PRAM-On-Chip Vision
Explicit Multi-Threading (XMT) Technology

Uzi Vishkin

University of Maryland

A. JAMES CLARK SCHOOL *of* ENGINEERING

What I (really) want you to remember

- PRAM silicon is here and more is coming.
- It provides freedom and opportunity to pursue PRAM-related research and education without waiting for vendors. This is legitimate work. No need to wait for parties that have their own agenda, or do not care for parallel algorithms and/or programming in the same way that some of us do.
- There may be something in it for you if you do.
- Consider using our tool chain (compiler & hardware) in your F07/S08 class
- Documentation includes: Tutorial & Manual.
- Also, Methodology for advancing from PRAM algorithm to programs
- Let's talk if you are interested

General-Purpose Market History

Processing power on-chip advancing rapidly since the 1970's

- Feature size ↓, transistor count and clock frequency ↑ => increasing power needs
- Not sustainable
  - Since 2003: Clock frequency traded for power consumption within limits

Dual (and multi-) Core Potential

	Area	Voltage	Freq.	Power	Performance…
Single core	1	1	1	1	1.0
Single core	1	0.85	0.85	0.5	~0.9
Dual core	2	0.85	0.85	1	~1.8

Crucial for IT growth (& Nat. Sec.): is this sustainable?

@HotChips06: "Intel has found applications, including benchmarks, scale well to around four cores before leveling off. The company has known for a while that just throwing cores at the problem won't necessarily make computers run faster."

Read: NO, unless something is done about it. Easier said than done. Widely known outcomes of 4 decades of parallel computing and looking under some stones …

The Pain of Parallel Programming

- Parallel programming is currently too difficult, making it unacceptable for many objectives.
  - To many users programming existing parallel computers is "as intimidating and time consuming as programming in assembly language" [NSF Blue-Ribbon Panel on Cyberinfrastructure].
- Tribal lore, parallel programming profs, DARPA HPCS Development Time study (2004-2008): "Parallel algorithms and programming for parallelism is easy. What is difficult is the programming/tuning for performance that comes after that."
- J. Hennessy: "Many of the early ideas were motivated by observations of what was easy to implement in the hardware rather than what was easy to use"

Note To meet timing constraints, some slides have detailed counterparts under "back-up slides"

Solution Approach to Parallel Programming Pain

- Parallel programming hardware should be a natural outgrowth of a well-understood parallel programming methodology
  - Methodology first
  - Build architecture
  - Validate approach

A parallel programming methodology got to start with parallel algorithms--exactly where our approach is coming from

[Parallel algorithms had to be new. The rest can be obtained using a pathway of levers. Special slide at the end.]

Parallel Random Access Model

(Recognizing par algs as an alien culture, “parallel-algorithms-first”--as opposed to: build-first, figure-out how to program later--started for me in 1979)

- PRAM Theory
  - Assume latency for arbitrary number of memory accesses is the same as for one access.
  - Model of choice for parallel algorithms in all major algorithms/theory communities. No real competition!
  - Main algorithms textbooks included PRAM algorithms chapters by 1990
  - Huge knowledge-base
  - Parallel computer architecture textbook [CS-99]: “.. breakthrough may come from architecture if we can truly design a machine that can look to the programmer like a PRAM”

Example of PRAM-like Algorithm

Input: (i) All world airports.

(ii) For each, all airports to which there is a non-stop flight.

Find: smallest number of flights from DCA to every other airport.

Basic algorithm

Step i:

For all airports requiring i-1flights

For all its outgoing flights

Mark (concurrently!) all "yet unvisited" airports as requiring i flights (note nesting)

Serial: uses "serial queue".

O(T) time; T – total # of flights

Parallel: parallel data-structures.

Inherent serialization: S.

Gain relative to serial: (first cut) ~T/S!

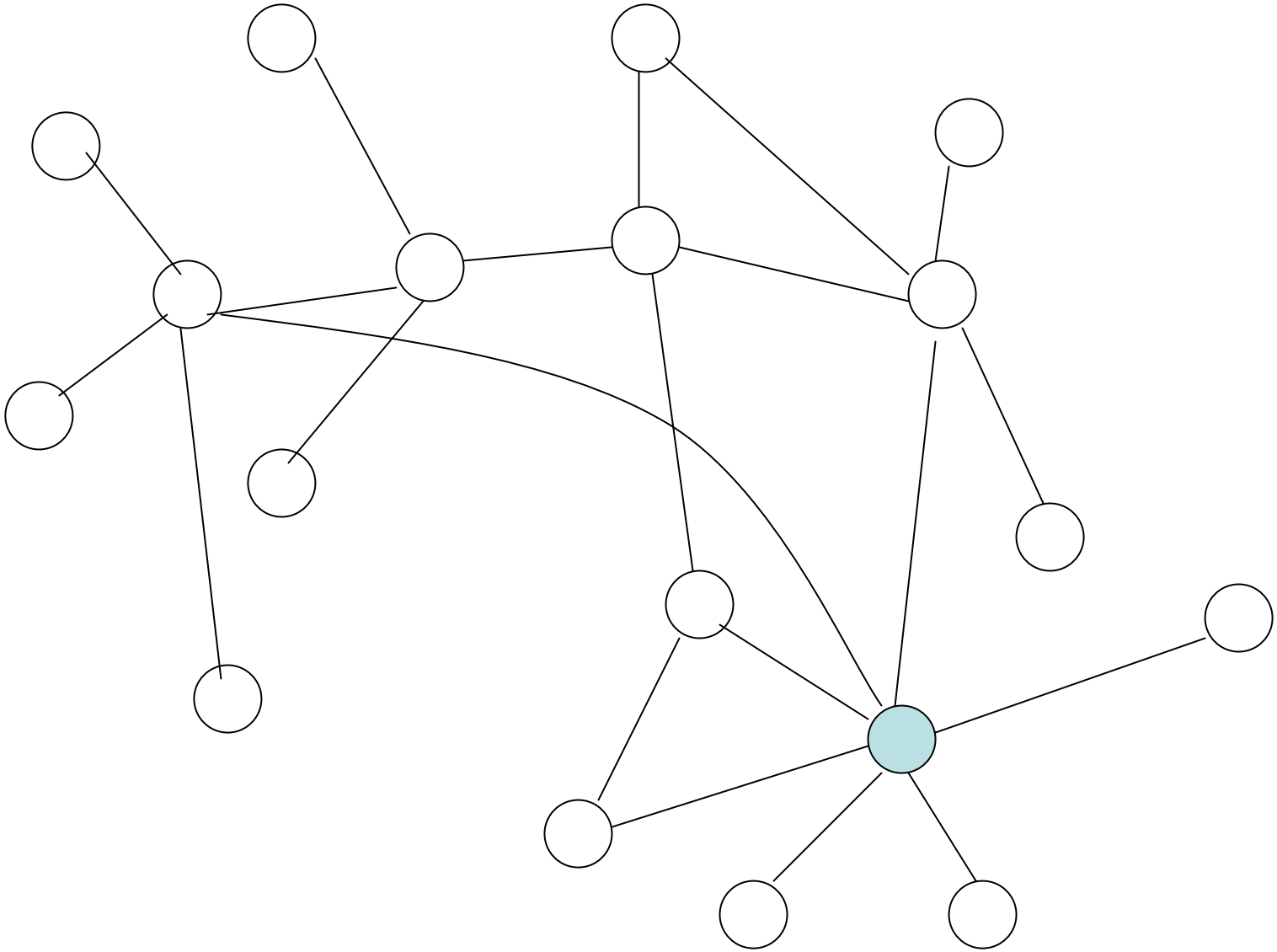
Decisive also relative to coarse-grained parallelism.

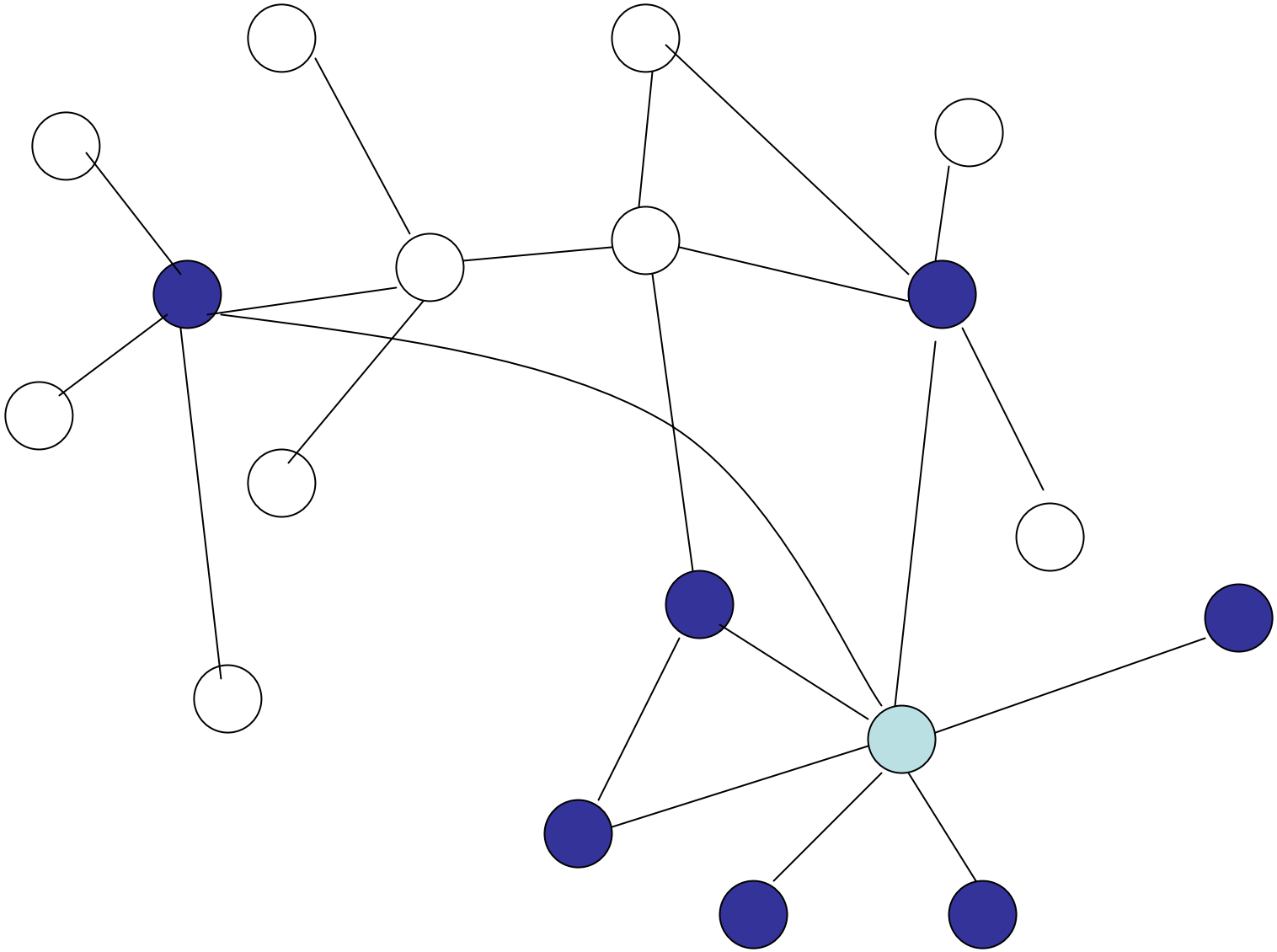
Note: (i) "Concurrently": only change to serial algorithm

