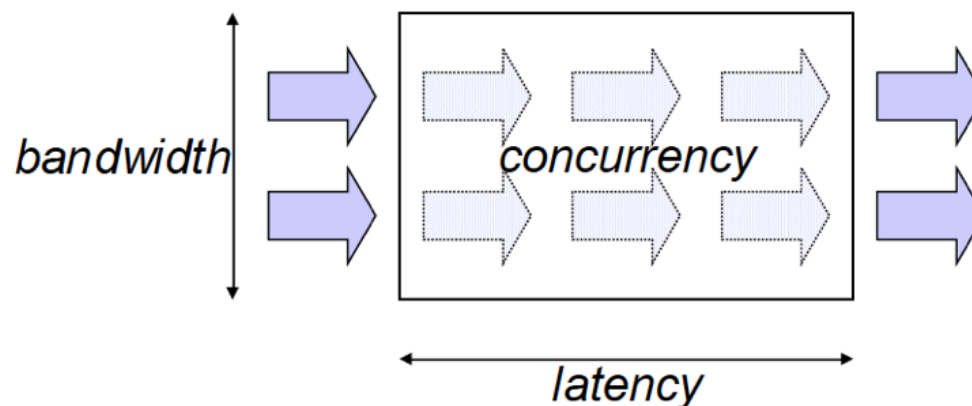
- **Question: what degree of multithreading is necessary to cover memory latency?**
- **Answer: need a sufficient number of memory accesses in flight to cover the bandwidth delay product**

The steady state parallelism required to hide the latency is equivalent to the number of in-flight requests

- In a system that transports objects from input to output without creating or destroying them,

$$\textit{latency} \times \textit{bandwidth} = \textit{concurrency}$$

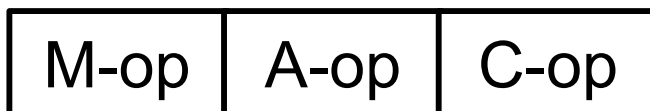
- In queueing theory, this result is known as Little's law.



Cray MTA-2 and Threadstorm

Tera MTA-2 and Cray Threadstorm

- 128 hardware thread streams
- One instruction pipeline
- Switch between instruction streams with no delay
- Instruction streams
 - instruction types
 - a memory operation (M-op)
 - a fused multiply-add (A-op)
 - a branch/logical or add operation (C-op)
 - MTA-2
 - individual instructions
 - explicit distance to next dependent instruction (3 bits max)
max of 8-instructions in flight from one thread
 - Threadstorm “long instruction word”



At least 21 ready threads needed to keep MTA-2 processor fully utilized

Sequence Alignment on the MTA-2. S. Bokhari, J. Sauer. IPDPS 2003, 152.

MTA-2 Processor

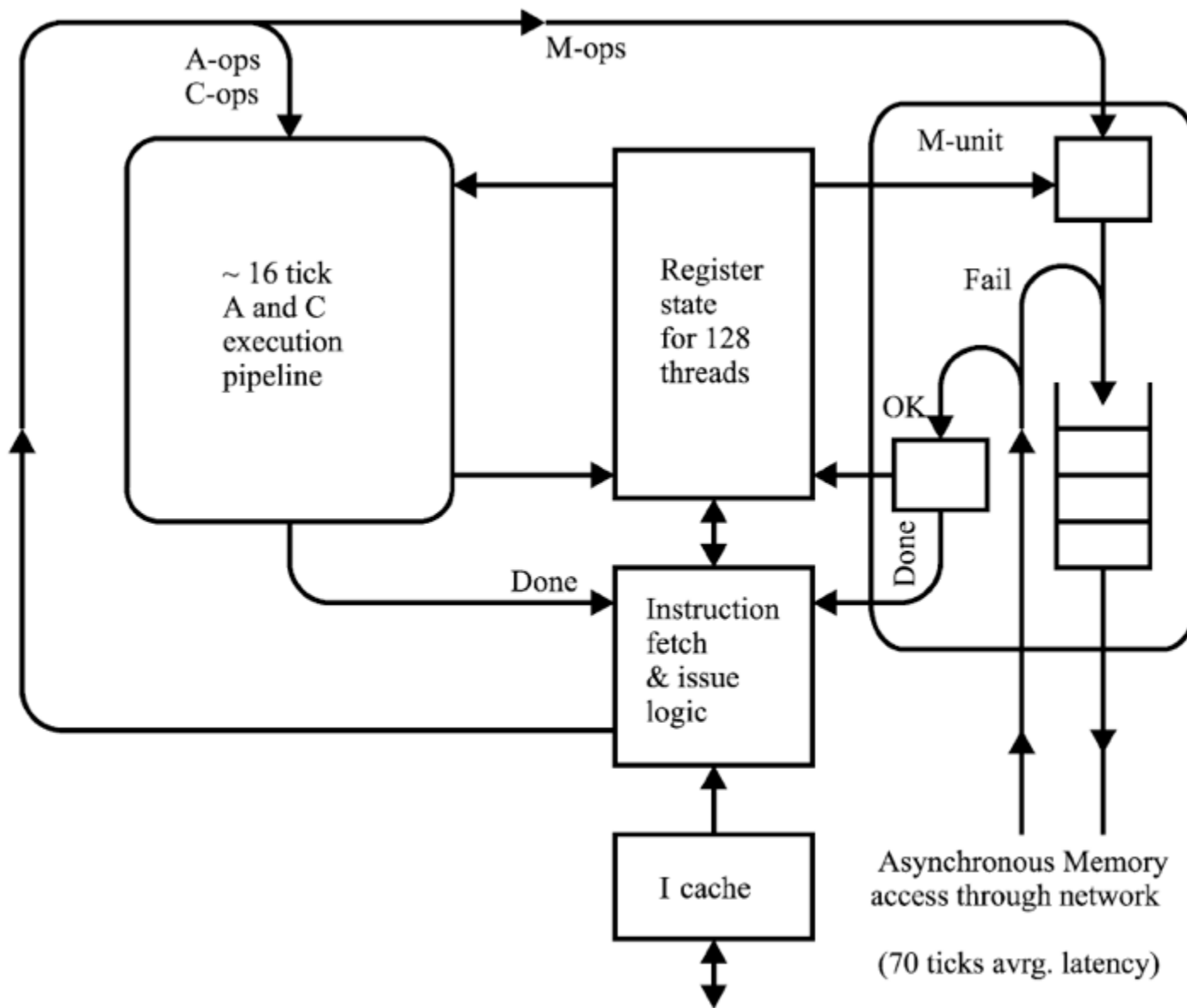


Figure credit: T. Ungerer, B. Robič, J. and Šilc. A survey of processors with explicit multithreading. *ACM Computing Surveys* 35, 1 (Mar. 2003), 29-63.

Threadstorm Processor

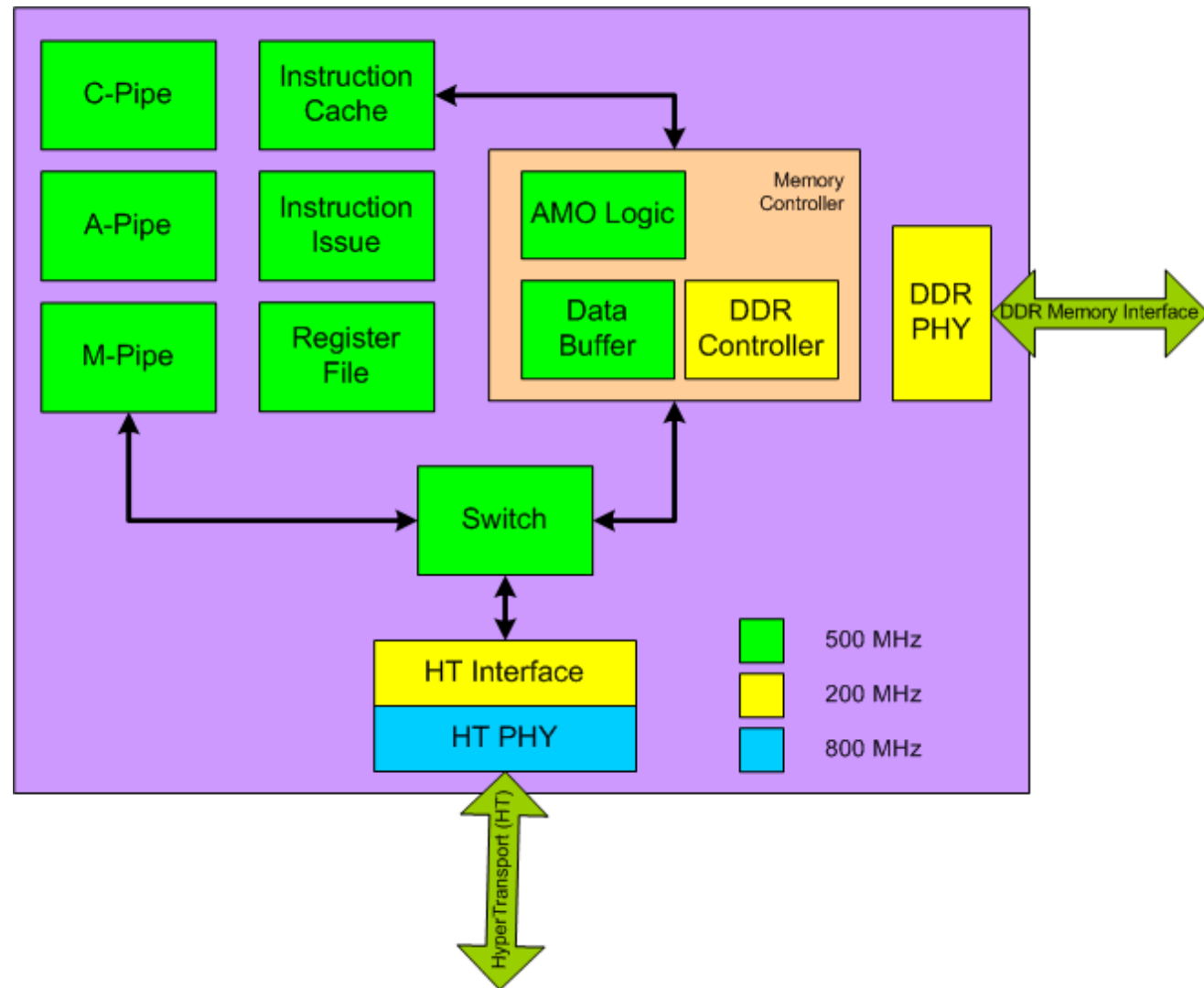
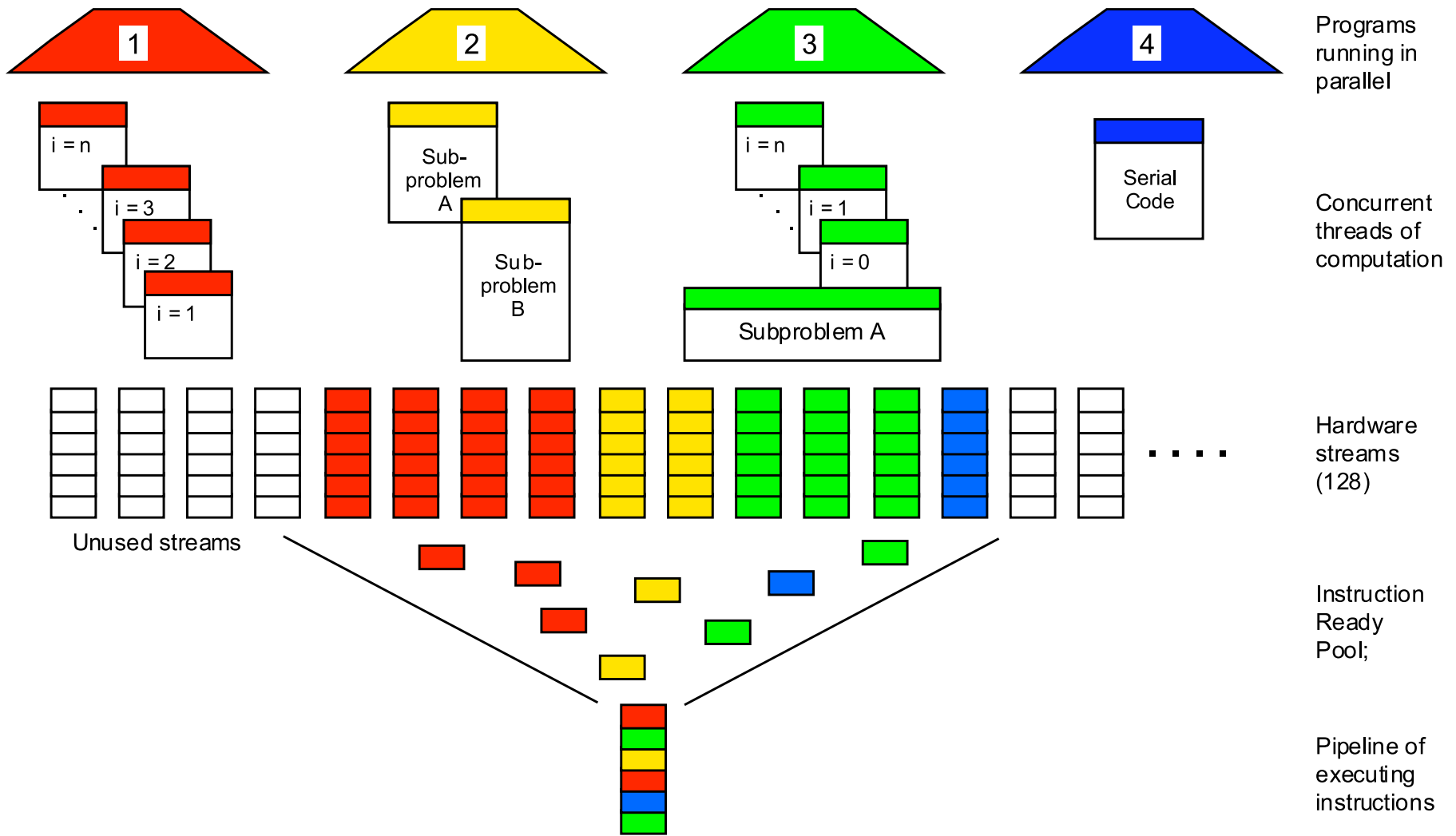


