The eXplicit MultiThreading (XMT) Parallel Computer Architecture

Next generation dektop supercomputing

Uzi Vishkin

UMIACS

University of Maryland Institute for Advanced Computer Studies

A. JAMES CLARK SCHOOL *of* ENGINEERING

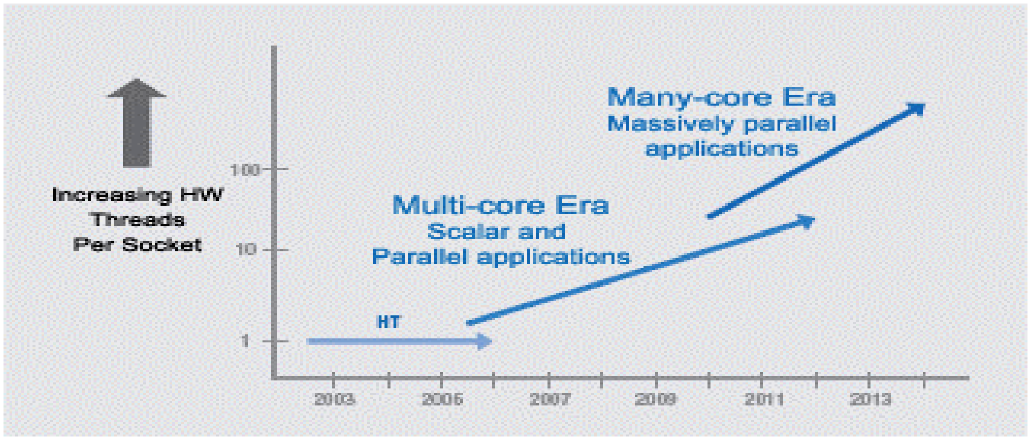
UNIVERSITY OF MARYLAND 18 56

Commodity computer systems

Chapter 1 1946→2003: Serial. 5KHz→4GHz.

Chapter 2 2004--: Parallel. #"cores": ~d^{y-2003}

Source: Intel Platform 2015
Date: March 2005



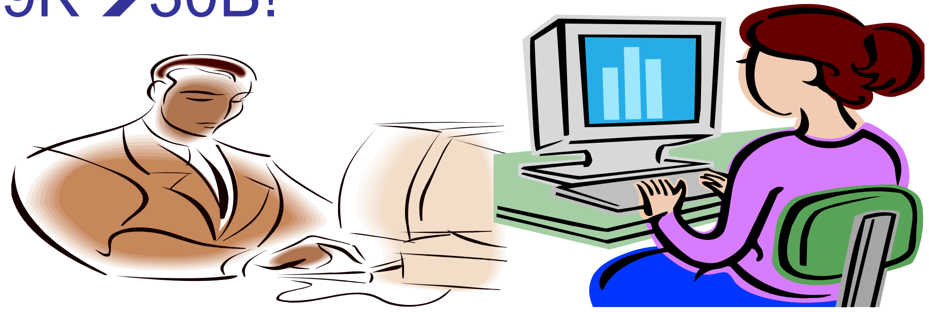
BIG NEWS:

Clock frequency growth: flat.

If you want your program to run significantly faster … you're going to have to parallelize it → Parallelism: only game in town

#Transistors/chip 1980→2011: 29K→30B!

Programmer's IQ? Flat..



The world is yet to see a successful general-purpose parallel computer: Easy to program & good speedups

2008 Impasse

All vendors committed to multi-cores. Yet, their architecture and how to program them for single task completion time not clear → SW vendors avoid investment in long-term SW development since may bet on the wrong horse. Impasse bad for business.

What about parallel programming education?

All vendors committed to parallel by 3/2005 → WHEN (not IF) to start teaching?

But, why not same impasse?

Can teach common things.

State-of-the-art: only the education enterprise has an actionable agenda! tie-breaker: isn't it nice that Silicon Valley heroes can turn to teachers to save them?

Need

A general-purpose parallel computer framework ["successor to the Pentium for the multi-core era"] that:

(i) is easy to program;

(ii) gives good performance with any amount of parallelism provided by the algorithm; namely, up- and down-scalability including backwards compatibility on serial code;

(iii) supports application programming (VHDL/Verilog, OpenGL, MATLAB) and performance programming; and

(iv) fits current chip technology and scales with it.

(in particular: strong speed-ups for single-task completion time)

Main Point of talk: PRAM-On-Chip@UMD is addressing (i)-(iv).

The Pain of Parallel Programming

- Parallel programming is currently too difficult

  To many users programming existing parallel computers is “as intimidating and time consuming as programming in assembly language” [NSF Blue-Ribbon Panel on Cyberinfrastructure].

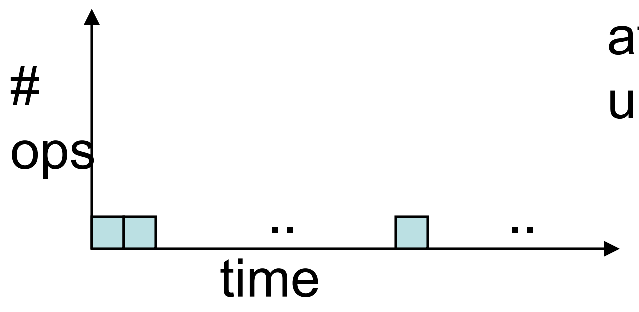
- J. Hennessy: “Many of the early ideas were motivated by observations of what was easy to implement in the hardware rather than what was easy to use” Reasonable to question build-first figure-out-how-to-program-later architectures.

- Lesson ➔ parallel programming must be properly resolved

Parallel Random-Access Machine/Model (PRAM)

Serial RAM Step: 1 op (memory/etc).
PRAM Step: many ops.

Serial doctrine

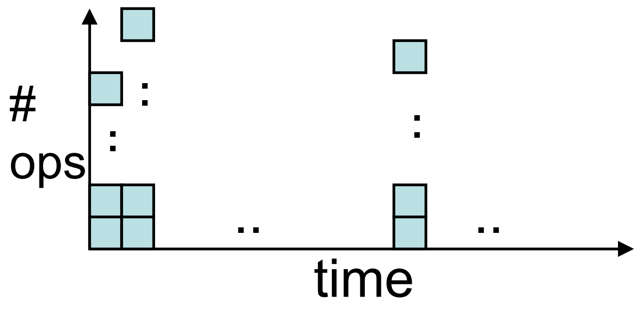


time = #ops

What could I do in parallel at each step assuming unlimited hardware

➔

Natural (parallel) algorithm



time << #ops

1979- : THEORY figure out how to think algorithmically in parallel
(Also, ICS07 Tutorial). "In theory there is no difference between theory and practice but in practice there is" ➔
1997- : PRAM-On-Chip@UMD: derive specs for architecture; design and build

Flavor of parallelism

Problem Replace A and B. Ex. A=2,B=5➔A=5,B=2.