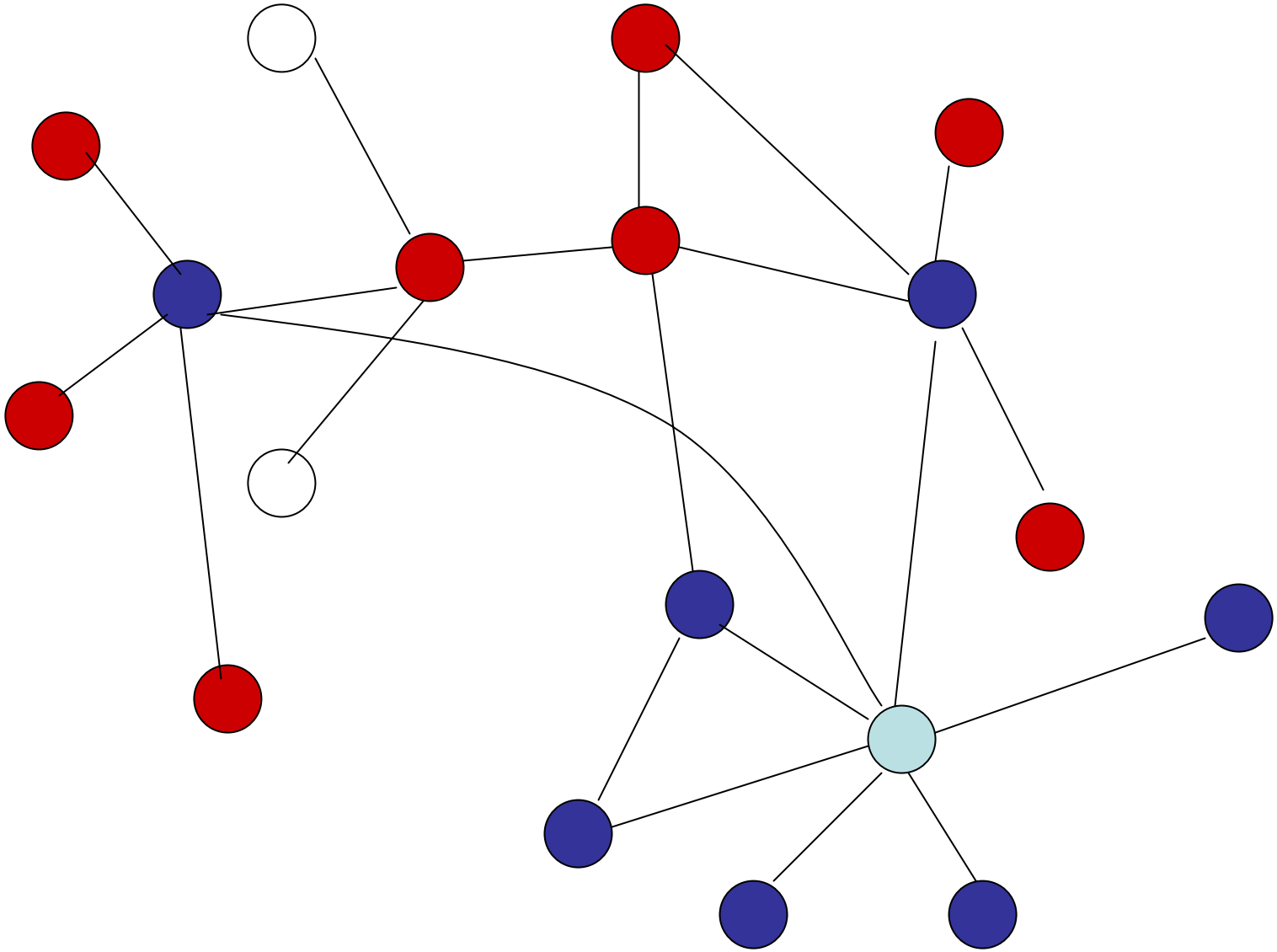
(ii) No "decomposition"/"partition"

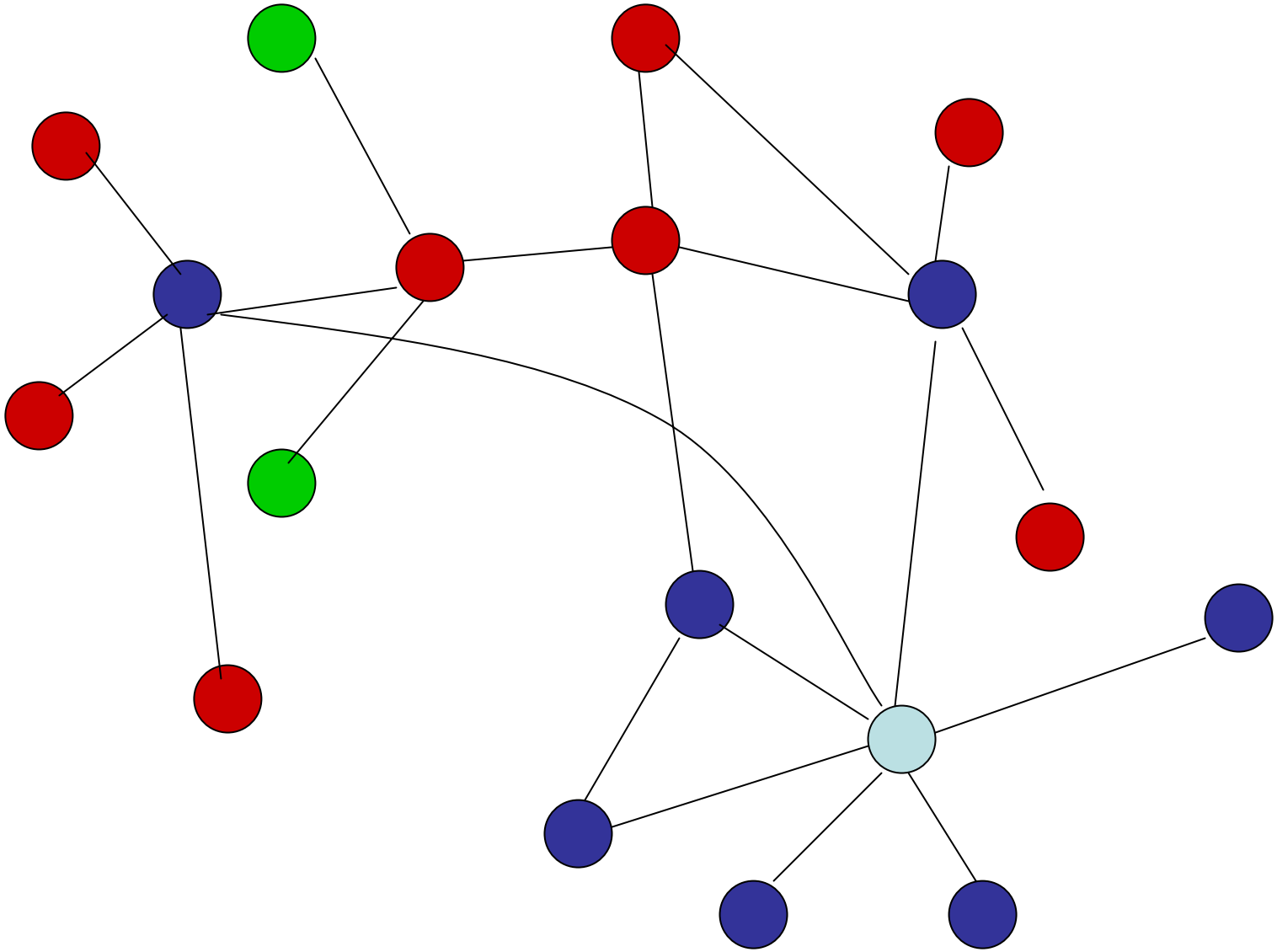
(iii) Takes the better part of a semester to teach!