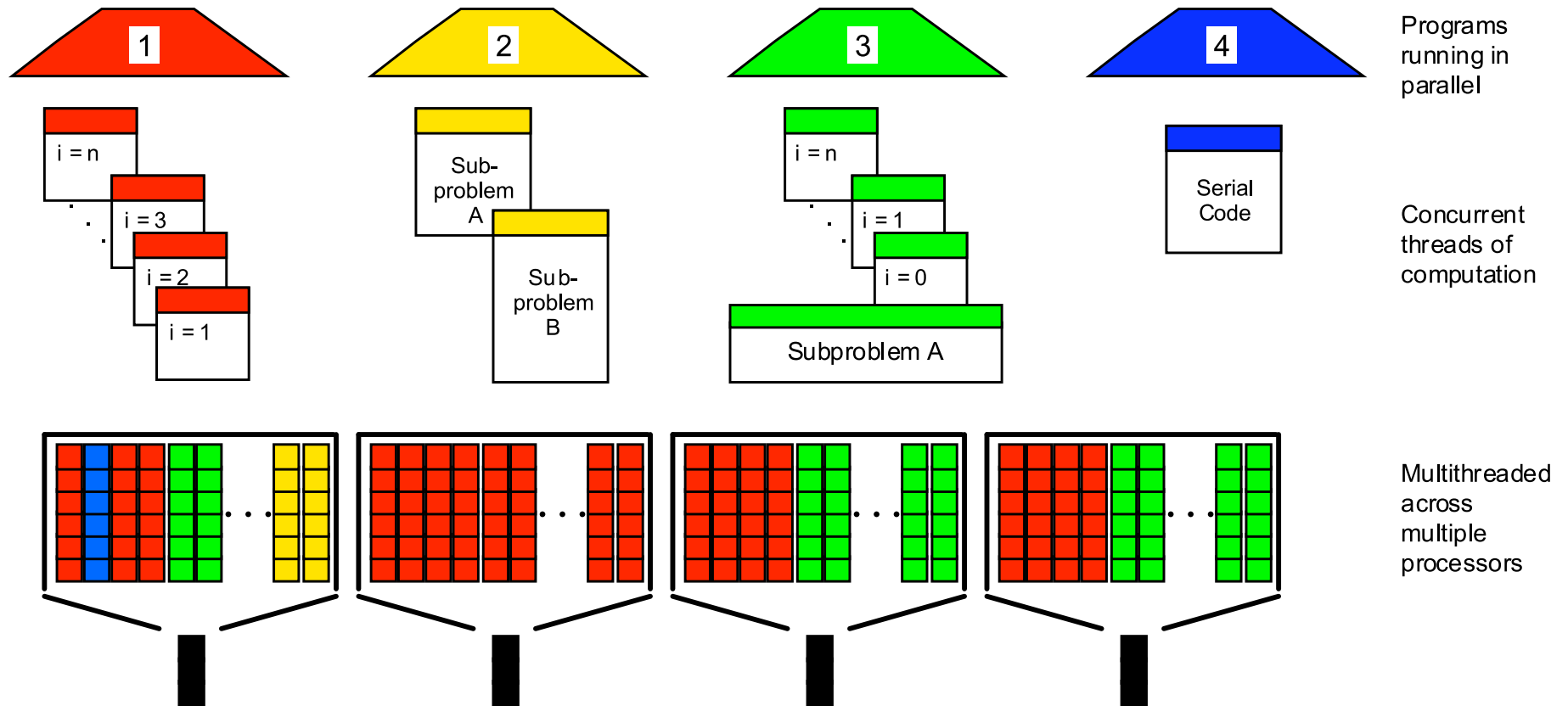
Figure credit: John Feo, Cray Inc.

Threadstorm Processor Logical View

Interleaved multithreading



Cray XMT System Logical View



Cray XMT Memory Subsystem

- **Shared memory**
 - some memory can be reserved as local memory at boot time
 - only compiler and runtime system have access to local memory
 - stack space, register spill area, thread private data
- **Memory module cache**
 - decreases memory latency and increases bandwidth
 - no coherence issues
- **8 word data blocks randomly distributed across the system**
 - eliminates stride sensitivity and hotspots
 - makes programming for data locality impossible
 - block moves to data buffer, but only word moves to processor
- **Full/empty bits on each data word**

Conventional Parallel Programming Wisdom

For high performance ...

- Place data near computation
- Avoid modifying shared data
- Access data in order and reuse
- Avoid indirection and linked data structures
- Partition program into independent, balanced computations
- Avoid adaptive and dynamic computations
- Avoid synchronization
- Minimize inter-process communication
- Rule of thumb: stride 1 + heavy data reuse → performance

Unnecessary on the XMT!

John Feo, Cray Inc.

Cray XMT System Architecture

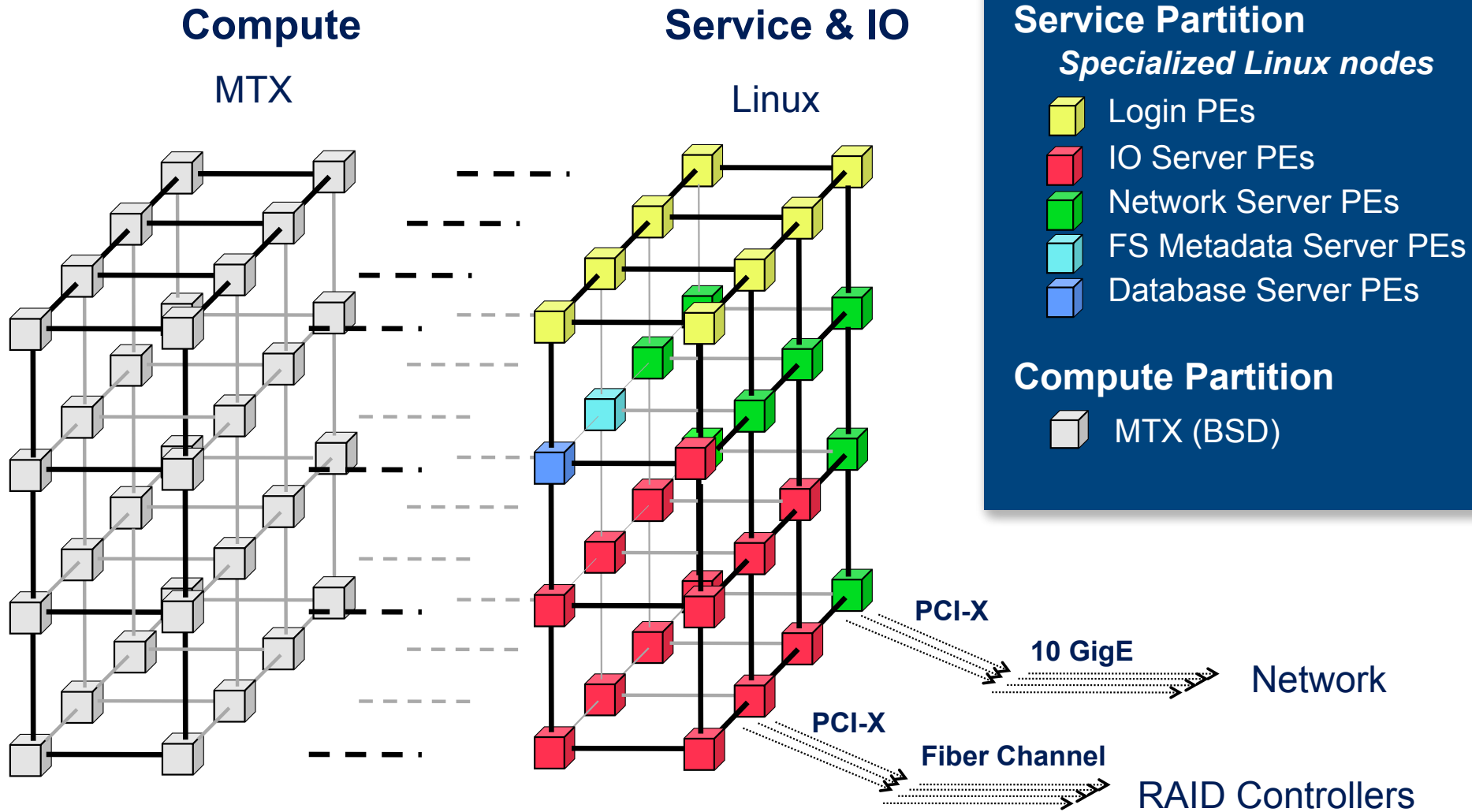


Figure credit: John Feo, Cray Inc.

System Comparison: MTA-2 vs. XMT

	MTA-2	XMT
CPU clock speed	220 MHz	500 MHz
Max system size	256 P	8192 P
Max memory capacity	1 TB (4 GB/P)	128 TB (16 GB/P)
TLB reach	128 GB	128 TB
Network topology	Modified Cayley graph	3D torus
Network bisection bandwidth	$3.5 * P$ GB/s	$15.36 * P^{2/3}$ GB/s
Network injection rate	220 MW/s per processor	Varies with system size

Table courtesy of John Feo, Cray Inc.

Cray XMT Memory Bandwidth Scaling

Example torus configurations	6x12x8	11x12x8	11x12x16	22x12x16	14x24x24
Processors	576	1056	2112	4224	8064
Memory capacity	9 TB	16.5 TB	33 TB	66 TB	126 TB
Sustainable remote memory reference rate (per processor)	60 MW/s	60 MW/s	45 MW/s	33 MW/s	30 MW/s
Sustainable remote memory reference rate (aggregate)	34.6 GW/s	63.4 GW/s	95.0 GW/s	139.4 GW/s	241.9 GW/s
Relative size	1.0	1.8	3.7	7.3	14.0
Relative performance	1.0	1.8	2.8	4.0	7.0

Table courtesy of John Feo, Cray Inc.

Cray Urika-GD Graph Discovery Appliance

Product Brief | Technical Specifications



Cray® Urika-GD™ Technical Specifications

The Urika-GD™ graph analytics appliance designed for data discovery in very large datasets using graph analytics. The Urika appliance is available in a range of different sizes. Contact Cray for specifications of appliances larger than 512 processors (Urika-512).

The Urika-GD appliance consists of:

- Graph analytics platform, providing graph-optimized hardware with shared-memory, multithreading and scalable I/O
- Graph analytics database, providing an RDF triplestore and SPARQL query engine
- Graph analytics application services, providing management, security and data pipeline functions

2012-2014?

GRAPH ANALYTICS PLATFORM				
	Urika-64	Urika-128	Urika-256	Urika-512
Cabinets - Processor	1	2	3	6
Cabinets - Storage	1	1	1	1
Processors				
Threadstorm4 Graph Accelerators with 128 hardware threads per processor	64	128	256	512
x86 Management and I/O	34	68	34	68
Global Shared Memory (TB)	2	4	8	16
External Connectivity				
10 Gigabit Ethernet Ports	1	2	2	2
Gigabit Ethernet ports	8	8	8	8
Infiniband HCAs	Optional	Optional	Optional	Optional
Fibre Channel HBAs	4	4	4	4
Interconnect	Three-dimensional torus interconnect using the Cray Seastar2 communications processor. Each SeaStar2 provides six 7.6 GB/s links to neighbors, and a 6.4 GB/s connection to the x86 or Threadstorm processor. Total interconnect bandwidth scales linearly with increasing system size.			