Serial Alg: X:=A;A:=B;B:=X.   3 Ops. 3 Steps. Space 1.

Fewer steps (FS):
```
X:=A     |     B:=X
Y:=B     |     A:=Y          4 ops. 2 Steps. Space 2.
```

Problem Given A[1..n] & B[1..n], replace A(i) and B(i) for i=1..n.

Serial Alg:
```
For i=1 to n do
    X:=A(i);A(i):=B(i);B(i):=X   /*serial replace
```
3n Ops. 3n Steps. Space 1.

Par Alg1:
```
For i=1 to n pardo
    X(i):=A(i);A(i):=B(i);B(i):=X(i) /*serial replace in parallel
```
3n Ops. 3 Steps. Space n.

Par Alg2:
```
For i=1 to n pardo
    X(i):=A(i)   |   B(i):=X(i)
    Y(i):=B(i)   |   A(i):=Y(i)    /*FS in parallel
```
4n Ops. 2 Steps. Space 2n.

Discussion
- Parallelism requires extra space (memory).
- Par Alg 1 clearly faster than Serial Alg.
- Is Par Alg 2 preferred to Par Alg 1?

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.

<u>Basic algorithm</u>

```
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.

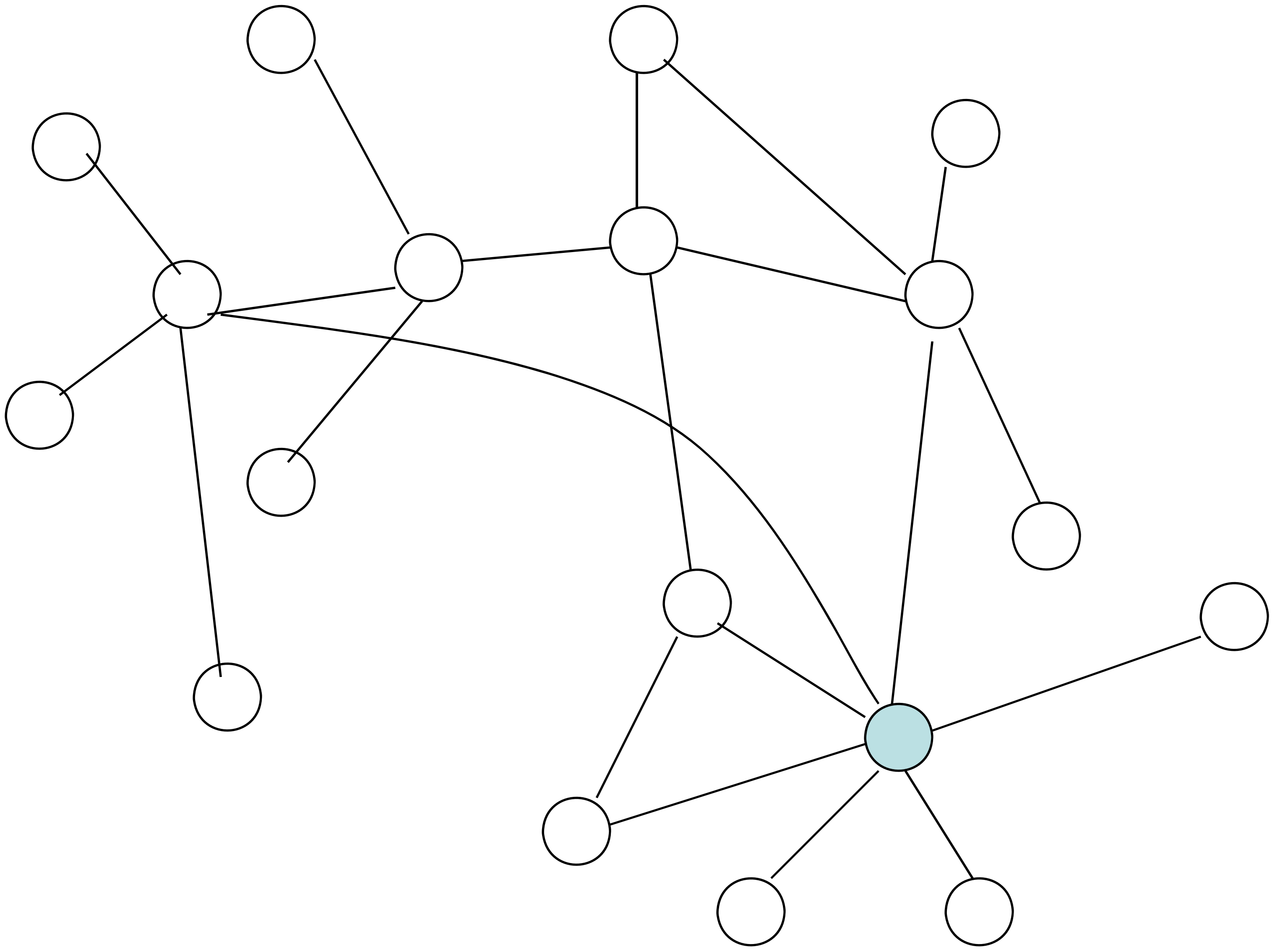
Gain relative to serial: (first cut) ~T/S!