Please take into account that based on experience with scores of good students this semester-long course is needed to make full sense of the approach presented here.









PRAM-On-Chip

- Reduce general-purpose single-task completion time.
- Go after any amount/grain/regularity of parallelism you can find.
- Premises (1997):
  - within a decade transistor count will allow an on-chip parallel computer (1980: 10Ks; 2010: 10Bs)
  - Will be possible to get good performance out of PRAM algorithms
- But why? crash course on parallel computing
  - How much processors-to-memories bandwidth?

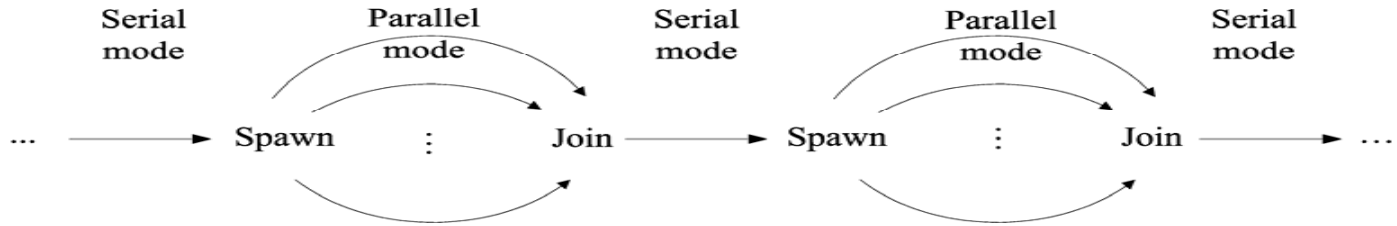
Enough	Limited
Ideal Programming Model: PRAM	Programming difficulties

PRAM-On-Chip provides enough bandwidth for on-chip processors-to-memories interconnection network. PRAM was just ahead of its time.

How does it work

"Work-depth" Algorithms State all ops you can do in parallel. Repeat. Minimize: Total #operations, #rounds (source SV82)

Program single-program multiple-data (SPMD). Short (not OS) threads. Independence of order semantics (IOS). XMTC: C plus 3 commands: Spawn+Join, Prefix-Sum



Programming methodology Algorithms → effective programs.

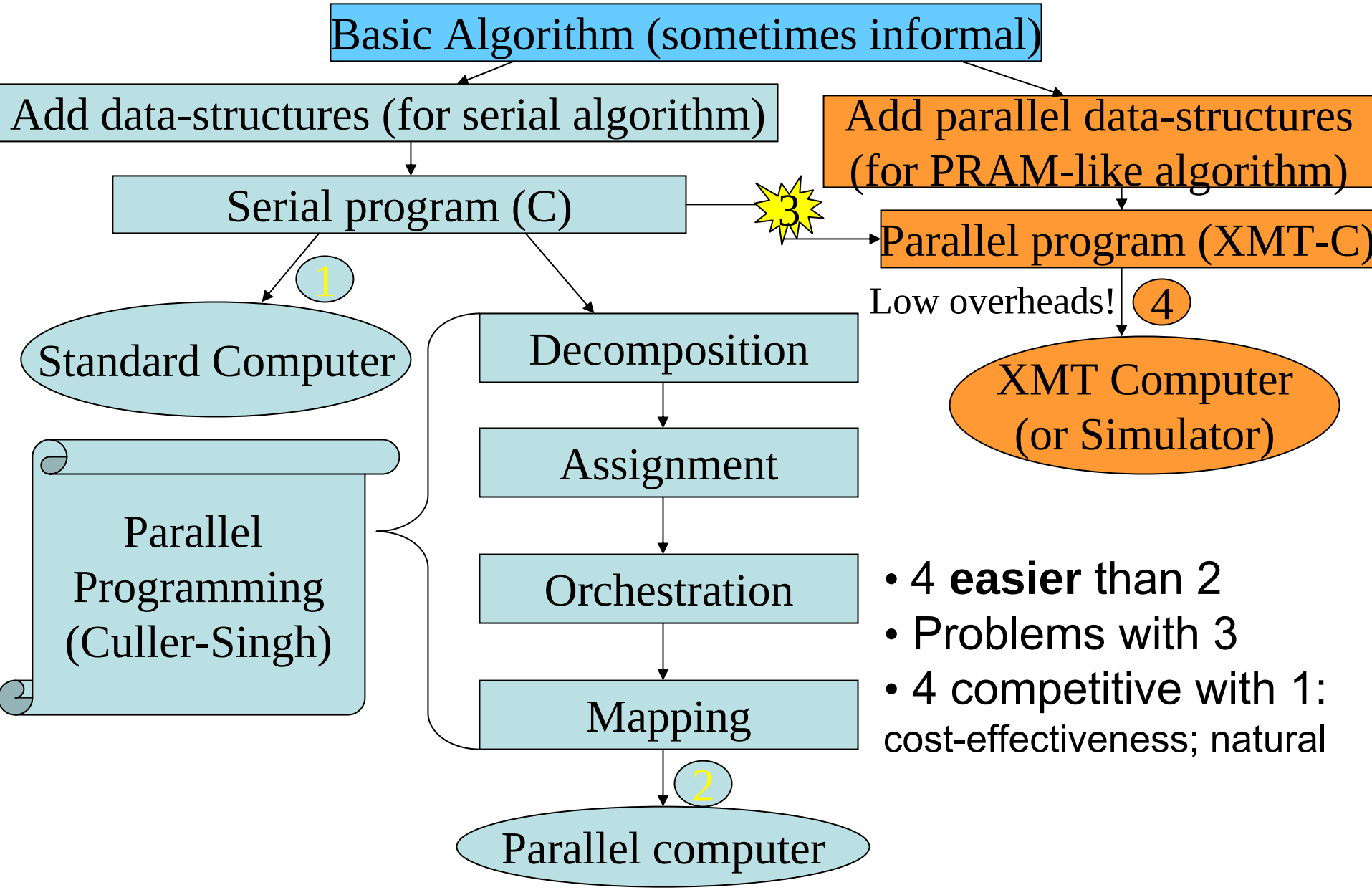
Extend the SV82 Work-Depth framework from PRAM to XMTC

Or ***Established APIs*** (VHDL/Verilog, OpenGL, MATLAB) "win-win proposition"

→ ***Compiler*** minimize length of sequence of round-trips to memory; take advantage of architecture enhancements (e.g., prefetch)

Architecture Dynamically load-balance concurrent threads over processors. "OS of the language". (Prefix-sum to registers & to memory.)

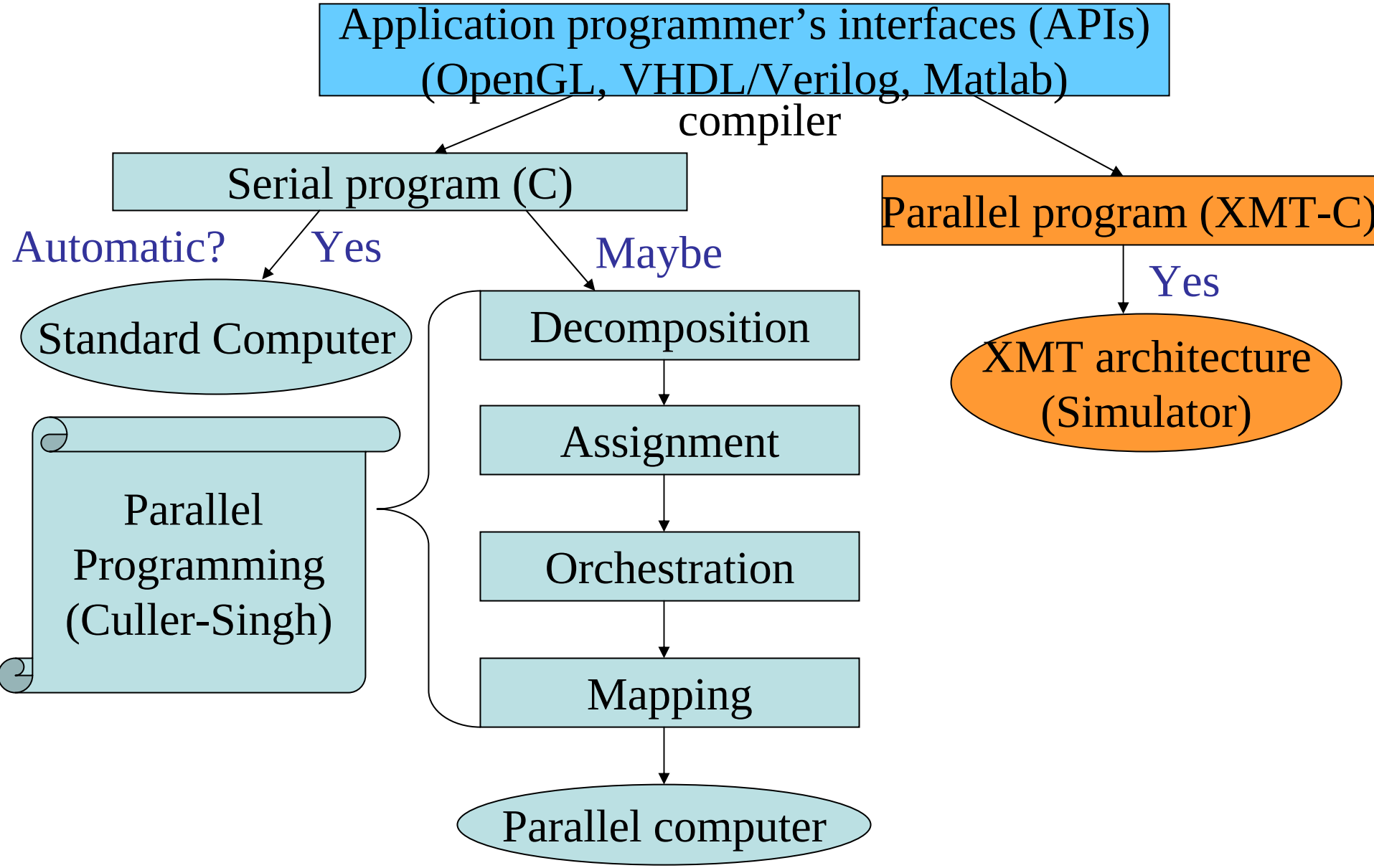
PERFORMANCE PROGRAMMING & ITS PRODUCTIVITY