Sun Niagara-1

Niagara-1 Design Goals

- **Optimize performance of multithreaded commercial workloads**
 - specifically, improve throughput across all threads
- **Hide memory latency**
- **Operate with low power (Niagara-1 T1: 72W typical, 79W peak)**
 - very significant issue for data centers
 - need not met by single-threaded ILP processors
- **Deliver high performance / watt**

Characteristics of Commercial Server Apps

- **High thread-level parallelism**
 - client request parallelism in WWW applications
- **Typically**
 - low instruction-level parallelism
 - except DSS (high), SAP-2T (medium)
 - large working sets
 - except SAP-2T (medium)
 - medium to high data sharing
 - except Web99
- **Data sharing can cause high coherence miss rates**

Niagara-1 at a Glance

- **Threads rather than speed: 32 hardware threads**
 - 4 threads per thread group, 8 thread groups
 - each thread group shares a pipeline
- **SPARC processing pipeline**
 - L1 caches for instructions, data
 - hide memory latency with zero-delay thread switch
- **Shared 3MB L2 cache**
 - 4 banks, pipelined for bandwidth
 - 12-way set associative
- **Crossbar interconnect**
 - links SPARC processing pipelines to L2 cache banks
 - provides 200GB/s bandwidth
 - provides two-entry Q for each src/dest pair
- **Memory: 4 banks, DDR2 memory; > 20GB/s bandwidth**

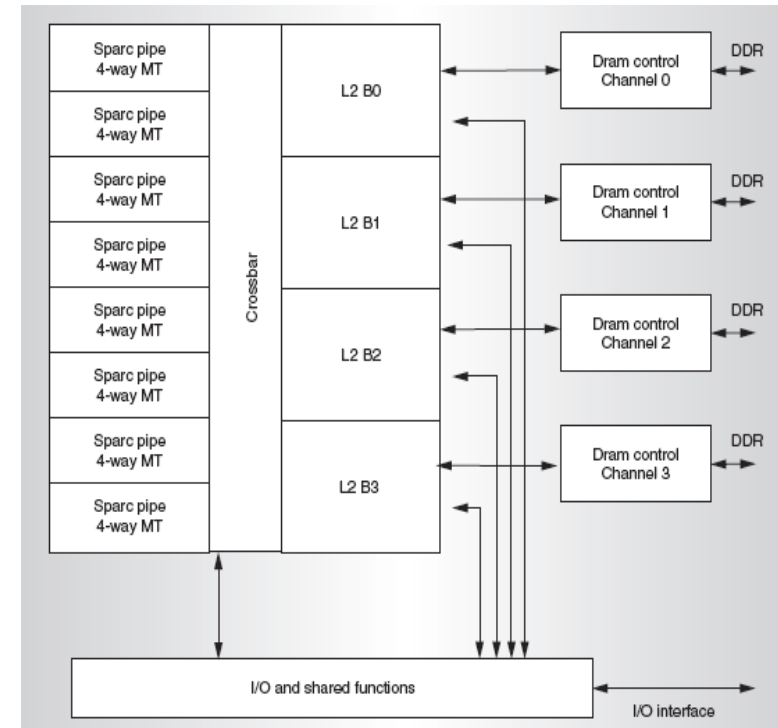


Figure credit: Niagara: A 32-Way Multithreaded SPARC Processor, P. Kongetira, K. Aingaran, and K. Olukotun, IEEE Micro, pp. 21-29, March-April 2005.

SPARC Pipeline Features

- **Single issue, 6-stage pipeline**
- **4 threads in a group share pipeline**
 - private resources
 - registers (register windows)
 - instruction and store buffers
 - shared resources (among thread group)
 - L1 cache
 - TLB
 - exception units
 - most pipeline registers
- **All pipelines share one FP unit on chip**
 - a bottleneck for scientific computing!
(however, the design point for the chip was commercial server applications, not scientific computing)

Niagara-1's SPARC Pipeline

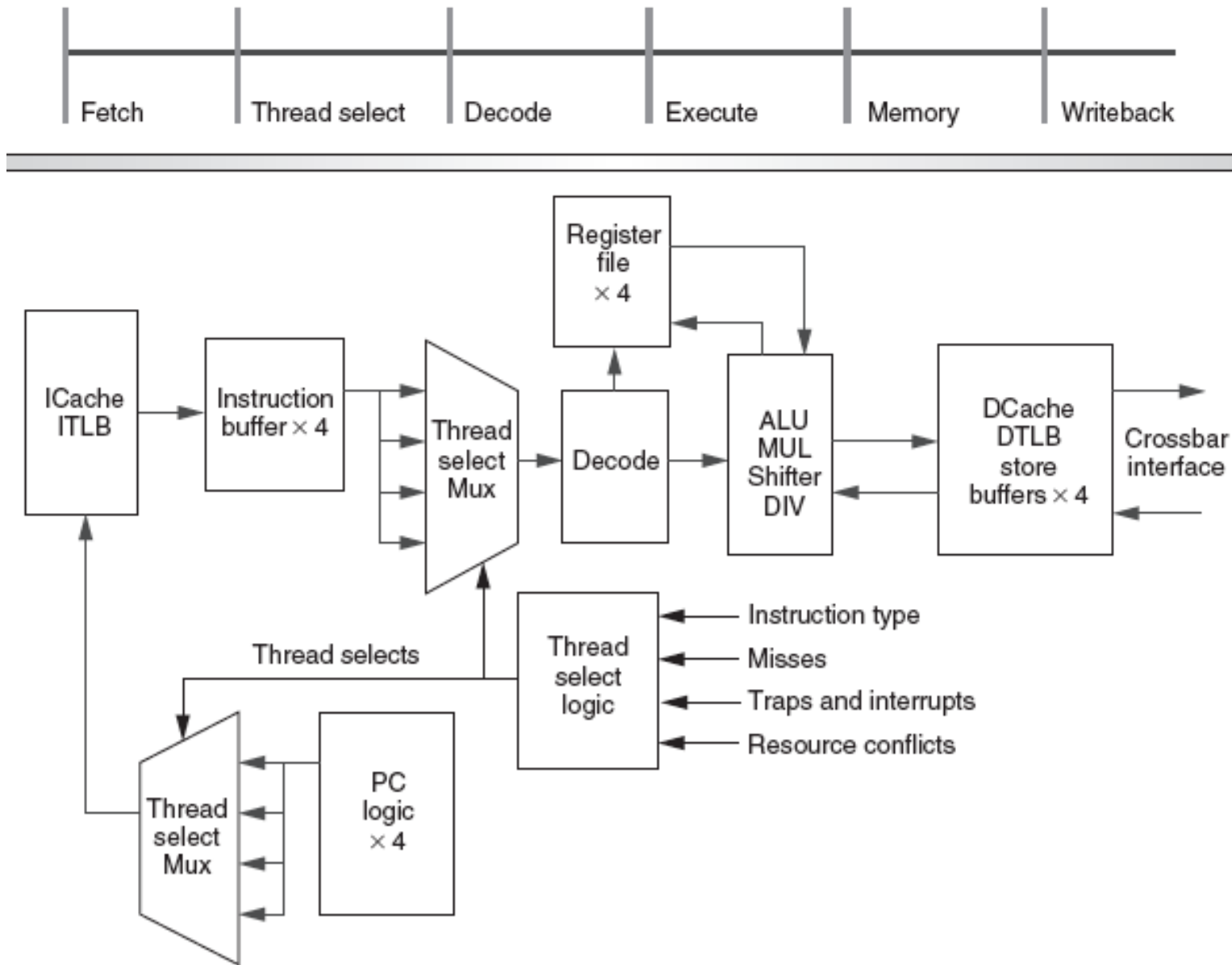



Figure credit: Niagara: A 32-Way Multithreaded SPARC Processor, P. Kongetira, K. Aingaran, and K. Olukotun, IEEE Micro, pp. 21-29, March-April 2005.

Niagara-1's Fine-grain Multithreading

- **Thread select logic decides which thread**
 - fetches an instruction into instruction buffer
 - issues an instruction to decode stage } same thread
- **Selection policy**
 - typically, switch threads every cycle
 - favor least-recently-executed thread
 - scheduler assumes cache hits
 - speculatively issue next instruction (but with lower priority)
- **Select the next thread using info from various stages**
 - instruction type (e.g. deselect successor to load)
 - miss
 - trap or interrupt
 - resource conflicts (e.g. FPU, division)

Multithreading & Pipelining (Ideal)

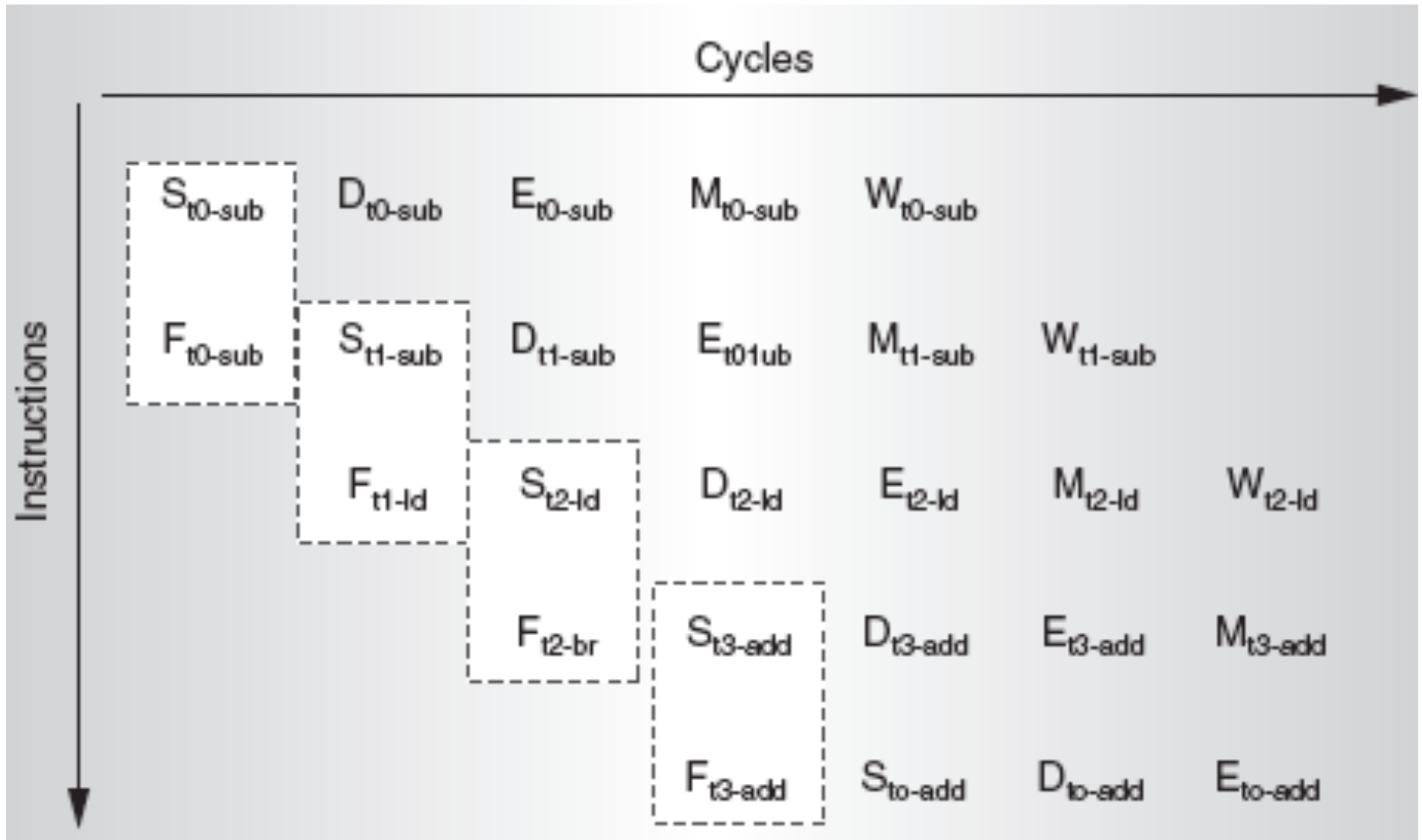


Figure credit: Niagara: A 32-Way Multithreaded SPARC Processor, P. Kongetira, K. Aingaran, and K. Olukotun, IEEE Micro, pp. 21-29, March-April 2005.