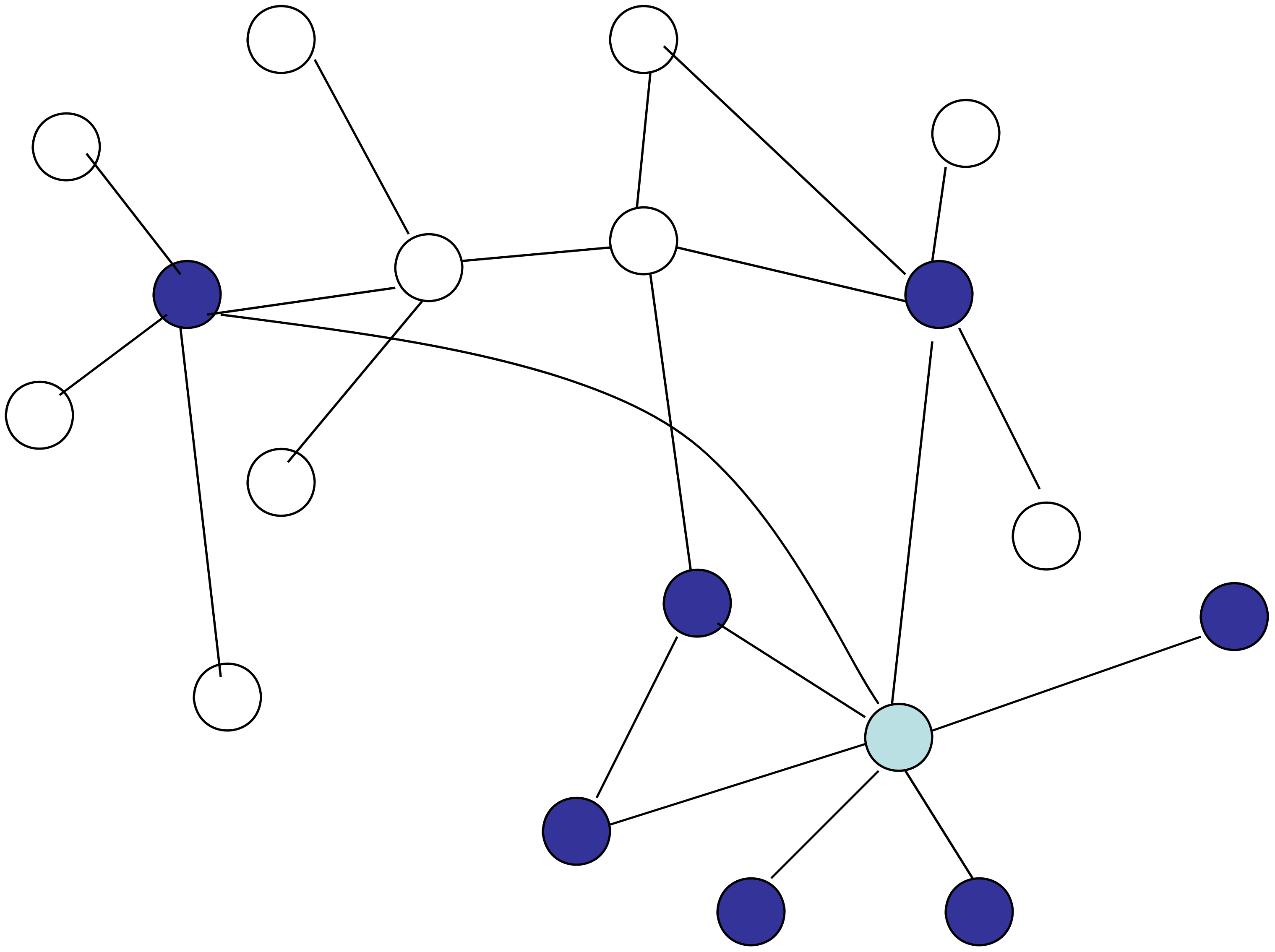
Decisive also relative to coarse-grained parallelism.

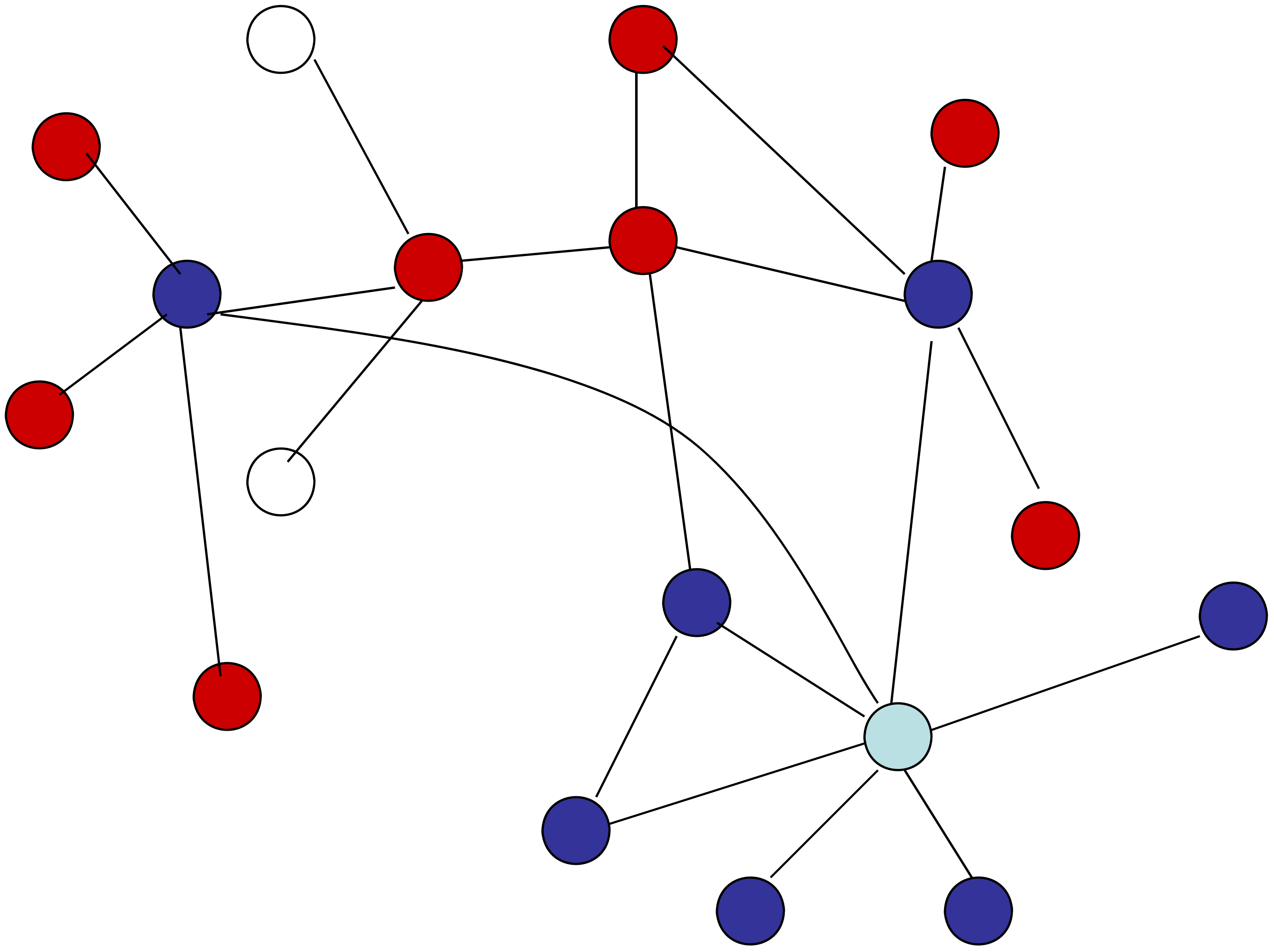
Note: (i) "Concurrently": only change to serial algorithm

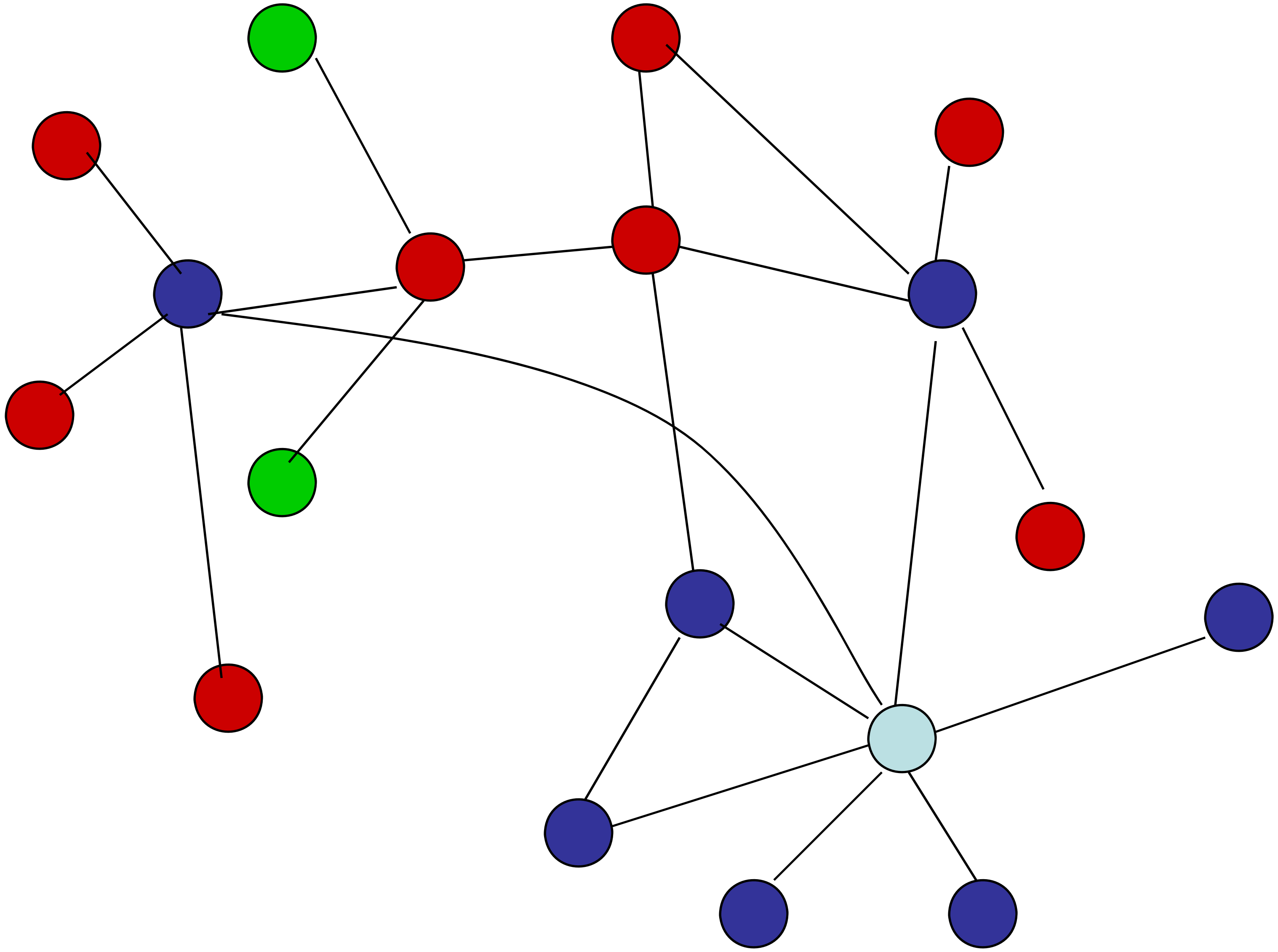
(ii) No "decomposition"/"partition"

KEY POINT: Mental effort of PRAM-like programming is considerably easier than for any of the computer currently sold. Understanding falls within the common denominator of other approaches.









The PRAM Rollercoaster ride

Late 1970's Theory work began

UP Won the battle of ideas on parallel algorithmic thinking. No silver or bronze!

Model of choice in all theory/algorithms communities. 1988-90: Big chapters in standard algorithms textbooks.

DOWN FCRC'93: "PRAM is not feasible". ['93+ despair → no good alternative! *Where vendors expect good* enough *alternatives to come from* in 2008?]

UP Highlights: eXplicit-multi-threaded (XMT) FPGA-prototype computer (not simulator), SPAA'07; ASIC tape-out of interconnection network, HotI'07.