- 4 **easier** than 2
- Problems with 3
- 4 competitive with 1:

cost-effectiveness; natural

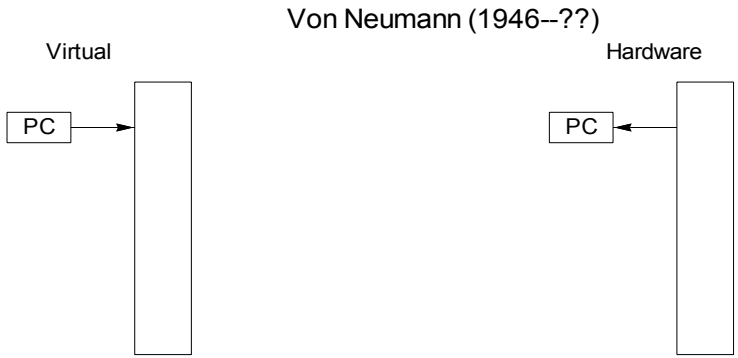
APPLICATION PROGRAMMING & ITS PRODUCTIVITY

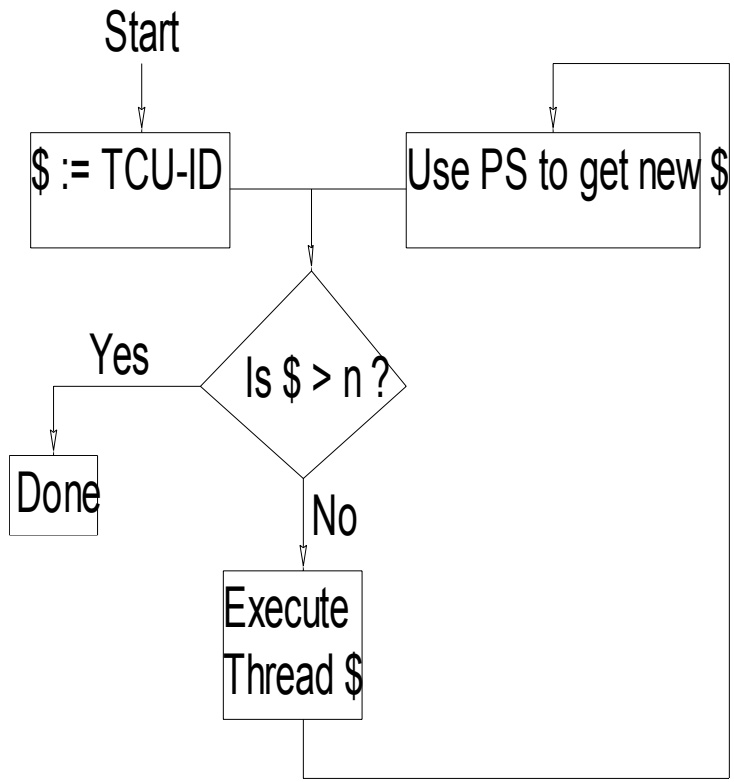


How-To Nugget - Time allows only one

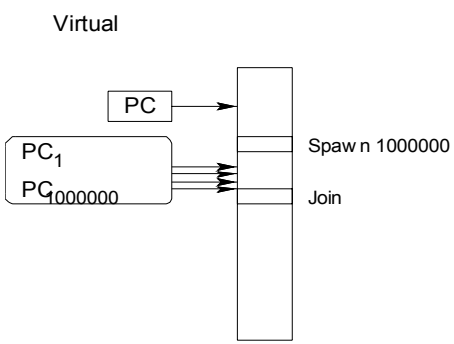
Seek 1st (?) upgrade of program-counter & stored program since 1946

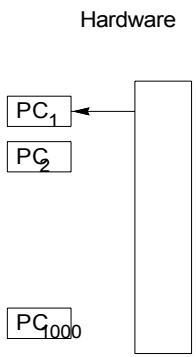
Virtual over physical: distributed solution





XMT

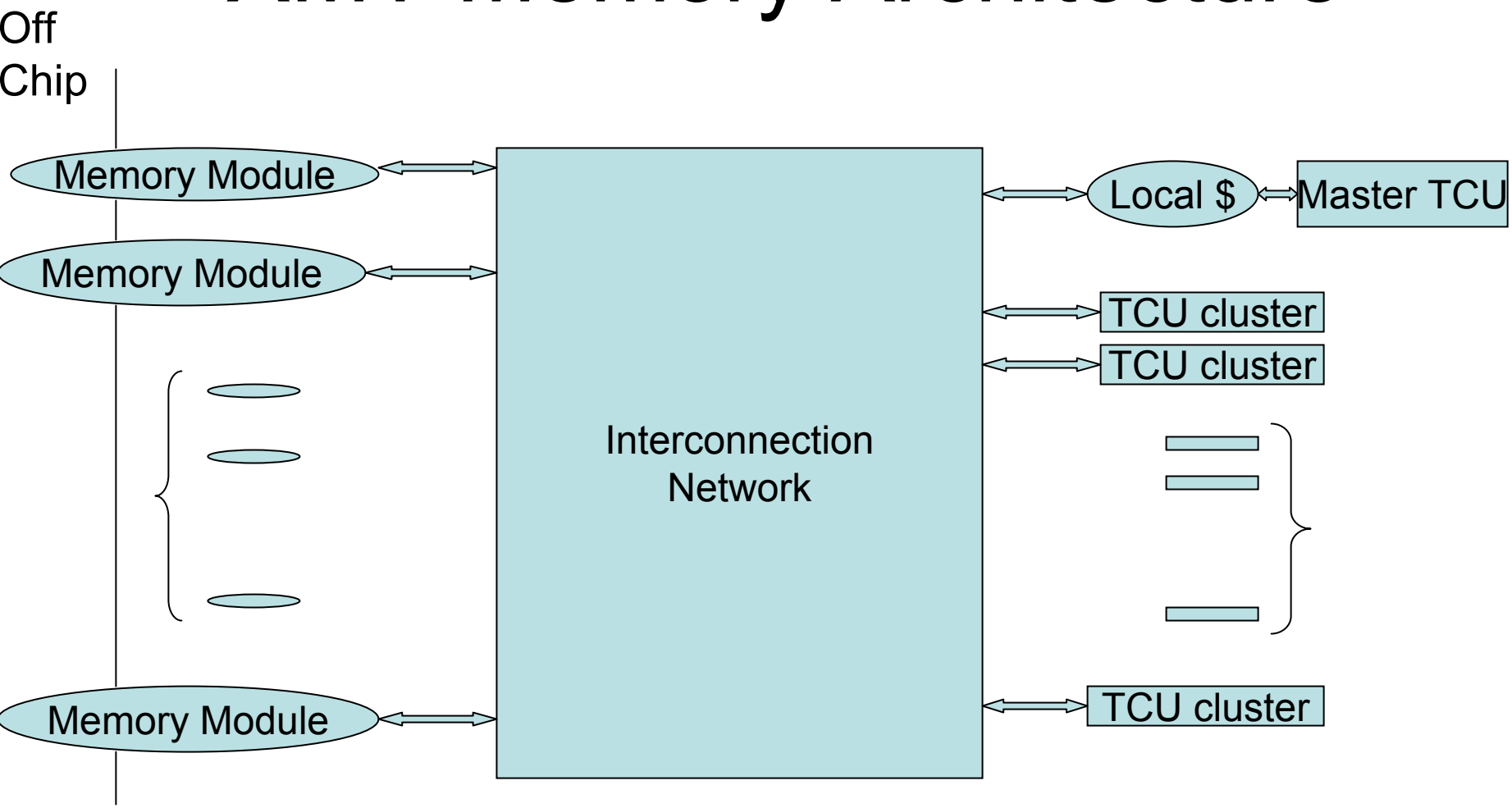




When PC1 hits Spawn, a spawn unit broadcasts 1000000 and the code

to PC1, PC 2, PC1000 on a designated bus

XMT Memory Architecture



-Cache-coherence defined away
-Transition from serial to parallel, and serial compatibility
-Size example: 1024 TCUs in 64 clusters, 64 MMs

XMT Development

- Hardware Track
  - Interconnection network. Led so far to:
  - ❑ ASAP'06 Best paper award for mesh of trees (MoT) study
  - ❑ Using IBM+Artisan tech files: 4.6 Tbps average output at max frequency (1.3 - 2.1 Tbps for alt networks)! No way to get such results without such access
  - ❑ 90nm ASIC for March'07 tapeout??
  - Synthesizable Verilog of the whole architecture. Led so far to:
  - ❑ Cycle accurate simulator. Slow. For 11-12K X faster:
  - ❑ 1st commitment to silicon—64-processor, 75MHz computer; uses FPGA: Industry standard for pre-ASIC prototype; have done our homework for ASIC
  - ❑ 1st ASIC prototype?? 90nm ASIC for Summer'07 tapeout. 5 grad students working full time
- Compiler Done: Basic. To do: Optimizations. Match HW enhancement.
  - Basic, yet stable, compiler completed
  - To do: prefetch, clustering, broadcast, nesting, non-blocking store. Optimizations.
- Applications
  - Methodology for advancing from PRAM algorithms to efficient programs
  - Understanding of backwards compatibility with (standard) higher level programming interfaces (e.g., Verilog/VHDL, OpenGL, MATLAB)
  - More work on applications with progress on compiler, cycle-accurate simulator, new XMT FPGA and ASIC. Feedback loop to HW/compiler.
  - A DoD-related benchmark coming

Tentative DoD-related speedup result

- DARPA HPC Scalable Synthetic Compact Application (SSCA 2) Benchmark – Graph Analysis. (Problems size: 32k vertices, 256k edges.)