Handling Interlocks in Niagara-1

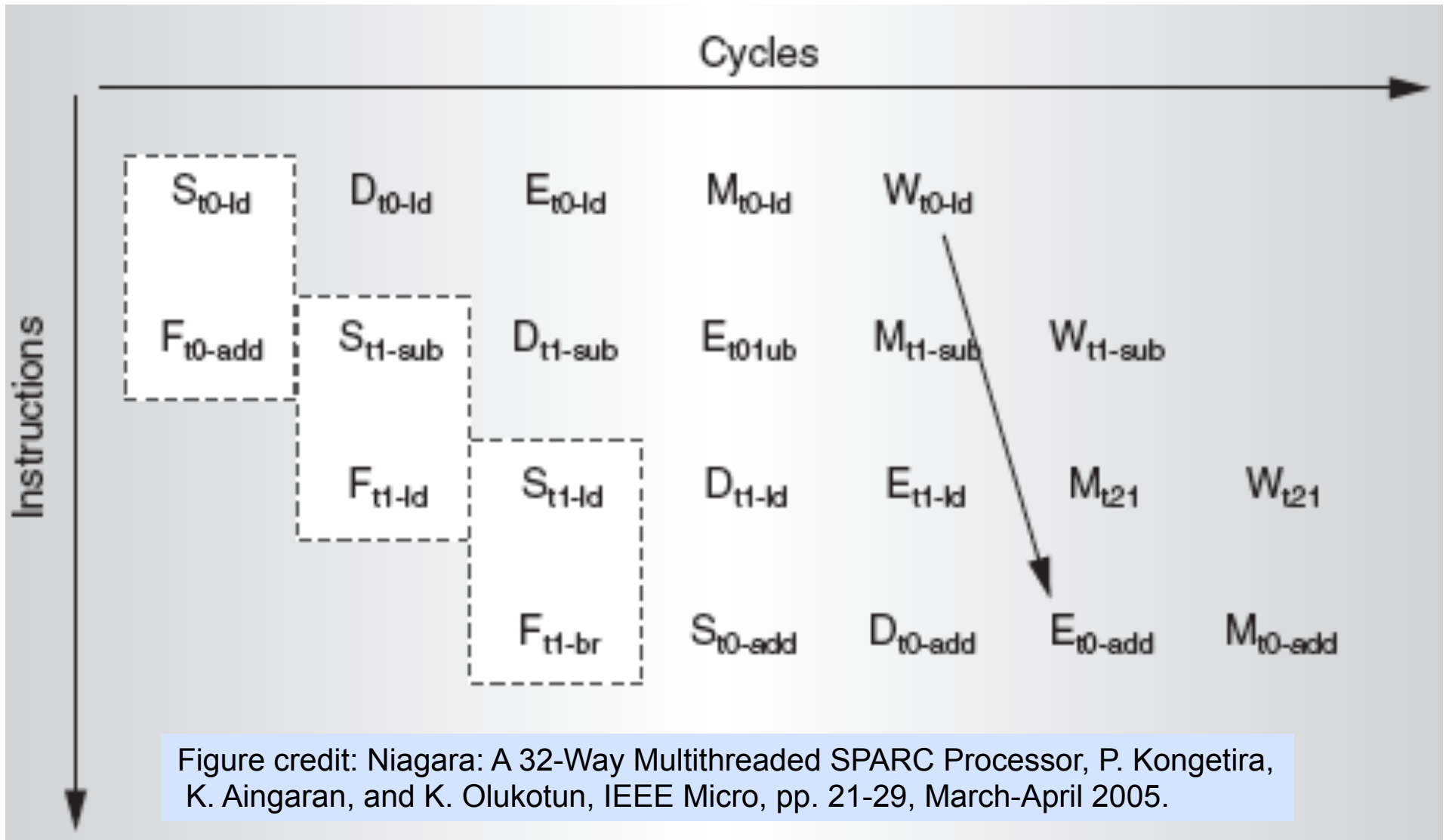
- **Loads**

- 3 cycle latency
- stall thread until hazard clears

- **Divider**

- ALU throughput = 1/cycle; divider = $< 1/\text{cycle}$
- implication: any thread executing DIV may need to wait
- scheduling
 - priority given to least recently executed thread
 - while divider or FPU in use, other threads can use other resources

2 Threads + Hazard



On load miss: flush add, reissue when value arrives from L2

Niagara-1 Memory Subsystem

- **L1 Icache**

- 16KB, 4-way set associative, 32-byte lines, 3 cycle latency
- random replacement: less area
- ifetch: two instructions/cycle

- **L1 Dcache**

- 8KB, 4-way set associative, 16-byte lines, write through
- reduce avg access time: miss rates ~ 10%
- why not larger?
 - commercial server applications need much larger caches for < miss rates
tradeoff not favorable to area
- 4 threads compensate for higher miss rates (hide latency)
- states: valid, invalid
- stores do not update L1 cache until update L2: global visibility in L2

- **L2 shared cache**

- 3MB, 4 banks, 12-way set associative, pipelined for bandwidth
- shadows L1 tags in a directory
- deliver load miss and line being replaced to L2 at same time
- copy-back policy

Using Niagara-1

- **Appears as if 32 processors to OS**
- **Expect multi-threaded SMP applications to benefit**
 - fast data sharing in L2 rather than using SMP bus
- **Simple pipeline**
 - no special instruction scheduling necessary

Niagara-1 vs. Niagara-2

- **Processor core**
 - 32 vs. 64 threads (doubled number of threads per core)
 - double number of execution units per core (2 vs. 1)
 - new pipeline stage “pick”
 - pick 2 of 8 threads to issue in a cycle
- **Memory hierarchy**
 - doubled set associativity of L1 from 4 to 8
 - doubled the number of L2 cache banks from 4 to 8
 - now 1 per core: boosts performance ~18% over just 4 banks
 - from 12-way to 16-way set associative
- **Floating point**
 - one per core rather than one per chip
 - turned Niagara2 into a very respectable chip for scientific computing

Performance Comparisons

Niagara-1 Performance

- **Claim almost linear scaling on commercial workloads**
- **Peak and average load expected to be similar**
- **Performance modeled at 1, 1.5 and 2GHz**
 - clock rate had a minimal impact on performance¹**

Sun's Niagara falls neatly into multithreaded place.
Charlie Demerjian, The Inquirer, 02 November 2004,

Niagara-1 vs. Opteron Performance

Central Server	HP ProLiant DL385	Sun Fire Model T2000
Operating System	Windows Server 2003	Solaris 10
SAP Release	SAP R/3® Enterprise 4.7	mySAP™ ERP 2004
Database	SQL Server 2000	MaxDB 7.5
Certification Number	2005026	2005047
Processor Type	Dual-Core Opteron 2.2 GHz	UltraSPARC T1 1.2 GHz
Processor/Cores/Threads	2/4/4	1/8/32
Configured Memory	16 GB	32 GB
Form Factor (Rack Units)	2U	2U
Calculated System Power (Watts) ²	388	376
Number of SAP SD Benchmark Users	983	950
Number of SAP SD Benchmark Users/Watt	2.5	2.5

The Real Story about Sun's CoolThreads (aka Niagara)
<http://h71028.www7.hp.com/ERC/cache/280124-0-0-0-121.html>

Stream Benchmarks: MTA-2 vs. Niagara

- **Copy:** $a(i) = b(i)$
- **Scale:** $a(i) = s * b(i)$
- **Add:** $a(i) = b(i) + c(i)$
- **Triad:** $a(i) = b(i) + s * c(i)$

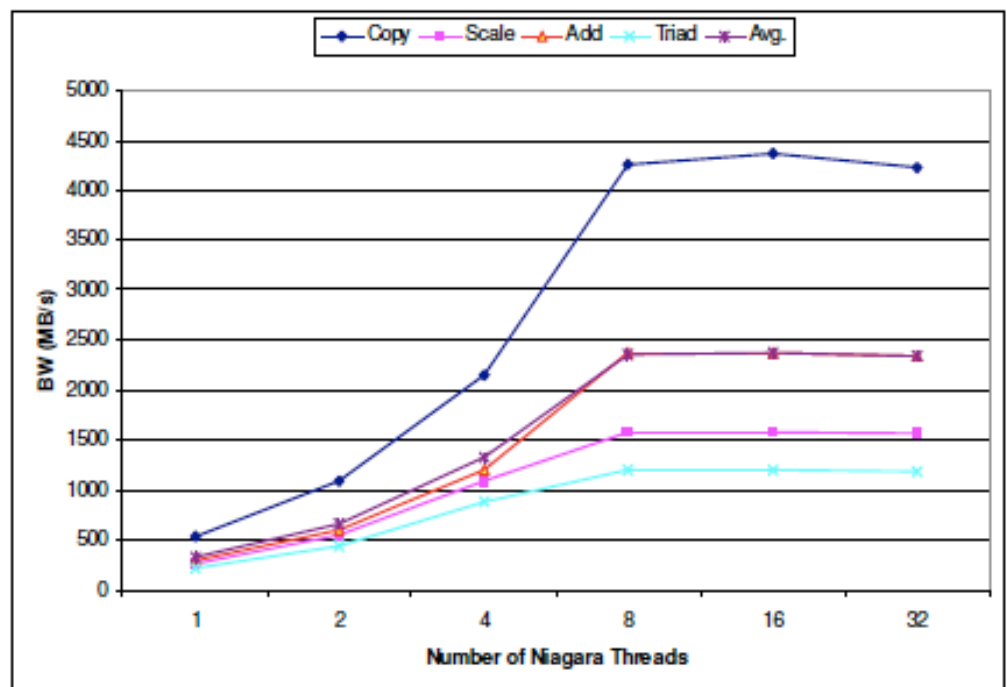
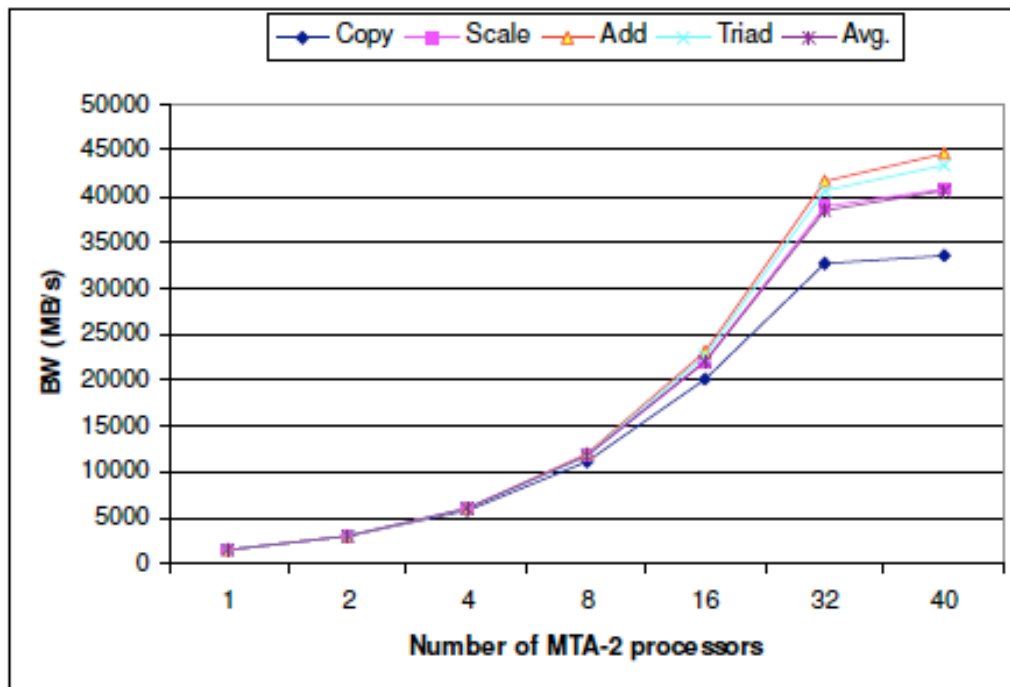
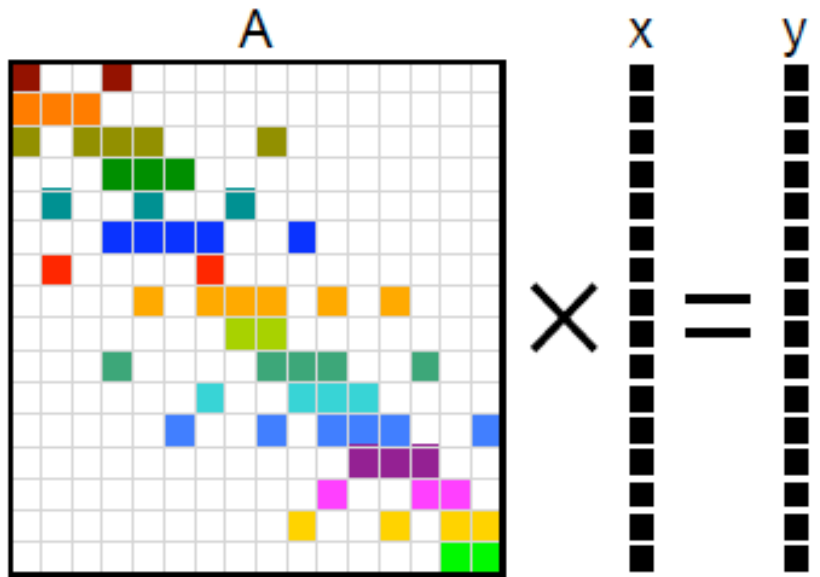


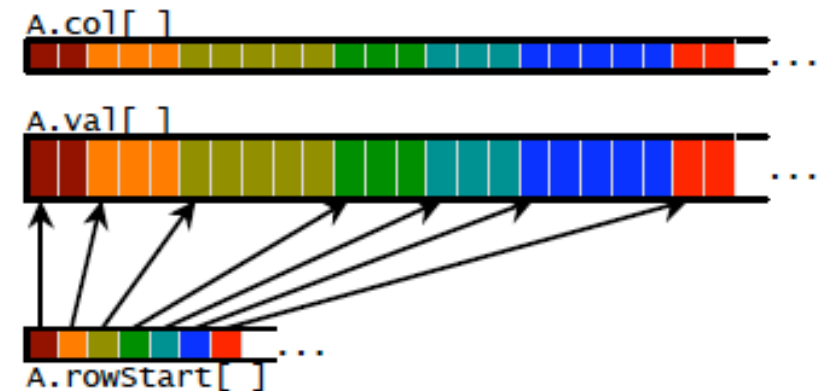
Figure credit: Evaluating the Potential for Multithreaded Platforms for Irregular Scientific Computations, Nieplocha et al. CF'05, Ischia, Italy.