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
  - Speed-of-light collides with 20+GHz serial processor. [Then came power ..]

  ➔Envisioned general-purpose chip parallel computer succeeding serial by 2010
- 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. XMT: enough bandwidth for on-chip interconnection network. [Balkan,Horak,Qu,V-HotInterconnects'07: 9mmX5mm, 90nm ASIC tape-out]

  One of several basic differences relative to "PRAM realization comrades": NYU Ultracomputer, IBM RP3, SB-PRAM and MTA.

➔PRAM was just ahead of its time.

Culler-Singh 1999: "Breakthrough can come from architecture if we can somehow…truly design a machine that can look to the programmer like a PRAM".

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?

<u>Spectrum of Explicit Multi-Threading (XMT) Framework</u>

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

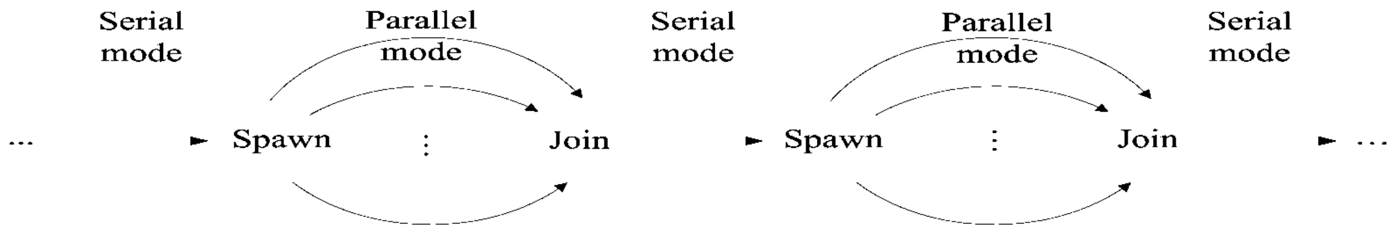
<u>Attributes</u>