	Speedup	Description
Kernel 1	72.68	Builds the graph data structure from the set of edges
Kernel 2	94.02	Searches multigraph for desired maximum integer weight, and desired string weight
Kernel 3	173.62	Extracts desired subgraphs, given start vertices and path length
Kernel 4	N/A	Extracts clusters (cliques) to help identify the underlying graph structure

- HPC Challenge Benchmarks

DGEMM	580.28	Dense (integer) matrix multiplication. Matrix size: 256x256.
HPL(LU)	54.62	Linear equation system solver. Speedup computed for LU factorization kernel, integer values. XMT configuration: 256TCUs in 16Clusters. Matrix size: 256x256.

Serial programs are run on the Master TCU of XMT. All memory requests from Master TCU are assumed to be Master Cache hits-- An advantage to serial programs.
Parallel programs are ran with 2MB L1 cache 64X2X16KB. L1 cache miss is served from L2, which is assumed preloaded (by an L2 prefetching mechanism). Prefetching to prefetch buffers, broadcasting and other optimization have been manually inserted in assembly.
Except for HPL(LU), XMT is assumed to have 1024 TCUs grouped in 64 clusters.

New XMT (FPGA-based) computer

Some Specs

System clock rate	75MHz
Memory size	1GB DDR2 SODIMM
Memory data rate	300 MHz, 2.4 GB/s
# TCUs	64 (4 x 16)
Shared cache size	64KB (8X 8KB)
MTCU local cache size	8KB

AMD Opteron 2.6 GHz, RedHat Linux Enterprise 3, 64KB+64KB L1 Cache, 1MB L2 Cache (none in XMT), memory bandwidth 6.4 GB/s (X2.67 of XMT)

M_Mult was 2000X2000: XMT beats AMD Opteron
QSort was 20M

Execution time

App.	XMT Basic	XMT Enhanced	AMD
M-Mult	182.8 sec	80.44	113.83
QSort	16.06	7.57	2.61

Note: First commitment to silicon. “We can build”.

Aim: prototype main features.
No FP. 64→32-bit.

Enhanced XMT: Broadcast, prefetch + buffer, non-blocking store. Nearly done: non-blocking caches.

More XMT Outcomes & features

- 100X speedups for VHDL gate-level simulation on common benchmark. Journal paper 12/2006.
- Backwards compatible (&competitive) for serial
- Works with whatever parallelism. scalable (grain, irregular)

- Programming methodology & training kit (3 docs: 150 pages)
  - Hochstein-Basili: 50% development time of MPI for MATVEC (2nd vs 4th programming assignment at UCSB)
  - Class tested: parallel algorithms (not programming) class, assignments on par with serial class
- Single inexperienced student in 2+ years from initial Verilog design: FPGA of a Billion transistor architecture that beats 2.6 GHz AMD Proc. On M_Mult. Validates: XMT architecture (not only the prog model) is a very simple concept. Implies: faster time to market, lower implementation cost.

Application-Specific Potential of XMT

- Chip-supercomputer chassis for application-optimized ASIC.

❑ General idea: Fit to suit – function, power, clock

❑ More/less FU of any type

❑ Memory size/issues

❑ Interconnection options; synchrony levels

❑ All: easy to program & jointly SW compatible.

Examples: MIMO; Support in one system >1 SW defined radio/wireless standards; recognition of need for general-purpose platforms in AppS is growing; reduce synchrony of int. connect for power (battery life)

Other approaches

None has a competitive parallel programming model, or supports a broad range of APIs

- Streaming: XMT can emulate (*using prefetching). Not the opposite.
- Transactional memory: OS threads+PS. Like streaming, does some things well, not others.
  - What TM can do XMT can, but not the opposite.
  - TM less of a change to past architectures. But, why architecture loyalty? backwards compatibility on code is important
- Cell-Processor Based:

  Not easy to program.

http://www.electronicsweekly.com, 4'06: **Will multi-core processing fulfill its potential?**
The goal is a programming tool which is understandable, accessible and readily usable by programmers, and which fully exploits the power of parallelism. No one is expecting this anytime soon. So there is the problem. Multi-core is perceived as difficult to program, programming tools do not really exist, and the programmers are entrenched in a serial mind-set. **...**
Although everyone accepts that multi-cores running at low speed get over the power density brick wall which was hit by single processors running at high speed, people are finding multi-core much more difficult to implement than expected. "There's a lot of snake oil around in the multi-core business," says Alan Gatherer, CTO for communication infrastructure at Texas Instruments. "I'm not claiming we understand the secret. It varies from application to application. I'm not sure anyone knows how to build a generic multi-core architecture. It's a great goal, but the chances of failure are 100 per cent."
Gatherer's point is that multi-core delivers performance when it is targeted at a specific application when you know what each part of the chip will be doing, but it becomes a nightmare when you try to produce a microprocessor which can be programmed for many different applications which, of course, is supposed to be the whole point of a microprocessor.
The problem is caused by multiple cores, all the same, which are intended to work as a generic processor.
"We can produce a chip with a lot of theoretical Mips but it won't be very programmable, the difficulty is partitioning the programming across the cores," says TI's Gatherer. "Our customers tell us that, if you have to partition an algorithm across multiple cores and expect them all to talk to each other in real-time, that's a hard problem."

That problem has to be solved by programmers, and everyone seems to agree that programmers are entrenched in a multi-core resistant mind-set which is the main hindrance to the wider adoption of multi-core architectures.

Peter Claydon, founder and COO of PicoChip, reckons: “A lot of the parallel processing start-ups have failed because they try to use a sequential programming language like C. The mentality of programmers is to expect everything to be serial. Multi-core is here to stay, but it needs a new way of thinking. I have talked to people who are thinking of doing start-ups to produce multi-core programming tools.”

“Customers say: ‘Be very careful with parallel architectures’. They ask us, ‘you don’t know how to program these massively parallel devices yourselves do you?’,” says TI’s Gatherer. “That’s one of the reasons why there’s been no traction for those companies doing massively parallel architectures. These companies program their own devices because no one else can.

"When they say: ‘We’ve got a reference design so you don’t need to program it yourself’, what they really mean is, ‘we’ve got a reference design because we know you’ll never be able to program it yourself’. But a lot of our customers feel they have their own competitive differentiation, which they can provide from programming the chip themselves. That’s what we provide.”

The goal is a programming tool which is understandable, accessible and readily usable by programmers, and which fully exploits the power of parallelism. No one is expecting this anytime soon.

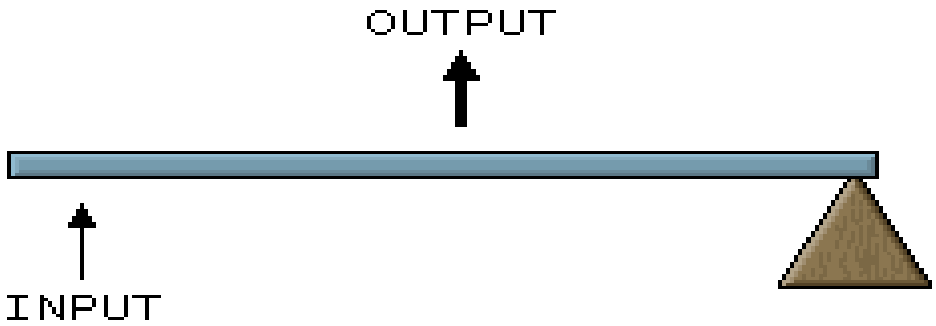
So there is the problem. Multi-core is perceived as difficult to program, programming tools do not really exist, and the programmers are entrenched in a serial mind-set.

[UV: 1st observation: too many targets to hit in one shot. Need a pathway to solution.]

That is a lot of problems but, as everyone agrees, multi-core is the future.

Summary of technical pathways

It is all about (2nd class) levers



Credit: Archimedes

Reported:

Parallel algorithms. First principles. Alien culture: had to do from scratch. (No lever)

Levers:

1. Input: Parallel algorithm. Output: Parallel architecture.
2. Input: Parallel algorithms & architectures. Output: parallel programming

Proposed:

- Input: Above. Output: For select AppS application niche.
- Input: Above Apps. Output: GP.

Bottom Line

Cures a potentially fatal problem for growth of general-purpose processors: How to program them for single task completion time?

Please consider our record

	Proposal	Over-Delivering
NSF ‘97-’02	exper. Algorithms	architecture
NSF 2003-8	arch. simulator	silicon (FPGA)
DoD 2005-7	FPGA	FPGA+ASIC*

*subject to access

Final thought: Created our own coherent planet

- When was the last time that a professor offered a (separate) algorithms class on own language, using own compiler and own computer?
- Colleagues could not provide an example since at least the 1950s. Have we missed anything?

List of recent papers

A.O Balkan, G. Qu, and U. Vishkin. Mesh-of-trees and alternative interconnection networks for single-chip parallel processing. In ASAP 2006: 17th IEEE Int. Conf. on Application-specific Systems, Architectures and Processors, 73–80, Steamboat Springs, Colorado, 2006. Best Paper Award.