Sparse Matrix Vector Multiply



(a)

algebra conceptualization



(b)

CSR data structure

```
for (r=0; r<A.rows; r++) {  
    double y0 = 0.0;  
    for (i=A.rowStart[r]; i<A.rowStart[r+1]; i++){  
        y0 += A.val[i] * x[A.col[i]];  
    }  
    y[r] = y0;  
}
```

(c)

CSR reference code

Niagara2 vs. Commodity Processors

Sparse matrix vector multiply

Core Architecture	AMD Opteron X2	Intel Clovertown	Sun Niagara2
Type	super scalar out of order	super scalar out of order	MT dual issue*
Clock (GHz)	2.2	2.3	1.4
L1 DCache	64KB	32KB	8KB
Local Store	—	—	—
DP flops/cycle	2	4	1
DP GFlop/s	4.4	9.33	1.4

System	Opteron X2	Clovertown	Niagara2
# Sockets	2	2	1
Cores/Socket	2	4	8
L2 cache	4MB (1MB/core)	16MB (4MB/2cores)	4MB (shared)
DP GFlop/s	17.6	74.7	11.2
DRAM Type	DDR2 667 MHz 2×128b	FBDIMM 667 MHz 4×64b	FBDIMM 667 MHz 4×128b
DRAM (read GB/s)	21.3	21.3	42.6
Ratio Flop:Byte	0.83	3.52	0.26
Max Socket Pwr (Watts)	190	160	84
Sustained Sys Pwr (Watts)	230	330	350

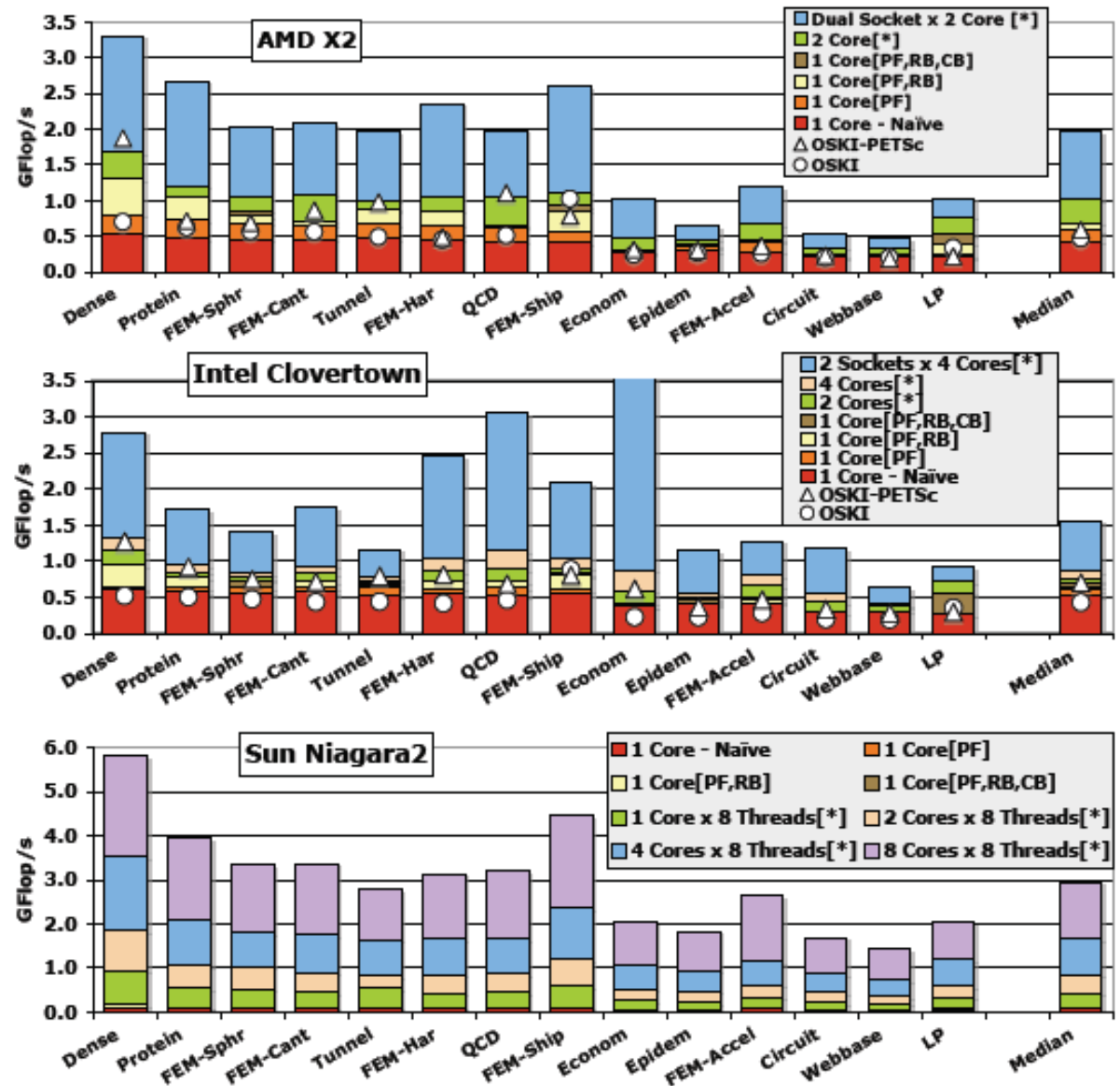


Figure credit: S. Williams et al. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms, SC2007.

Two More Benchmarks: Stencil & LBMHD

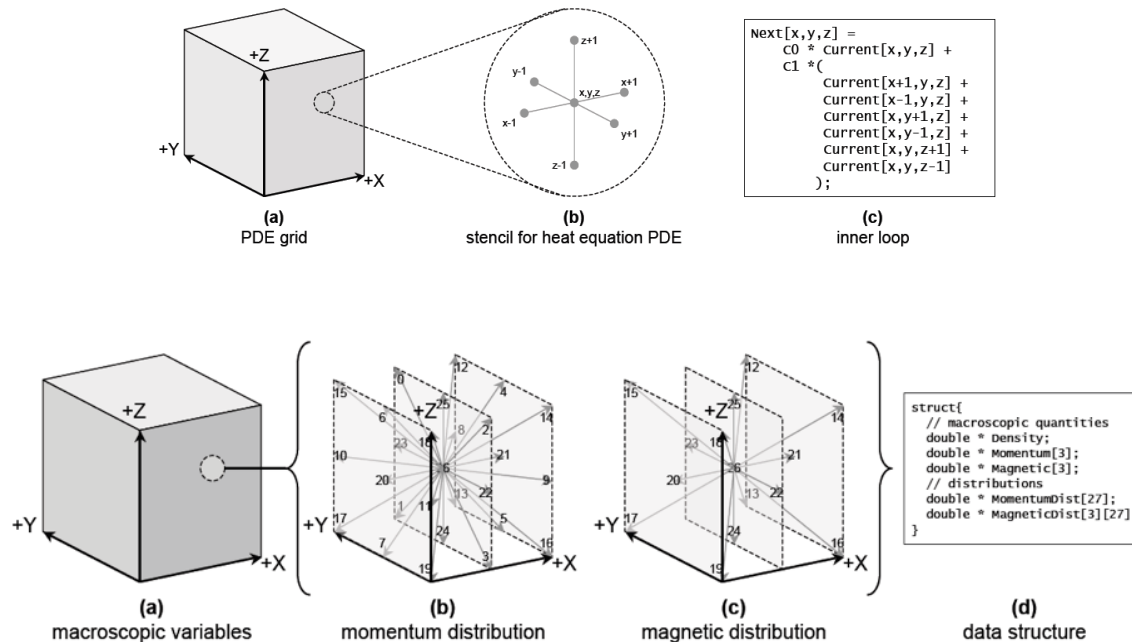
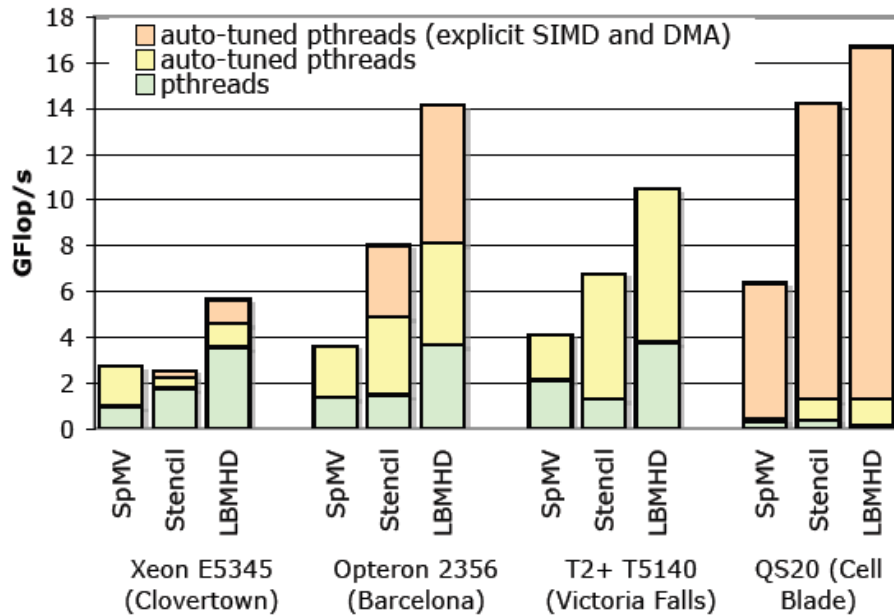


Figure 4. Visualization of the datastructures associated with LBMHD: (a) the 3D macroscopic grid, (b) the D3Q27 momentum scalar velocities, (c) D3Q15 magnetic vector velocities, and (d) C structure of arrays datastructure. Note, each pointer refers to a N^3 grid, and X is the unit stride dimension.

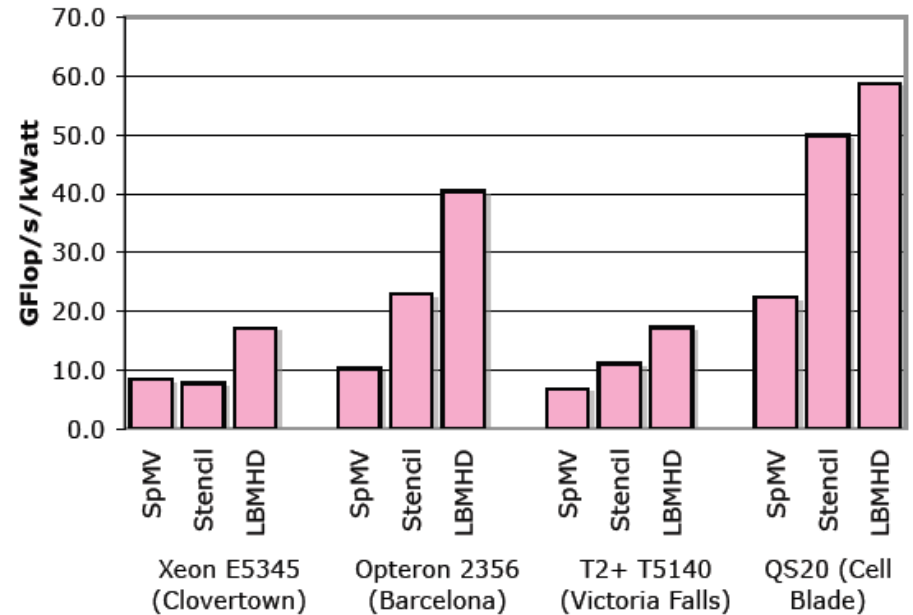
The code is far too complex to duplicate here, although a conceptualization of the lattice method and the data structure itself is shown in figure 4. Nevertheless, the *collision()* operator must read 73 doubles, write 79 doubles, and perform 1300 floating-point operations per lattice update. This results in a compulsory limited arithmetic intensity of about 0.7 on write allocate architectures.