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

How does it work

"Work-depth" Algs Methodology (source SV82) State all ops you can do in parallel. Repeat. Minimize: Total #operations, #rounds The rest is skill.

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



Programming methodology Algorithms → effective programs.

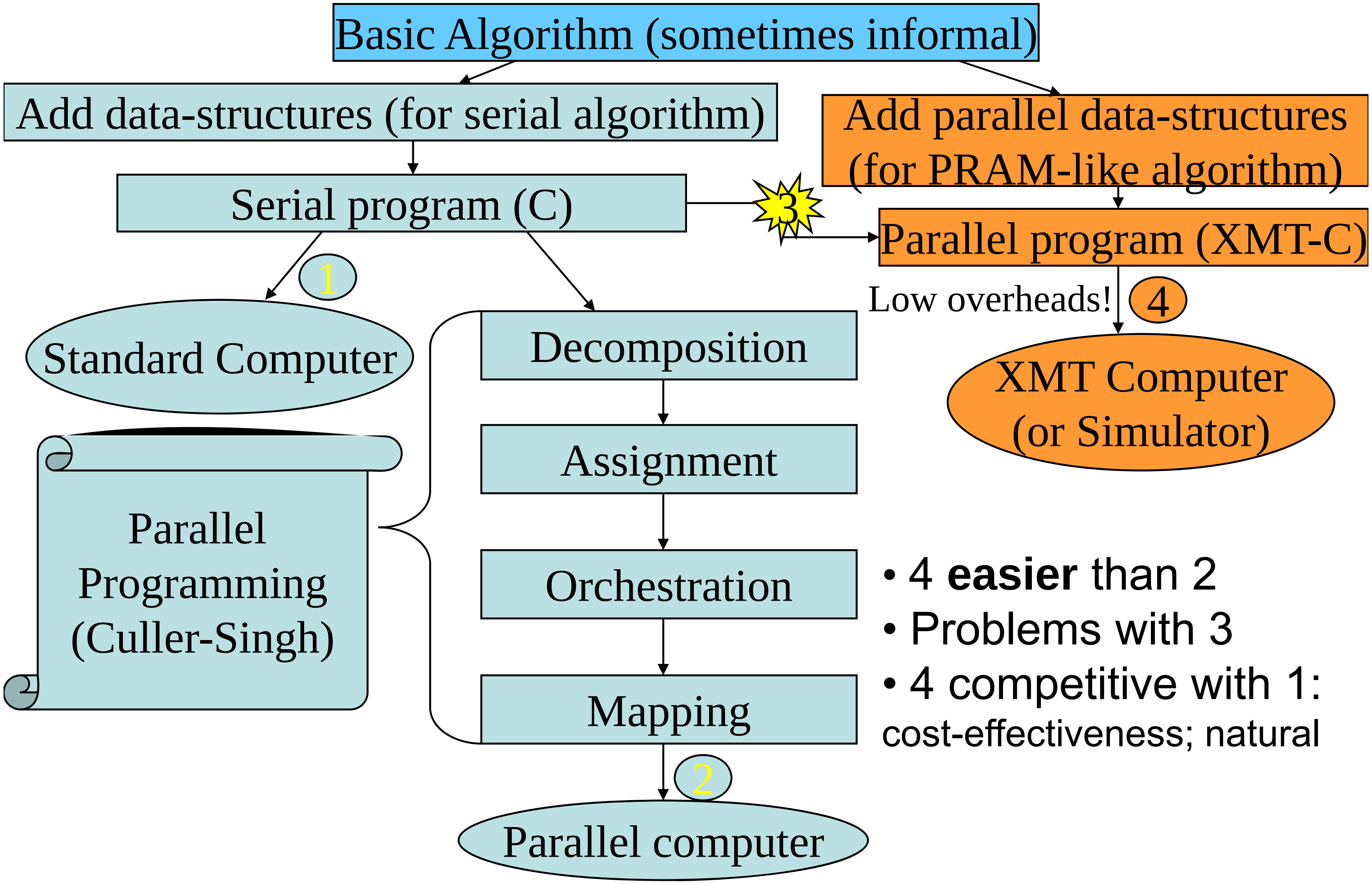
Extend the SV82 Work-Depth framework from PRAM to XMTC

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). [ideally: given XMTC program, compiler provides decomposition: "teach the compiler"]

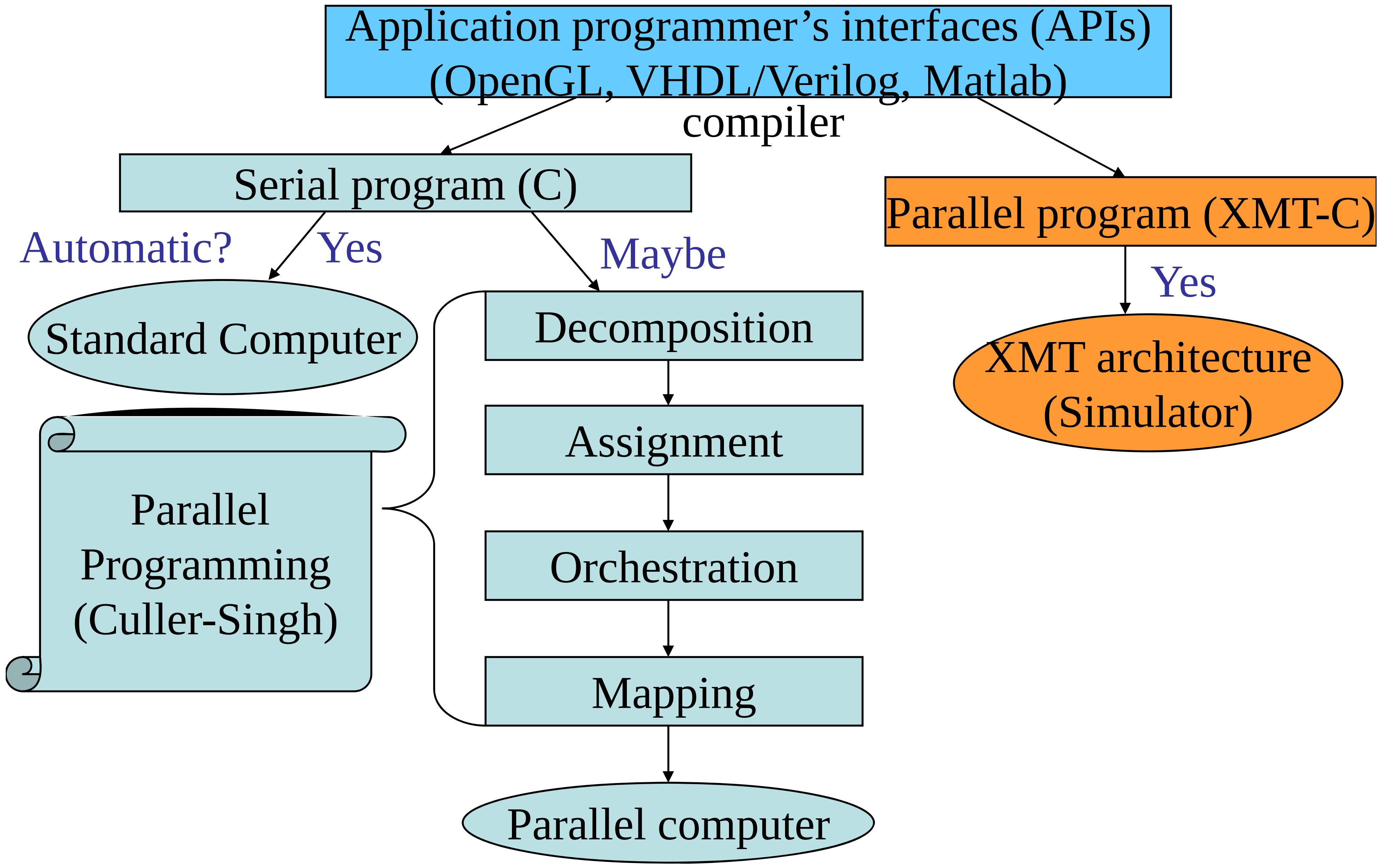
Architecture Dynamically load-balance concurrent threads over processors. "OS of the language". (Prefix-sum to registers & to memory.)

PERFORMANCE PROGRAMMING & ITS PRODUCTIVITY


Basic Algorithm (sometimes informal)
Add data-structures (for serial algorithm)
Add parallel data-structures (for PRAM-like algorithm)
Serial program (C)
3
Parallel program (XMT-C)
1