A.O. Balkan and U. Vishkin. Programmer's manual for XMTC language, XMTC compiler and XMT simulator. Technical Report, February 2006. 80+ pages.

P. Gu and U. Vishkin. Case study of gate-level logic simulation on an extremely fine-grained chip multiprocessor. Journal of Embedded Computing, Dec 2006.

D. Naishlos, J. Nuzman, C-W. Tseng, and U. Vishkin. Towards a first vertical prototyping of an extremely fine-grained parallel programming approach. In invited Special Issue for ACM-SPAA'01: TOCS 36,5, pages 521–552, New York, NY, USA, 2003.

A. Tzannes, R. Barua, G.C. Caragea, and U. Vishkin. Issues in writing a parallel compiler starting from a serial compiler. Draft, 2006.

U. Vishkin, G. Caragea and B. Lee. Models for Advancing PRAM and Other Algorithms into Parallel Programs for a PRAM-On-Chip Platform. In R. Rajasekaran and J. Reif (Editors), to appear in Handbook of Parallel Computing, CRC Press, December 2007. 60+ pages.

U. Vishkin, I. Smolyaninov and C. Davis. Plasmonics and the parallel programming problem. Silicon Photonics Conference, SPIE Symposium on Integrated Optoelectronic Devices 2007, January 20-25, 2007, San Jose, CA.

X. Wen and U. Vishkin. PRAM-On-Chip: First commitment to silicon. SPAA'07.

Contact Information

Uzi Vishkin

The University of Maryland Institute for Advanced Computer Studies (UMIACS) and Electrical and Computer Engineering Department

Room 2365, A.V. Williams Building

College Park, MD 20742-3251

Phone 301-405-6763. Shared fax: 301-314-9658

Home page: http://www.umiacs.umd.edu/~vishkin/

Back up slides

From here on all slides are back-up slides

Solution Approach to Parallel Programming Pain

- Parallel programming hardware should be a natural outgrowth of a well-understood parallel programming methodology
  - Methodology first
  - Architecture specs should fit the methodology
  - Build architecture
  - Validate approach

A parallel programming methodology got to start with parallel algorithms--exactly where our approach is coming from

Parallel Random Access Model

(started for me in 1979)

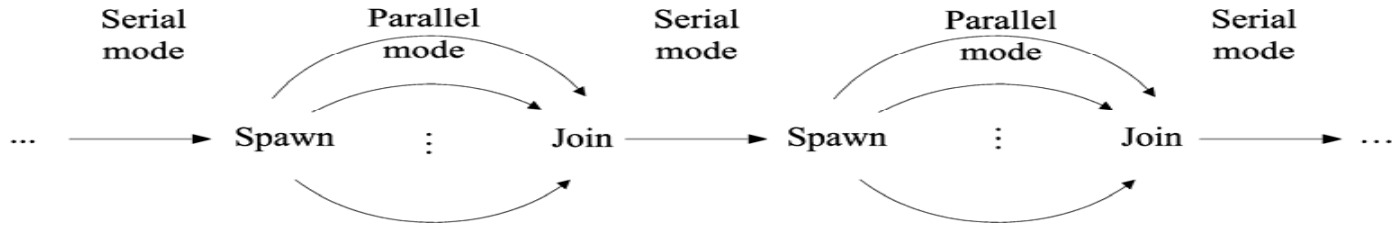
- PRAM Theory
  - Assume latency for arbitrary number of memory accesses is the same as for one access.
  - Full overheads model (like serial RAM).
  - Model of choice for parallel algorithms in all major algorithms/theory communities. No real competition!
  - Main algorithms textbooks included PRAM algorithms chapters by 1990
  - Huge knowledge-base
  - Parallel computer architecture textbook [CS-99]: ".. breakthrough may come from architecture if we can truly design a machine that can look to the programmer like a PRAM"

How does it work

Algorithms State all that can be done in parallel next. Repeat.

Minimize: Total #operations, #rounds Arbitrary CRCW PRAM SV-82a+b

Program single-program multiple-data (SPMD). Short (not OS) threads. Independence of order semantics (IOS). Nesting possible. XMTC: C plus 3 commands: Spawn+Join, Prefix-Sum



Programming methodology Algorithms → effective programs.

General Idea: Extend the SV-82b Work-Depth framework from PRAM to XMTC

Or ***Established APIs*** (VHDL/Verilog, OpenGL, MATLAB) "win-win proposition"

→ ***Compiler*** prefetch, clustering, broadcast, nesting implementation, non-blocking stores, minimize length of sequence of round-trips to memory

Architecture Dynamically load-balance concurrent threads over processors. "OS of the language". (Prefix-sum to registers & to memory.) Easy transition serial2parallel. Competitive performance on serial. Memory architecture defines away cache-coherence. High throughput interconnection network.

New XMT (FPGA-based) computer: Backup slide

Some Specs

System clock rate	75MHz
Memory size	1GB DDR2 SODIMM
Memory data rate	300 MHz, 2.4 GB/s
# TCUs	64 (4 x 16)
Shared cache size	64KB (8X 8KB)
MTCU local cache size	8KB

Execution time

App.	XMT Basic	XMT Enhanced	AMD
M-Mult	182.8 sec	80.44	113.83
QSort	16.06	7.57	2.61

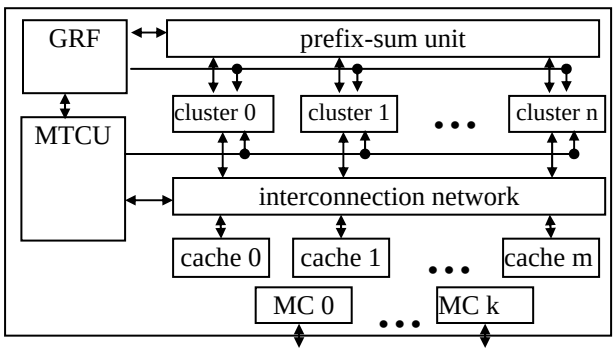
AMD Opteron 2.6 GHz, RedHat Linux Enterprise 3, 64KB+64KB L1 Cache, 1MB L2 Cache (none in XMT), memory bandwidth 6.4 GB/s (X2.67 of XMT)

M_Mult was 2000X2000: XMT beats AMD Opteron
QSort was 20M

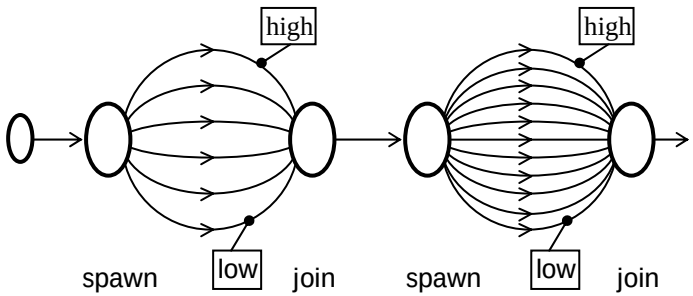
Note: First commitment to silicon. "We can build".

Aim: prototype main features.
No FP. 64→32-bit.
Imperfect reflection of ASIC performance
Irrelevant for power.

Enhanced XMT: Broadcast, prefetch + buffer, non-blocking store. Nearly done: non-blocking caches.