Figure credit: S. Williams et al. PERI - Auto-tuning memory-intensive kernels for multicore. Journal of Physics: Conference Series 125 (2008) 012038

Niagara 2 vs. Commodity Processors



(a) Performance



(b) Power Efficiency

System	Xeon E5345 (Clovertown)	Opteron 2356 (Barcelona)	UltraSparc T5140 T2+ (Victoria Falls)	QS20 Cell Blade	
# Sockets	2	2	2	2	
Cores/Socket	4	4	8	1	8
shared L2/L3 Cache	4×4MB(shared by 2)	2×2MB(shared by 4)	2×4MB(shared by 8)	–	–
DP GFlop/s	74.66	73.6	18.7	12.8	29
DRAM Bandwidth (GB/s)	21.33(read) 10.66(write)	21.33	42.66(read) 21.33(write)	51.2	
DP Flop:Byte Ratio	2.33	3.45	0.29	0.25	0.57
System Power (Watts) [§]	330	350	610	285 [†]	
Threading	Pthreads	Pthreads	Pthreads	Pthreads	libspe2.1
Compiler	icc 10.0	gcc 4.1.2	gcc 4.0.4	xlc 8.2	xlc 8.2

Figure credit: S. Williams et al. PERI - Auto-tuning memory-intensive kernels for multicore. Journal of Physics: Conference Series 125 (2008) 012038

Benchmarks - I

Power system state estimation problem

—conjugate gradient solver

—key kernel: sparse-matrix-vector product

```
do i = 1, N
  t = 0.0
  C$MTA loop serial
    do j = irow(i), irow(i + 1) - 1
      t = t + a(j) * x(icol(j))
    end do
  r(i) = t
end do
```

—parallelization for the MTA: almost entirely automatic

—parallelization for Niagara: OpenMP directives for loops

– parallel loop, reduction clause, data scoping (shared vs. private)

Power System State Estimation

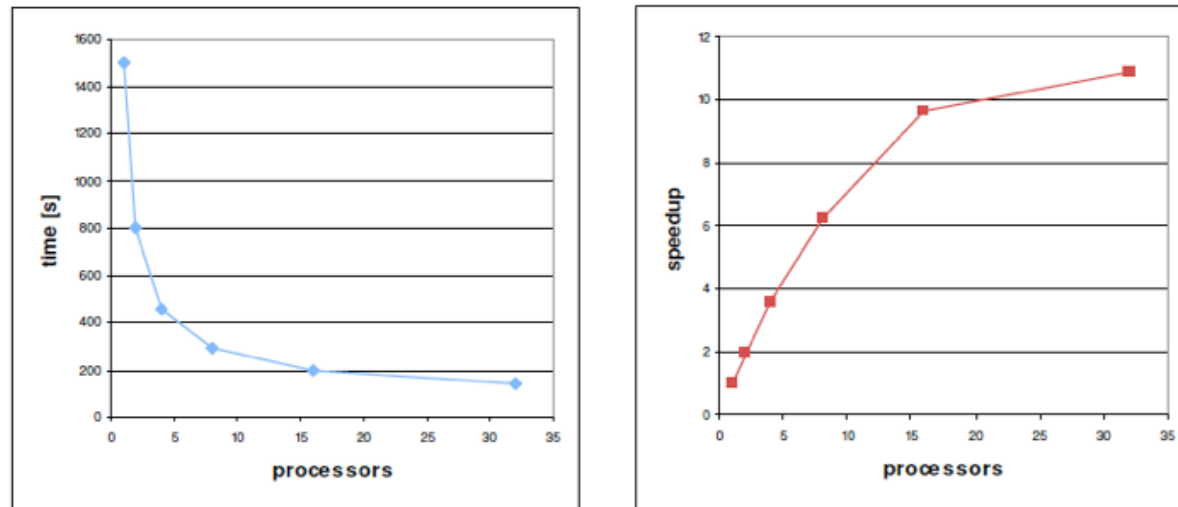


Figure 11: Wall clock time (left) and speedup (right) for PSE on the Cray MTA-2

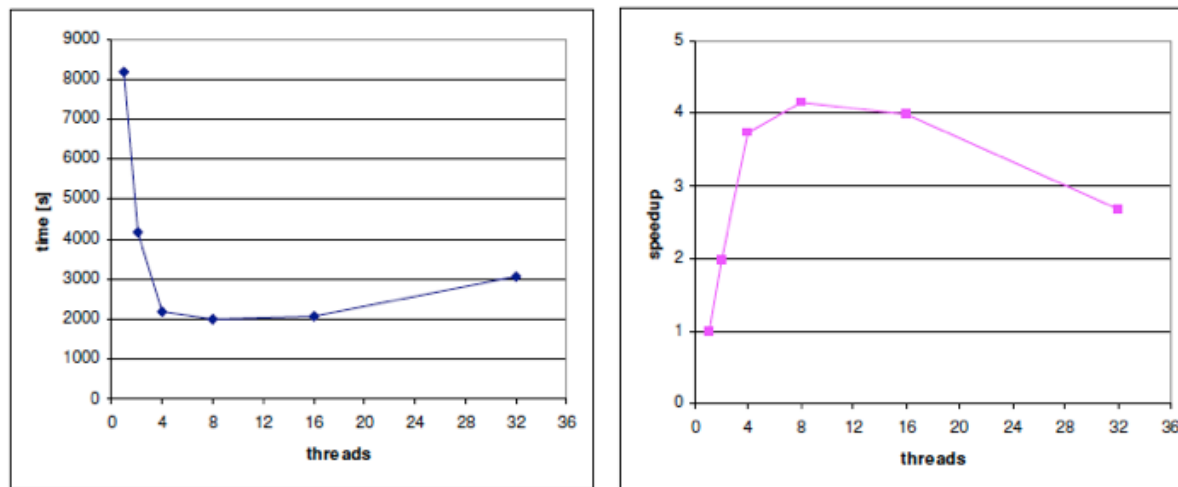


Figure 12: Wall clock time (left) and speedup (right) for PSE on the Sun Niagara

- **Is this a fair comparison?**
- **How might we improve it?**

Benchmarks - 2

Anomaly detection for categorical data

- traffic analysis of categorical data using a partial-dimension tree
- key computation: insert into linked list of a node's children
- parallelization for the MTA: full-empty bit synchronization

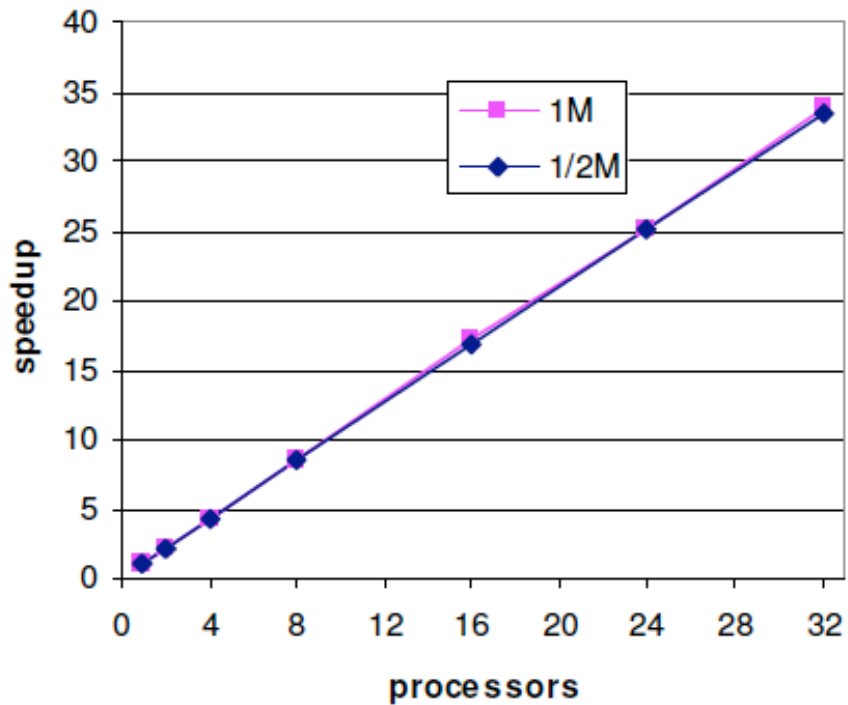
```
while true {
  ptr = node.next
  if ptr is null
    ptr = readfe(node.next)
    if ptr is not null then continue
    ptr = memory for new node
    initialize new node
    writeef(node.next, ptr)
    break
  else if next node is the one I want
    increment counter
    writeef(node.next, ptr)
    break
  else
    writeef(node.next, ptr)
    node = ptr
  end if
} end while
```

—parallelization for Niagara: OpenMP + hash table of locks

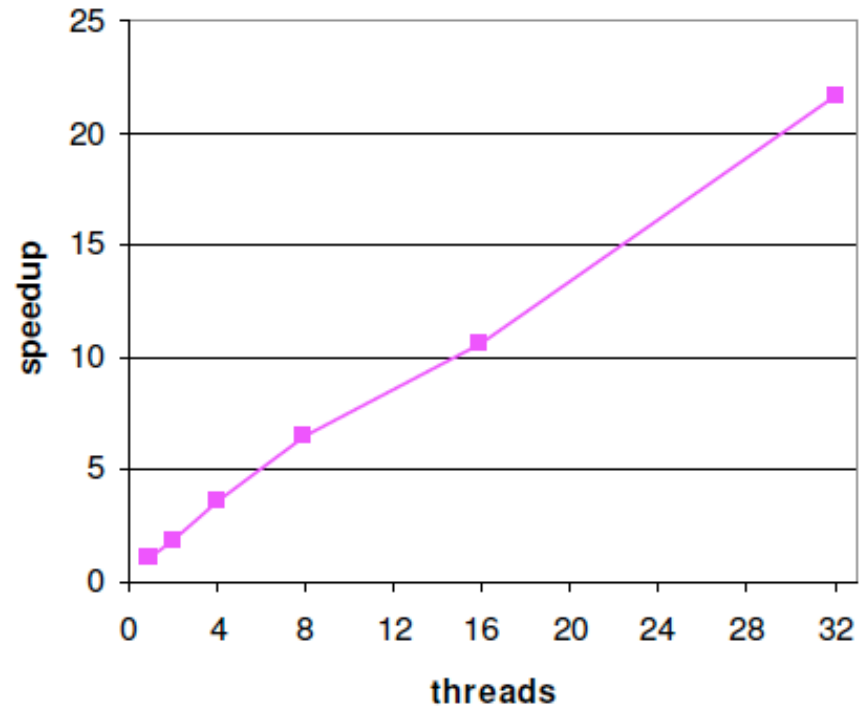
- hash pointer addresses onto locks for fine-grain synchronization

drawback: unrelated operations might contend for a lock

Anomaly Detection Speedup



MTA-2



Niagara-1

Betweenness Centrality

- Graph $G=(V,E)$
- Let σ_{st} denote the number of shortest paths between vertices s and t .
- Let $\sigma_{st}(v)$ be the count that pass through a specified vertex v
- Betweenness centrality of v is defined as

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

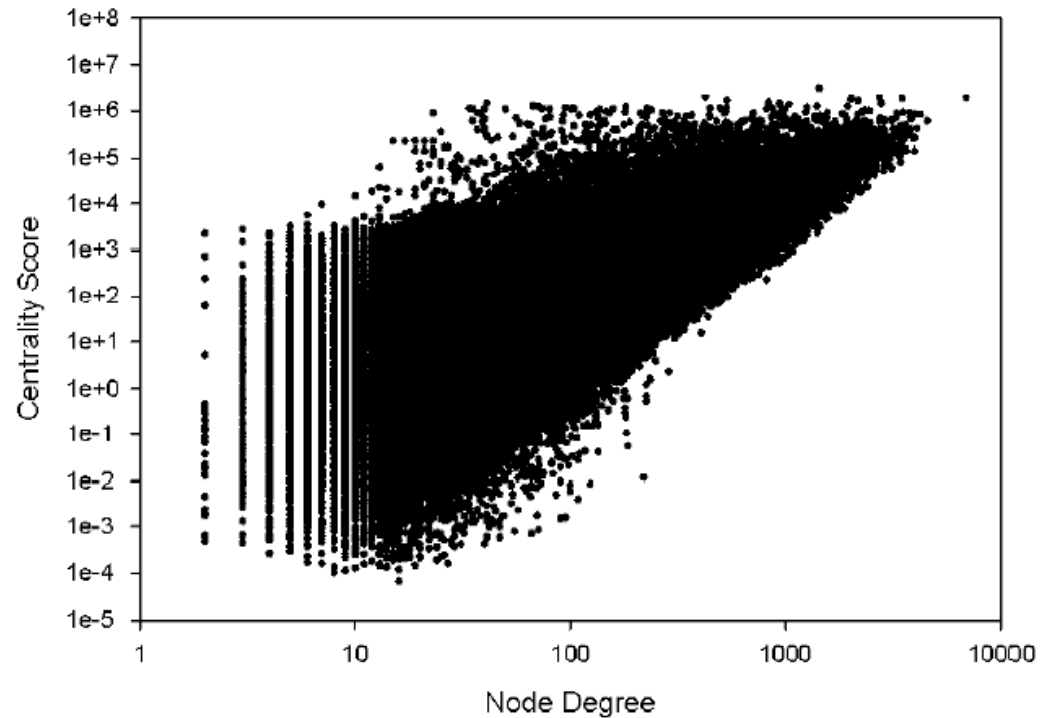


Figure 4. Centrality analysis (Node degree vs. approximate betweenness value) of the IBDb movie actor data set (1.54 million vertices and 78 million edges). Vertices represent actors, and edges correspond to actors co-starring in movies.

Figure credit: A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets, K. Madduri et al. IPDPS, 2009, pp.1-8.

Betweenness Centrality

Table 3. Performance of the SSCA#2 betweenness centrality kernel for a graph of SCALE 24 on the Cray XMT and the Cray MTA-2.