Low overheads!
4
Standard Computer
Decomposition
XMT Computer (or Simulator)
Assignment
Parallel Programming (Culler-Singh)
Orchestration
Mapping
2
Parallel computer

- 4 **easier** than 2
- Problems with 3
- 4 competitive with 1: cost-effectiveness; natural

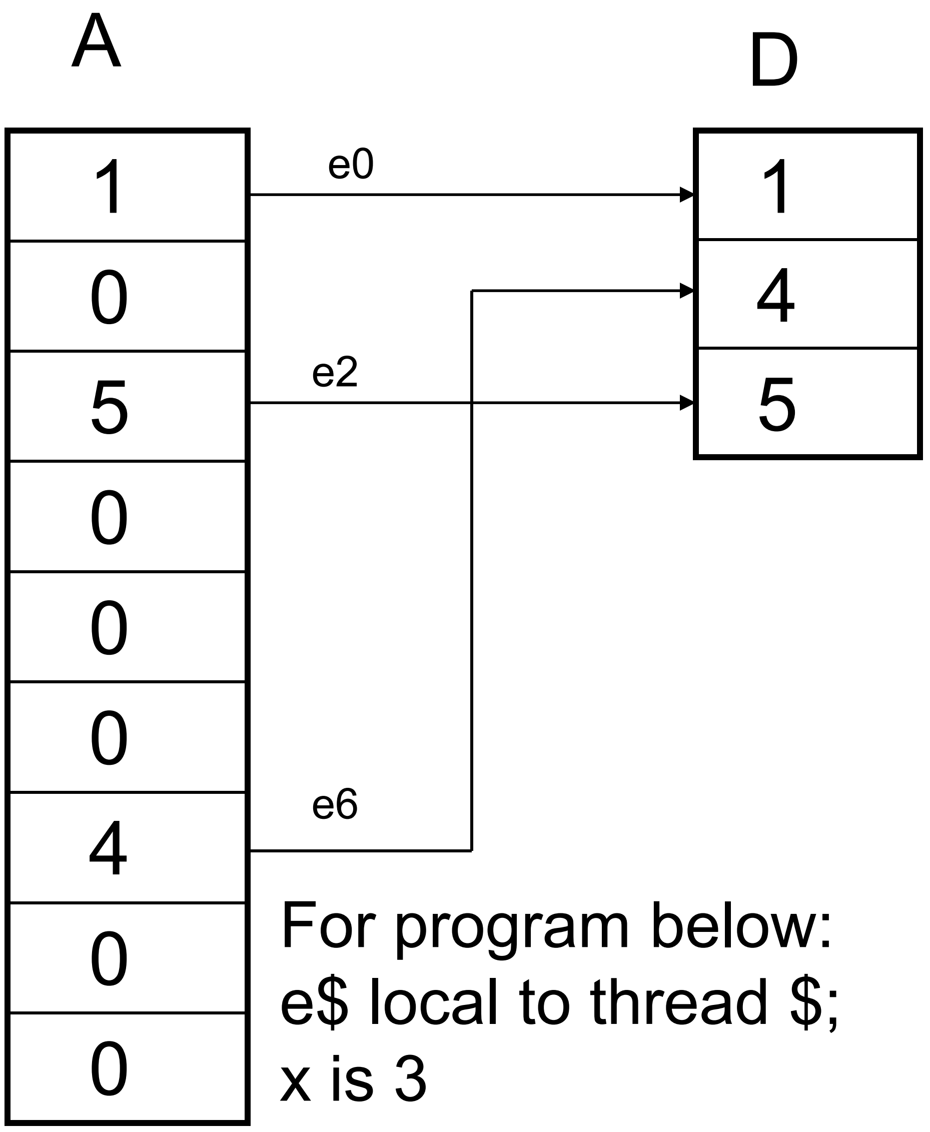
APPLICATION PROGRAMMING & ITS PRODUCTIVITY



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.



For program below:
e$ local to thread $;
x is 3

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.

<u>The PS multi-operand instruction</u>

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. Ideally compiler:

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

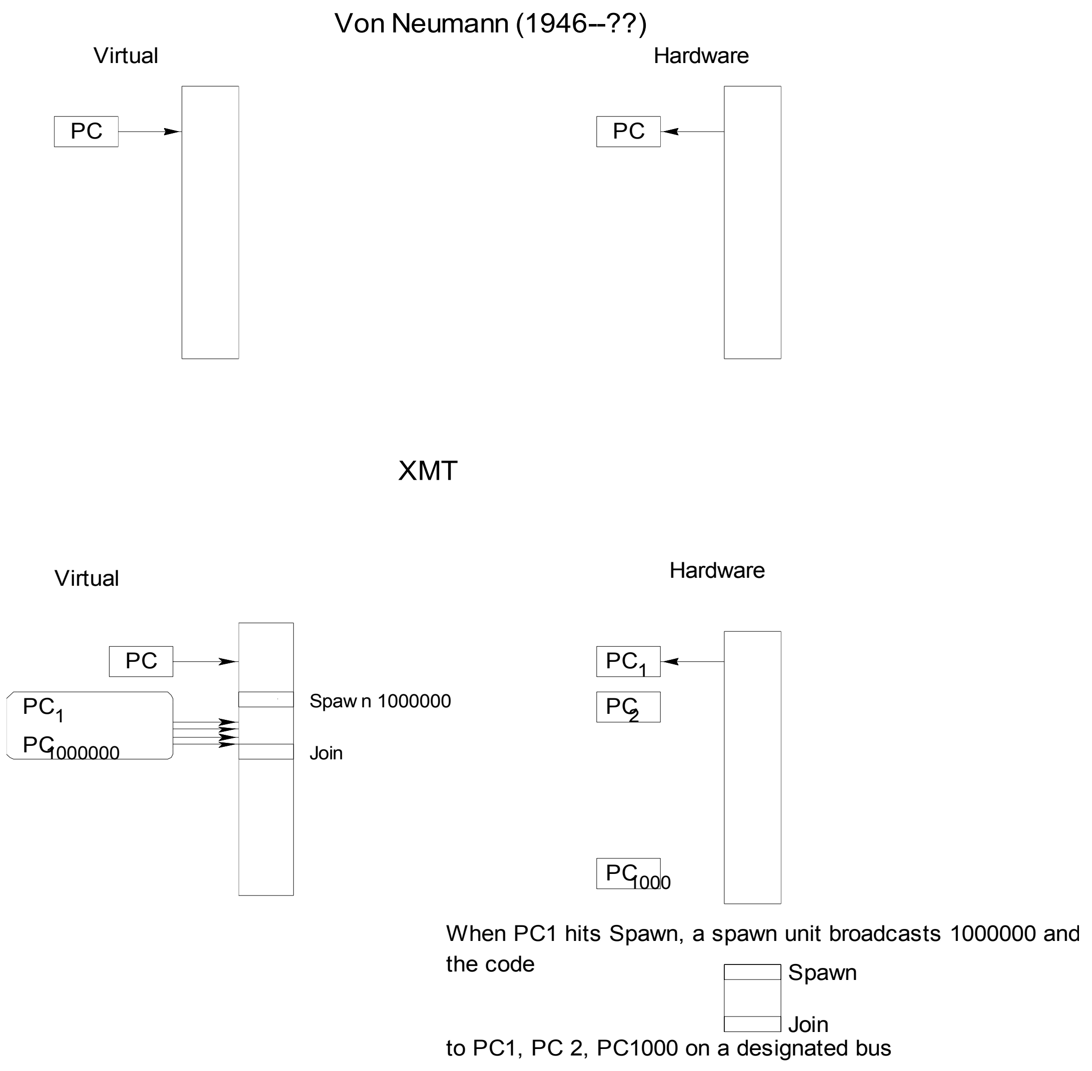
(1) Describe using simple nesting: for each vertex of a layer, for each of its edges... ;

(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 flattening works better;

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

How-To Nugget