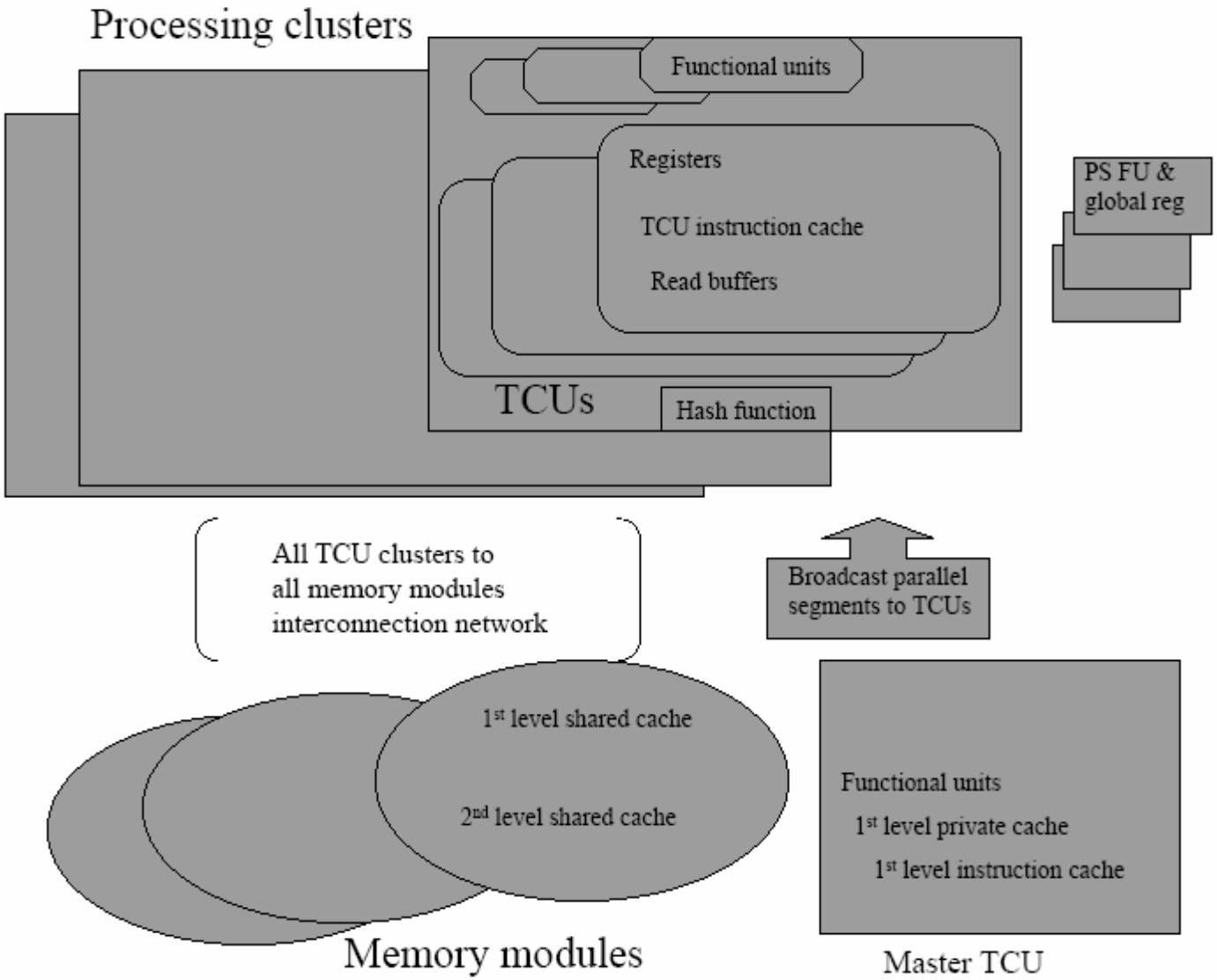


block diagram of the XMT processor



parallel and serial mode

XMT Block Diagram – Back-up slide



Back up slide: Post ASAP'06 Quantitative Study of Mesh of Trees & Others

	MOT-64	HYC-64 Typical	HYC-64 Max tput/cycle	BF-64 Typical	BF-64 Max tput/cycle
Number of packet registers	24k	3k	49k	6k	98k
Switch Complexity: Total Switch Delay and Pipeline Stages / Switch	0.43 ns, 1stage	1.3 ns, 3 stages	2.0 ns 3 stages	1.0 ns 3 stages	1.7 ns 3 stages
End-to-end packet latency with low traffic (cycles)	13	19	19	19	19
End-to-end packet latency with high traffic (cycles)	23	N/A	38	N/A	65
Maximum operating Frequency (GHz)	2.32	1.34	0.76	1.62	0.84
Cumulative Peak Tput at max Frequency (Tbps)	4.7	2.7	1.6	3.3	1.7
Cumulative Avg Tput at max Frequency (Tbps)	4.6	2.1	1.3	1.8	1.6
Cumulative Avg Tput at 0.5 GHz clock, (Tbps)	0.99	0.78	0.86	0.56	0.95

Technology files (IBM+Artisan) allowed this work

Backup slide: Assumptions

- **Typical** HYC/BF configurations have v=4 virtual channels (packet buffers)
- **Max Tput/Cycle** As one way for comparing the 3 topologies, a frequency (.5 GHz) was picked. For that frequency, throughout of both HYC and BF is maximized by configuring them to have v=64 virtual channels. As a result, we can compare the throughput of the 3 topologies by simply measuring **packets per cycle**. This effect is reflected at the bottom row, where all networks run at the same frequency. As can be seen, at that frequency, the max tput/cycle configurations performs better than their v=4 counterparts.
- **End-to-end packet latency** is measured
  - At 1% of network capacity for ***low traffic***
  - At 90% of network capacity for ***high traffic***
  - Network capacity is ***1 packet delivered per port per cycle***
- **Typical** configurations of HYC and BF could not support high traffic, they reach saturation at lower traffic rates.
  - **Typical** HYC saturates around 75% traffic
  - **Typical** BF saturates around 50% traffic
- **Cumulative Tput** includes all 64 ports

More XMT Outcomes & features

- 100X speedups for VHDL gate-level simulation on common benchmark. Journal paper 12/2006.
- Easiest approach to parallel algorithm & programming (PRAM) gives effective programs. *Irregular & fine-grained. Established APIs (VHDL/Verilon, OpenGL, MATLAB)
- Extendable to high-throughput light tasks (e.g., random-access)
- Works with whatever parallelism. scalable (grain, irregular)
- Backwards compatible (&competitive) for serial

- Programming methodology & training kit (3 docs: 150 pages)
  - Hochstein-Basili: 50% development time of MPI for MATVEC (2nd vs 4th programming assignment at UCSB)
  - Class tested: parallel algorithms (not programming) class, assignments on par with serial class
- Single inexperienced student in 2+ years from initial Verilog design: FPGA of a Billion transistor architecture that beats 2.6 GHz AMD Proc. On M_Mult. Validates: XMT architecture (not only the prog model) is a very simple concept. Implies: faster time to market, lower implementation cost.

The XMT Overall Design Challenge

- Assume algorithm scalability is available.
- Hardware scalability: put more of the same
- ... but, how to manage parallelism coming from a programmable API?

Spectrum of Explicit Multi-Threading (XMT) Framework

- Algorithms −− > architecture −− > implementation.
- XMT: strategic design point for fine-grained parallelism
- New elements are added only where needed

Attributes

- Holistic: A variety of subtle problems across different domains must be addressed:
- Understand and address each at its correct level of abstraction

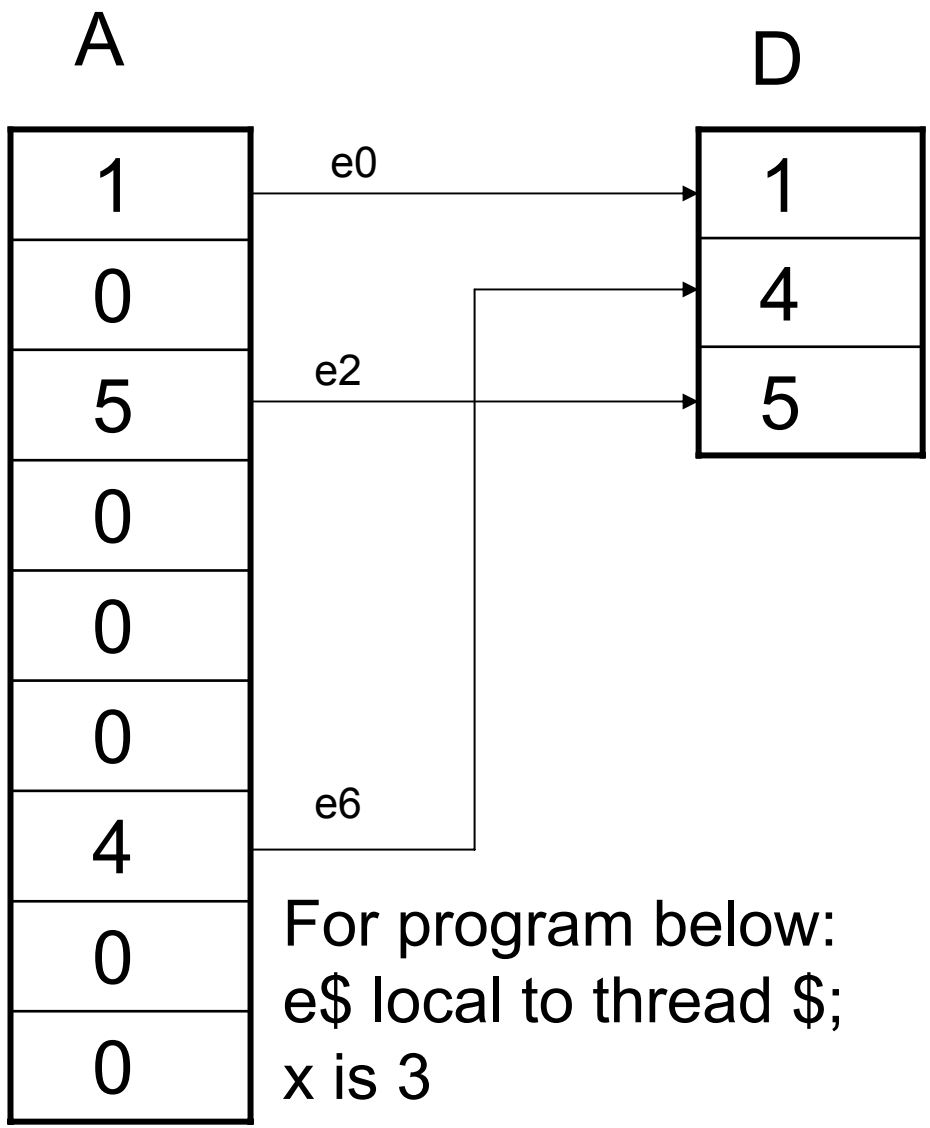
Snapshot: XMT High-level language

Cartoon Spawn creates threads; a thread progresses at its own speed and expires at its Join.
Synchronization: only at the Joins. So, virtual threads avoid busy-waits by expiring. New: Independence of order semantics (IOS).

The array compaction (artificial) problem

Input: Array A[1..n] of elements.

Map in some order all A(i) not equal 0 to array D.



XMT-C

Single-program multiple-data (SPMD) extension of standard C.
Includes Spawn and PS - a multi-operand instruction.

Essence of an XMT-C program

```
int x = 0;
Spawn(0, n) /* Spawn n threads; $ ranges 0 to n − 1 */
{ int e = 1;
   if (A[$] not-equal 0)
       { PS(x,e);
         D[e] = A[$] }
}
n = x;
```

Notes: (i) PS is defined next (think F&A). See results for e0,e2, e6 and x. (ii) Join instructions are implicit.

XMT Assembly Language

Standard assembly language, plus 3 new instructions: Spawn, Join, and PS.