System/Configuration	TEPS rate (Millions of edges per second)
XMT, 1 processor	15.33
XMT, 16 processors	160.00
MTA-2, 1 processor	10.39
MTA-2, 16 processors	160.16
MTA-2, 40 processors	353.53

- 1PE: XMT 47% faster than MTA-2
- 16 nodes: comparable
- MTA-2 modified Cayley graph network scales much better than XMT torus

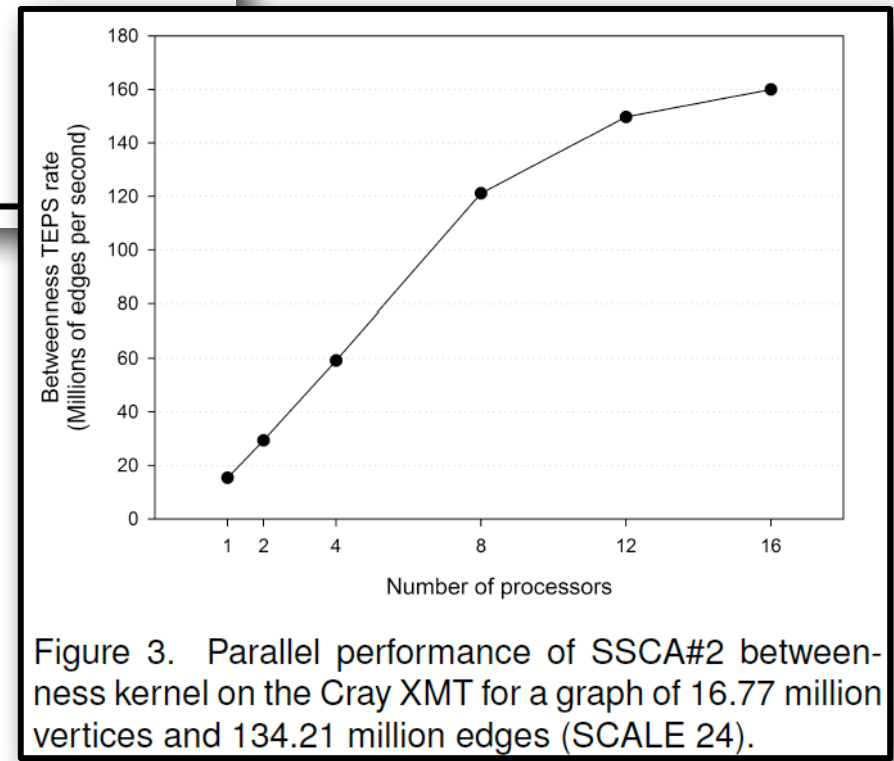


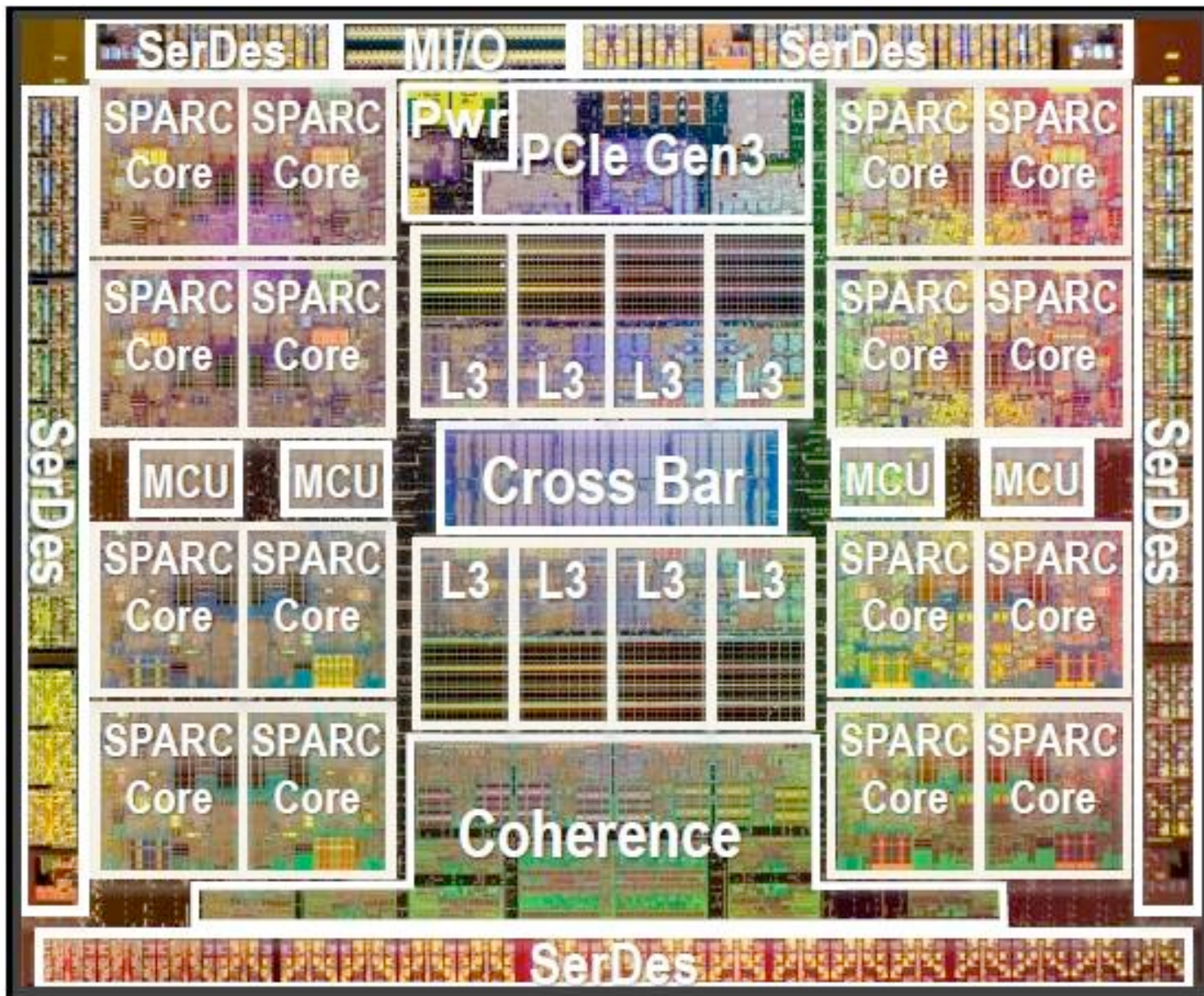
Figure credit: A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets, K. Madduri et al. IPDPS, 2009, pp.1-8.

Chip Multi Threading (CMT)

Oracle SPARC T5

(March 2013)

SPARC T5



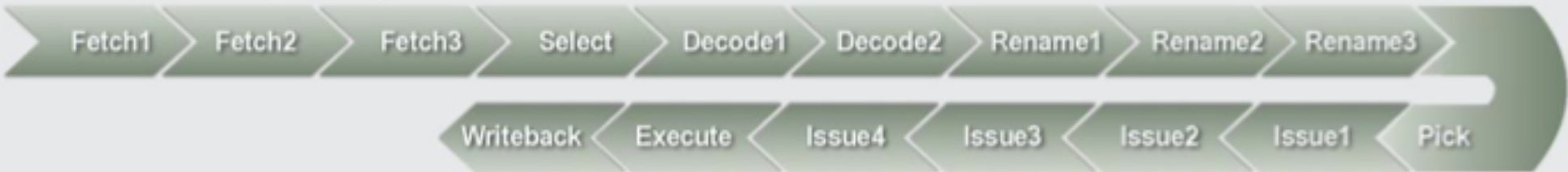
Cross Bar Bisection B/W = 1TB/s

SPARC T3 to T5

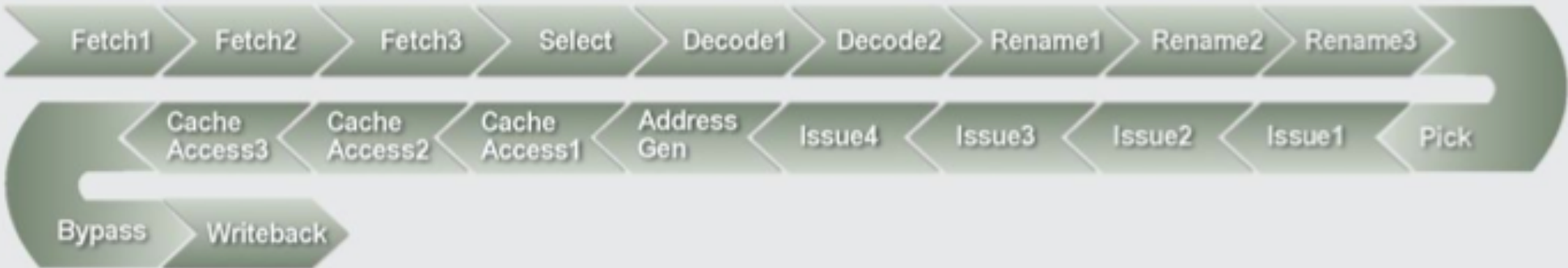
Feature	T5	T4	T3
Frequency	3.6 GHz	3 GHz	1.65 GHz
O-o-O	Yes	Yes	No
Dual Issue	Yes	Yes	No
I/D Prefetch	Yes	Yes	No
Cores	16	8	Up to 16
Threads/Core	8	8	8
Sockets	1,2,4,8	1,2,4	1,2,4
Caches	L1: 16KI, 16KD L2: 128K L3: 8MB (8 banks, 16 way)	L1: 16KI, 16KD L2: 128K L3: 4M (8 banks, 16 way)	L1: 16KI, 8KD L2: 6MB (16 banks, 24 way)
Functional Units/core	1 FPU, 2 Integer Crypto (14 ciphers)	1 FPU, 2 Integer Crypto (14 ciphers)	1 FPU, 2 Integer Crypto (12 ciphers)
Coherency Switch	7 x 153.6Gb/s	6 x 9.6Gb/s	6 x 9.6Gb/s

SPARC T5 Pipelines

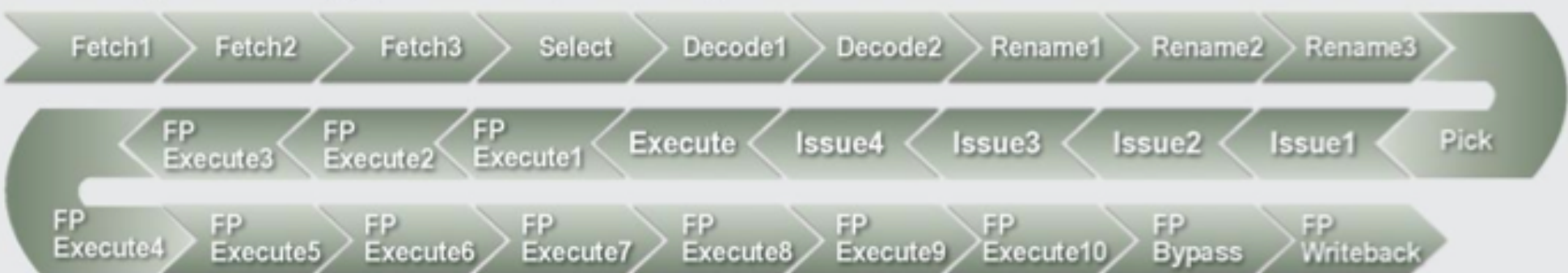
16-Stage Integer Pipeline



20-Stage Load-Store Pipeline



27-Stage Floating-point Graphics Pipeline



- Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture. Oracle White Paper. July 2013.

SPARC T5 Multithreading

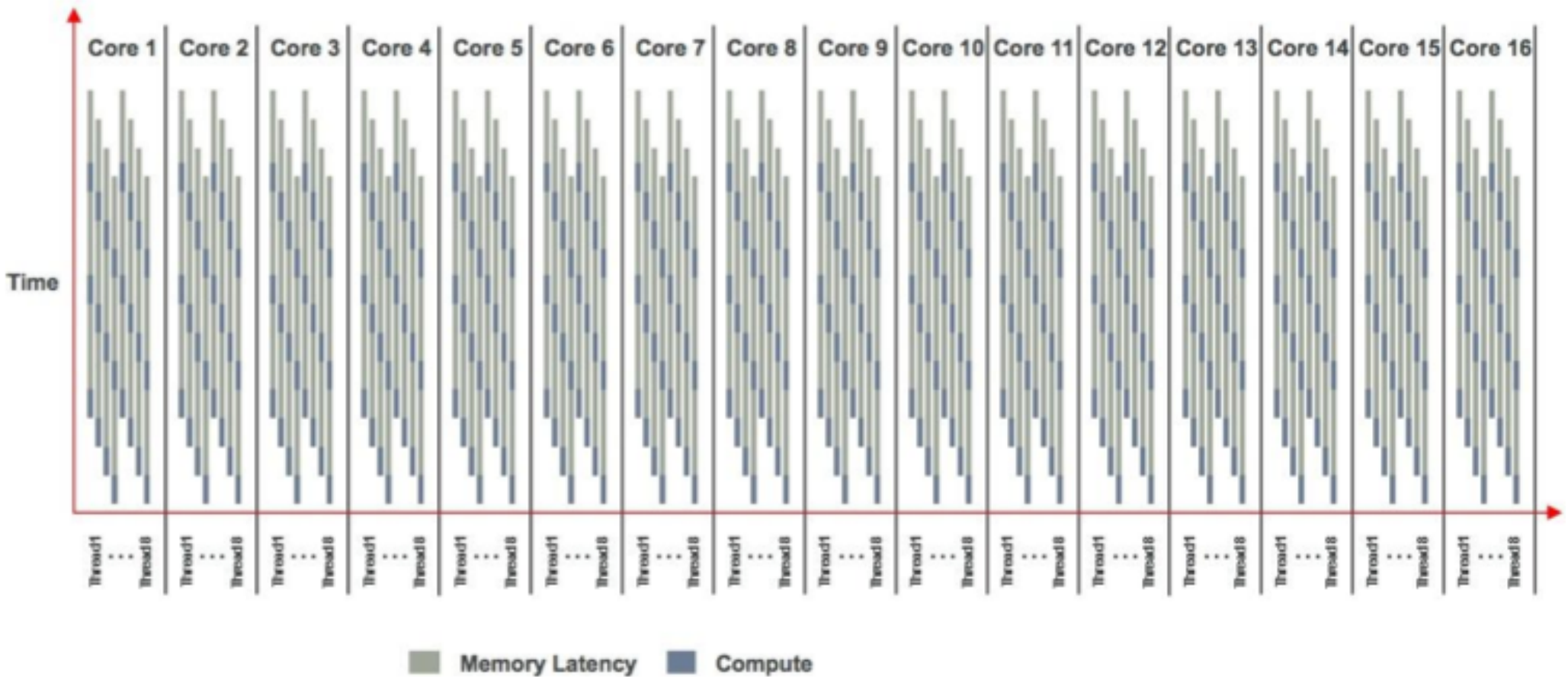


Figure 3. A single 16-core SPARC T5 processor supports up to 128 threads, with up to two threads running in each core simultaneously.

- Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture. Oracle White Paper. July 2013.

SPARC T5 Processor

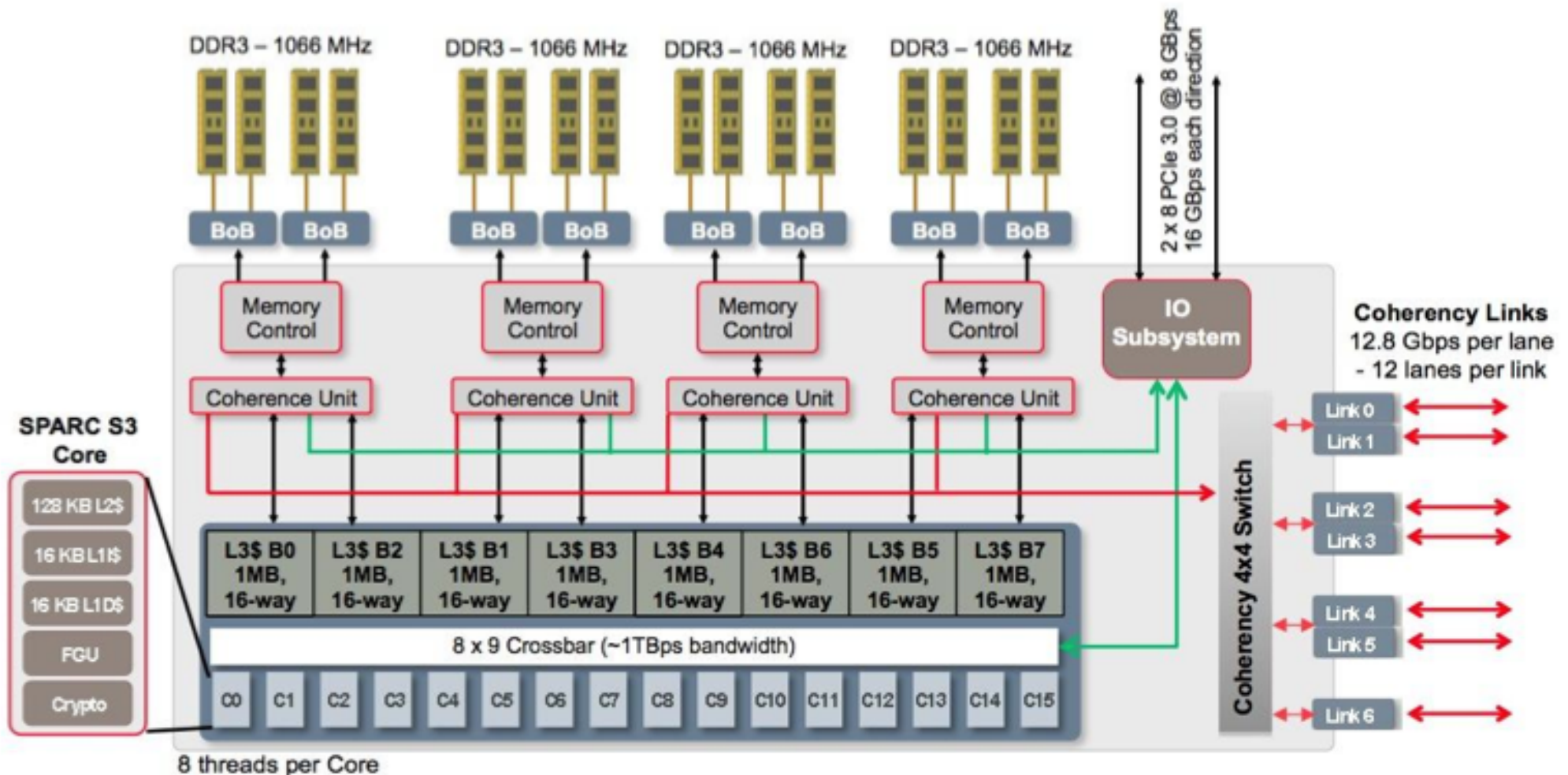
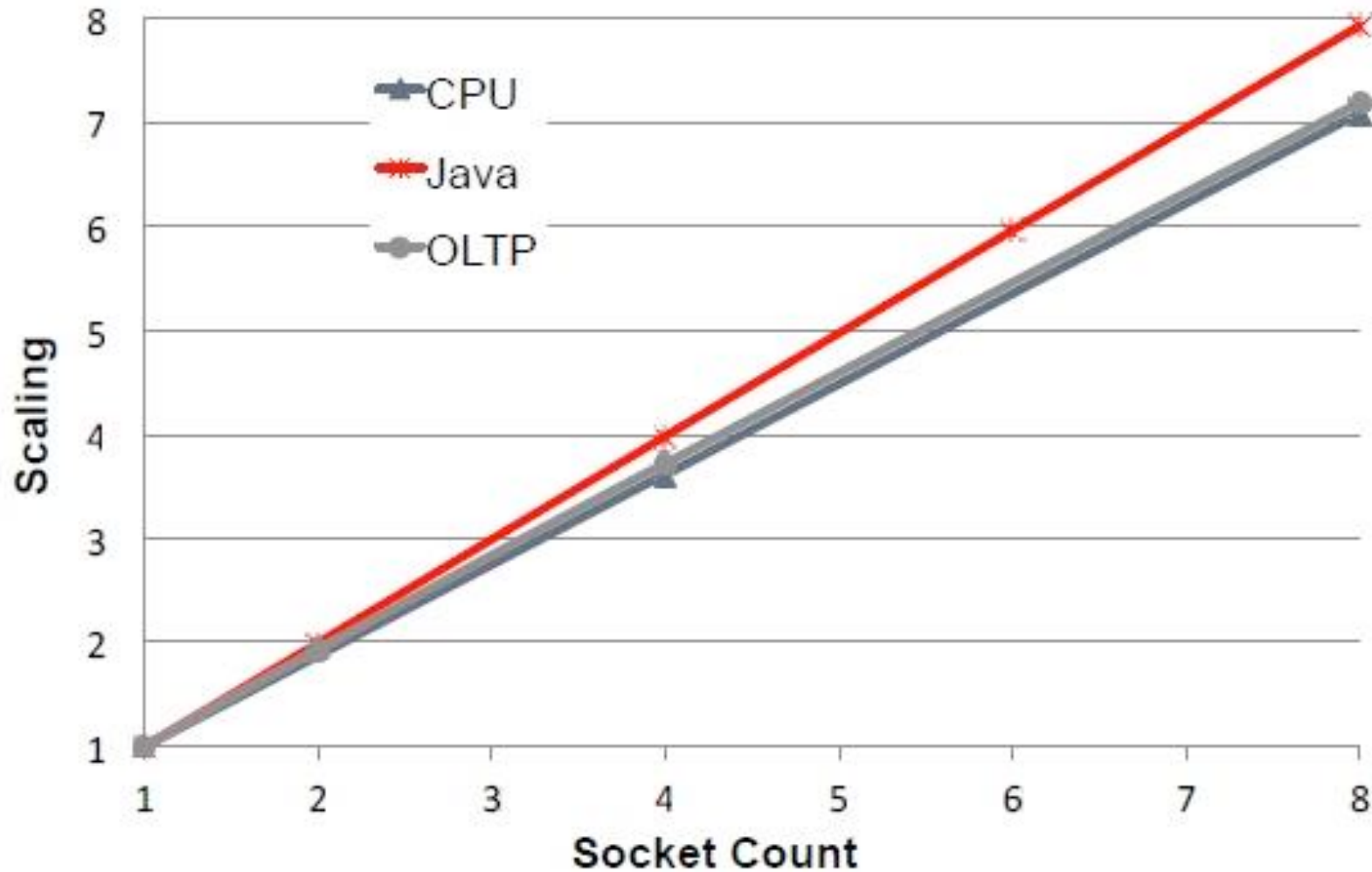


Figure 4. The SPARC T5 processor provides seven coherence links to connect to up to four other processors.

- Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture. Oracle White Paper. July 2013.

SPARC T5 Scaling



Key to scaling: directory-based coherence

- Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture. Oracle White Paper. July 2013.

SPARC T5 System Performance

“SPARC T5-8 Server Delivers World Record TPC-C Single System Performance”

TPC-C is an OLTP system benchmark. It simulates a complete environment where a population of terminal operators executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

	SPARC T5-8	IBM Power 780 3-node cluster	IBM Power 595	IBM x3850 X5	IBM Flex x240
Processor Model (CPUs/Cores/Threads)	3.6 GHz SPARC T5 (8/128/1024)	3.86 GHz Power 7 (24/192/768)	5.0 GHz Power 6 (32/64/128)	2.40 GHz Intel Xeon E7-8870 (4/40/80)	2.90 GHz Intel Xeon E5-2690 (2/16/32)
tpmC	8,552,523	10,366,254	6,085,166	3,014,684	1,503,544
Price / tpmC	\$0.55 USD	\$1.38 USD	\$2.81 USD	\$0.59 USD	\$0.53 USD
tpmC / CPU	1,069,065.4	431,927.3	190,161.4	753,671	751,772
Memory Size	4 TB	6 TB	4 TB	3 TB	768 TB
Database	Oracle Database 11g Release 2	IBM DB2 9.7	IBM DB2 9.5	IBM DB2 9.7	IBM DB2 9.7
Availability Date	9/25/2013	10/13/2010	12/10/2008	9/22/2011	8/16/2012

March 2013: <http://www.oracle.com/us/solutions/performance-scalability/sparc-t5-8-single-system-1925151.html>

Thought Questions

- **What are your thoughts about programmability of multithreaded processors?**
- **Comment about their suitability for**
 - dense matrix algorithms
 - sparse matrix algorithms
 - graph algorithms
- **What are the key issues in system design for multithreaded processors?**

References

- [Niagara: A 32-Way Multithreaded SPARC Processor](#), Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun, IEEE Micro, pp. 21-29, March-April 2005.
 - [Sun's Niagara falls neatly into multithreaded place](#), Charlie Demerjian, The Inquirer, 02 November 2004.
 - [OpenSparc T1 Microarchitecture Specification](#), Sun Microsystems, http://opensparc-t1.sunsource.net/specs/OpenSPARCT1_Micro_Arch.pdf
- [Niagara2: A Highly Threaded Server-on-a-Chip](#). Robert Golla. Sun Microsystems Slides, Oct. 10, 2006.
- [ELDORADO](#). John Feo, David Harper, Simon Kahan, Petr Konecny. Proceedings of the 2nd Conference on Computing Frontiers (Ischia, Italy, May 04 - 06, 2005). ACM, NY, NY, 28-34.

References

- [A survey of processors with explicit multithreading](#). T. Ungerer, B. Robič, and J. Šilc. ACM Computing Surveys 35, 1 (Mar. 2003), 29-63. DOI= <http://doi.acm.org/10.1145/641865.641867>
- [Eldorado](#). Presentation. John Feo. Cray, Inc., July 2005.
- [Evaluating the Potential for Multithreaded Platforms for Irregular Scientific Computations](#), Jarek Nieplocha, Andrés Márquez, John Feo, Daniel Chavarría-Miranda, George Chin, Chad Scherrer, Nathaniel Beagley. Proc. 4th Intl. Conf. on Computing Frontiers, Ischia, Italy, 2007, pages 47 - 58.
- [A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets](#), K. Madduri, D. Ediger, K. Jiang, D. A. Bader, D. Chavarria-Miranda. IPDPS, 2009, pp.1-8.
- [Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture](#). Oracle White Paper. July 2013.