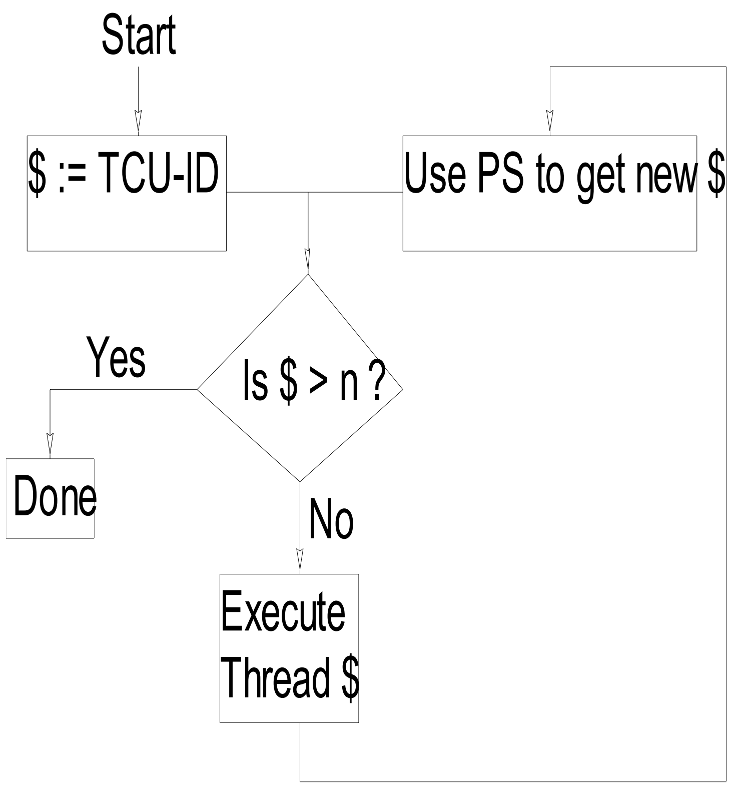
Seek 1st (?) upgrade of program-counter & stored program since 1946



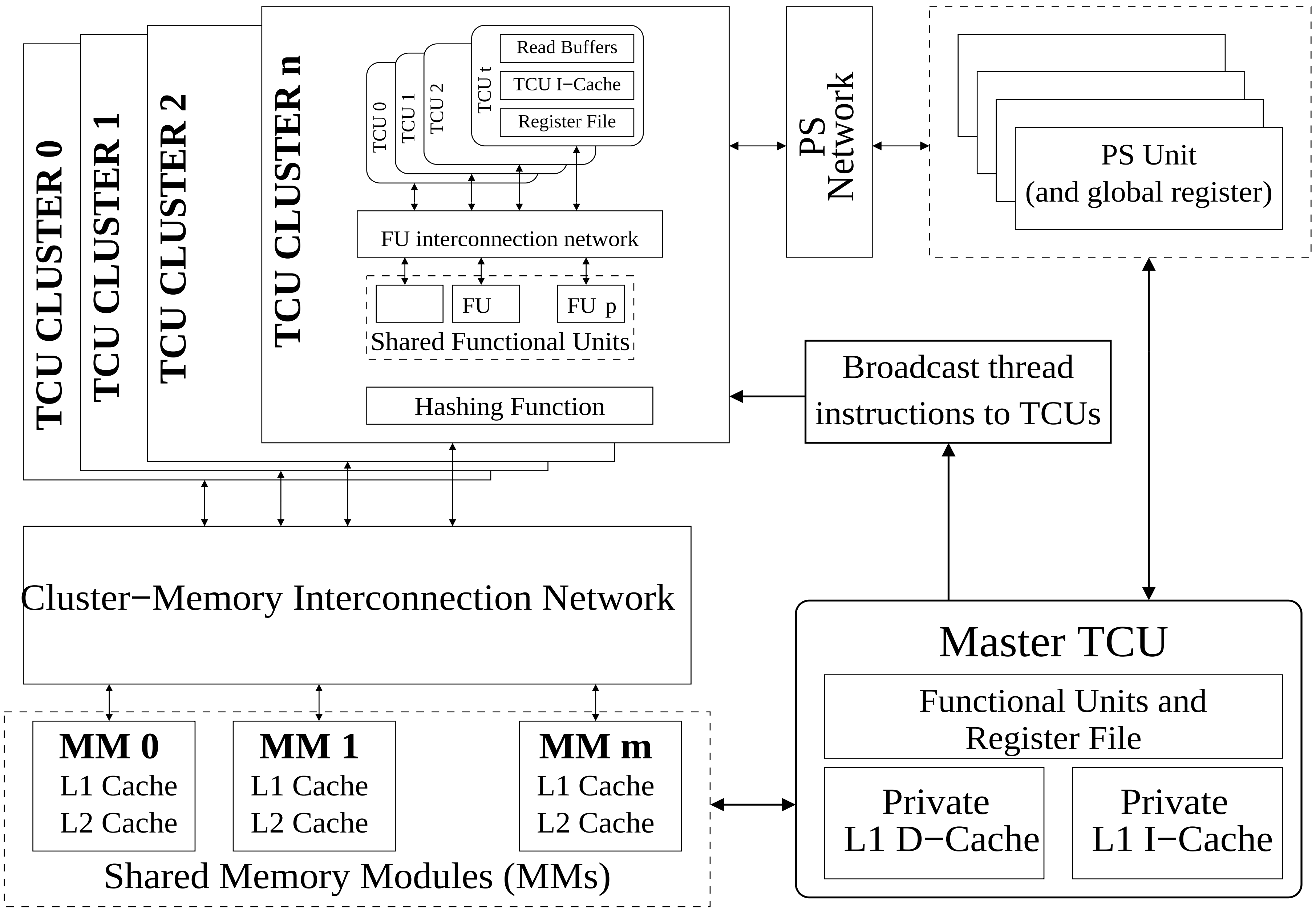
When PC1 hits Spawn, a spawn unit broadcasts 1000000 and the code

to PC1, PC 2, PC1000 on a designated bus

Virtual over physical: distributed solution



XMT Block Diagram – Back-up slide



ISA

- Any serial (MIPS, X86). MIPS R3000.
- Spawn (cannot be nested)
- Join
- SSpawn (can be nested)
- PS
- PSM
- Instructions for (compiler) optimizations

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.

XMT: An “UMA” Architecture

- Several current courses. Each has a text. The library has 1 copy. Should the copies be: (i) reserved at the library, (ii) reserved at a small library where the department is, or (iii) allow borrowing and then satisfy requests when needed
- Bandwidth, rather than latency, is the main advantage of XMT
- UMA seem counter-intuitive. Relax locality to make things equally far. However: (i) easier programming model; (ii) better scalability; cache coherence has issues (iii) off-chip bandwidth is adequate.