The PS multi-operand instruction

New kind of instruction: Prefix-sum (PS).
Individual PS, PS Ri Rj, has an inseparable ("atomic") outcome:
(i) Store Ri + Rj in Ri, and
(ii) store original value of Ri in Rj.

Several successive PS instructions define a multiple-PS instruction. E.g., the sequence of k instructions:
PS R1 R2; PS R1 R3; ...; PS R1 R(k + 1)
performs the prefix-sum of base R1 elements R2,R3, ...,R(k + 1) to get:
R2 = R1; R3 = R1 + R2; ...; R(k + 1) = R1 + ... + Rk; R1 = R1 + ... + R(k + 1).

Idea: (i) Several ind. PS's can be combined into one multi-operand instruction.
(ii) Executed by a new multi-operand PS functional unit.

Mapping PRAM Algorithms onto XMT

(1) PRAM parallelism maps into a thread structure

(2) Assembly language threads are not-too-short (to increase locality of reference)

(3) the threads satisfy IOS

How (summary):

I. Use work-depth methodology [SV-82] for "thinking in parallel". The rest is skill.

II. Go through PRAM or not.

III. Produce XMTC program accounting also for:

(1) Length of sequence of round trips to memory,

(2) QRQW.

Issue: nesting of spawns.

Some BFS Example conclusions

(1) Describe using simple nesting: for each vertex of a layer, for each of its edges... ;

(2) Since only single-spawns can be nested (reason beyond current presentation), for some cases (generally smaller degrees) nesting single-spawns works best, while for others flatening works better;

(3) Use nested spawn for improved development time and let compiler derive best implementation.

The Memory Wall

Concerns: 1) latency to main memory, 2) bandwidth to main memory.

Position papers: “the memory wall” (Wulf), “its the memory, stupid!” (Sites)

Note: (i) Larger on chip caches are possible; for serial computing, return on using them: diminishing. (ii) Few cache misses can overlap (in time) in serial computing; so: even the limited bandwidth to memory is underused.

XMT does better on both accounts:

- uses more the high bandwidth to cache.
- hides latency, by overlapping cache misses; uses more bandwidth to main memory, by generating concurrent memory requests; however, use of the cache alleviates penalty from overuse.

Conclusion: using PRAM parallelism coupled with IOS, XMT reduces the effect of cache stalls.

Memory architecture, interconnects

- High bandwidth memory architecture.
- Use hashing to partition the memory and avoid hot spots.
- Understood, BUT (needed) departure from mainstream practice.

- High bandwidth on-chip interconnects

- Allow infrequent global synchronization (with IOS).

Attractive: lower energy.

- Couple with strong MTCU for serial code.

Final thought: Created our own coherent planet

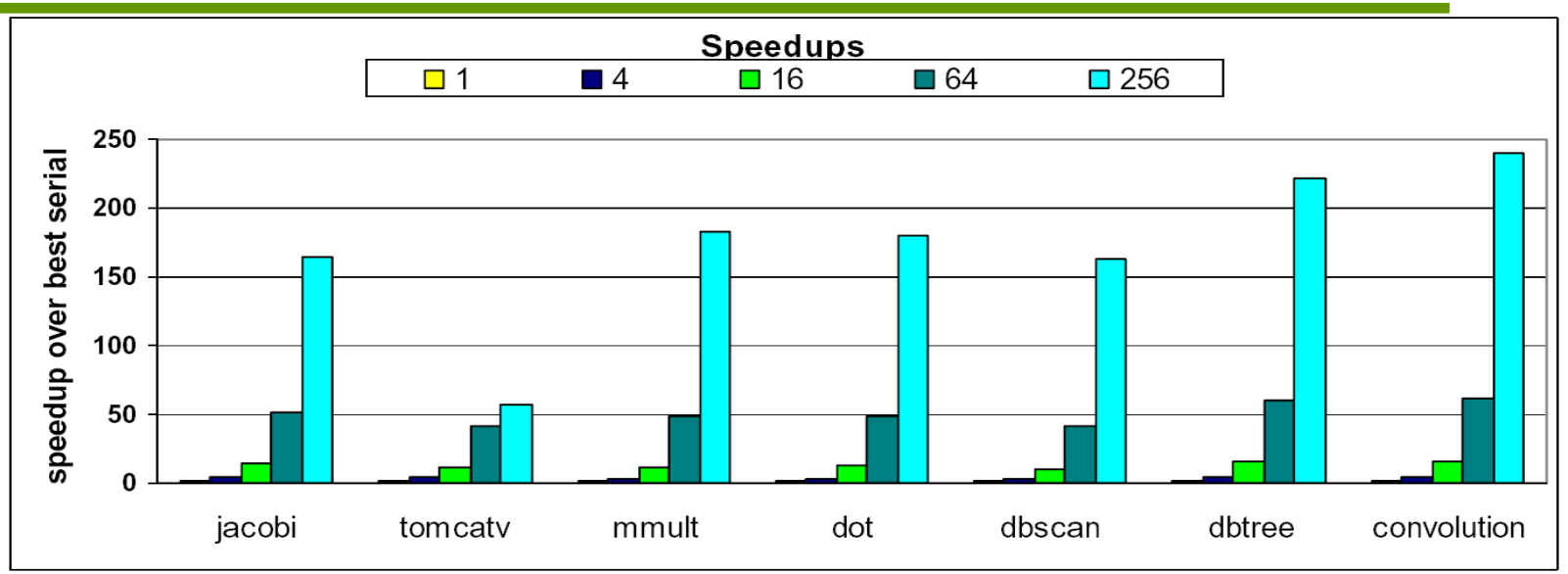
- When was the last time that a professor offered a (separate) algorithms class on own language, using own compiler and own computer?
- Colleagues could not provide an example since at least the 1950s. Have we missed anything?

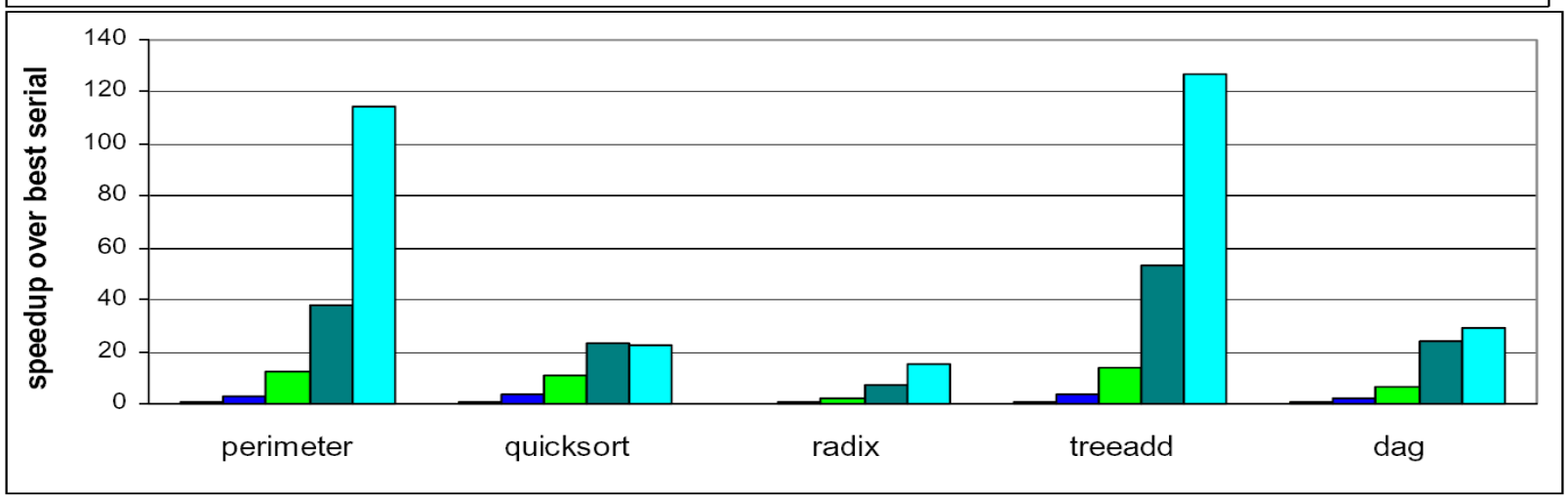
Teaching:

Class programming homework on par with serial algorithms class. In one semester: multiplication of sparse matrix by vector, deterministic general sorting, randomized sorting, Breadth-First Search (BFS), log-time graph connectivity and spanning tree.
In the past also: integer sorting, selection.

Consistent with claim that PRAM is a good alternative to serial RAM. Who else in parallel computing can say that?

Speed-up results from NNTV-03 Assumptions follow in 3 slides





Experimental Methodology

- Simulator
  - SimpleScalar parameters for instruction latencies
  - 1, 4, 16, 64, 256 TCUs
- Configuration:
  - 8 TCUs per cluster
  - 8K L1 cache
  - banked shared L2 cache 1MB
- Programs rewritten in XMT
  - Speedups of parallel XMT program compared to best serial program
    - parallel applications: scalability to high levels
    - speedups for less parallel, irregular applications

First Application Set

Domain	Program	source
Scientific Computation	1.jacobi	
	2.tomcatv	SPEC95
Linear Algebra	3.mmult	Livermore Loops
	4.dot	Livermore Loop
Database	5.dbscan	[]
	6.dbtree	MySQL
Image processing	7.convolution	[]

- Computation:
  - regular,
  - mostly array based,
  - limited synchronization needed

Second Application Set

Domain	Program	source
Sorting Algorithms	1.quicksort	
	2.radixsort	(SPLASH)
graph traversal	3.dag	
	4.treeadd	Olden
image processing	5.perimeter	Olden

- Computation:
  - irregular,
  - unpredictable
  - synchronization needed