- Learning to ride a bike: you got to the take your feet off the ground to move faster.

Namely, with bandwidth-rich parallel system [bike] you got to relax (not abandon) locality [raise your feet] to move faster.

Some supporting evidence

Large on-chip caches in shared memory.

8-cluster (128 TCU!) XMT has only 8 load/store units, one per cluster. [IBM CELL: bandwidth 25.6GB/s from 2 channels of XDR. Niagara 2: bandwidth 42.7GB/s from 4 FB-DRAM channels.

With reasonable (even relatively high rate of) cache misses, it is really not difficult to see that off-chip bandwidth is not likely to be a show-stopper for say 1GHz 32-bit XMT.

PRAM-On-Chip Silicon

Specs and aspirations

n=m	64
# TCUs	1024

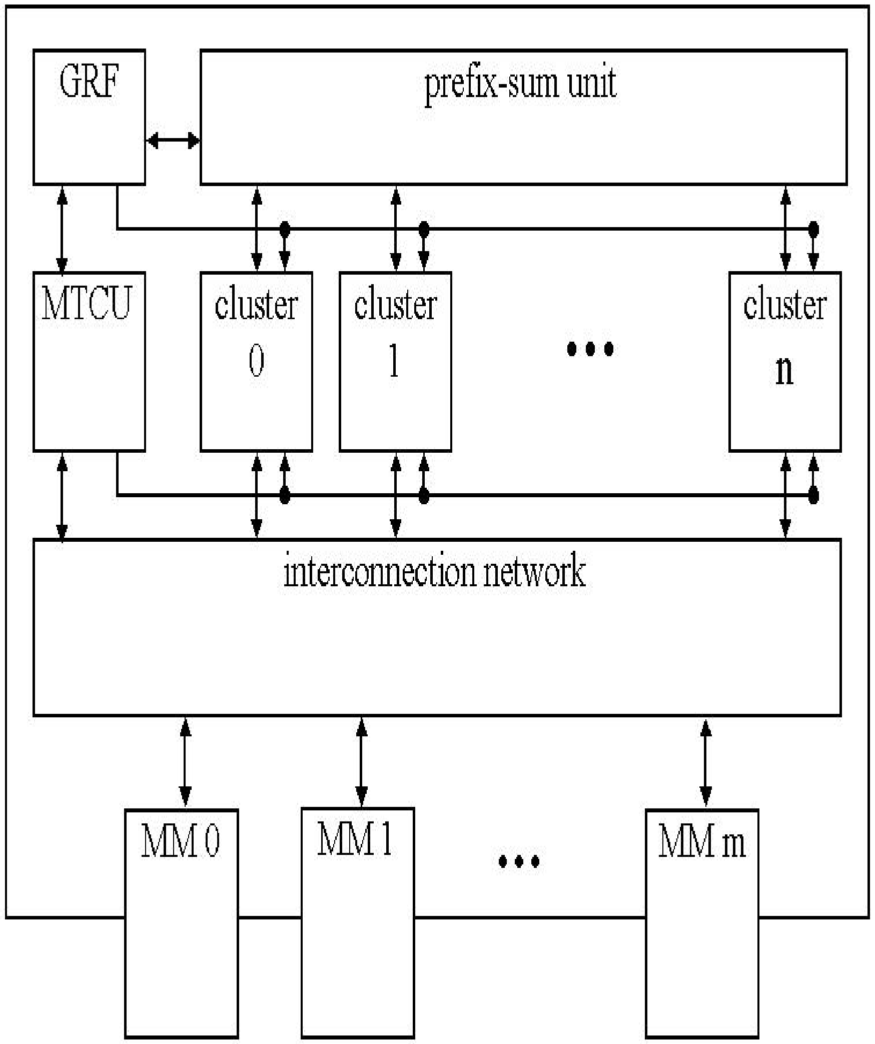
- Multi GHz clock rate
- Get it to scale to cutting edge technology
- Proposed answer to the many-core era: "successor to the Pentium"?

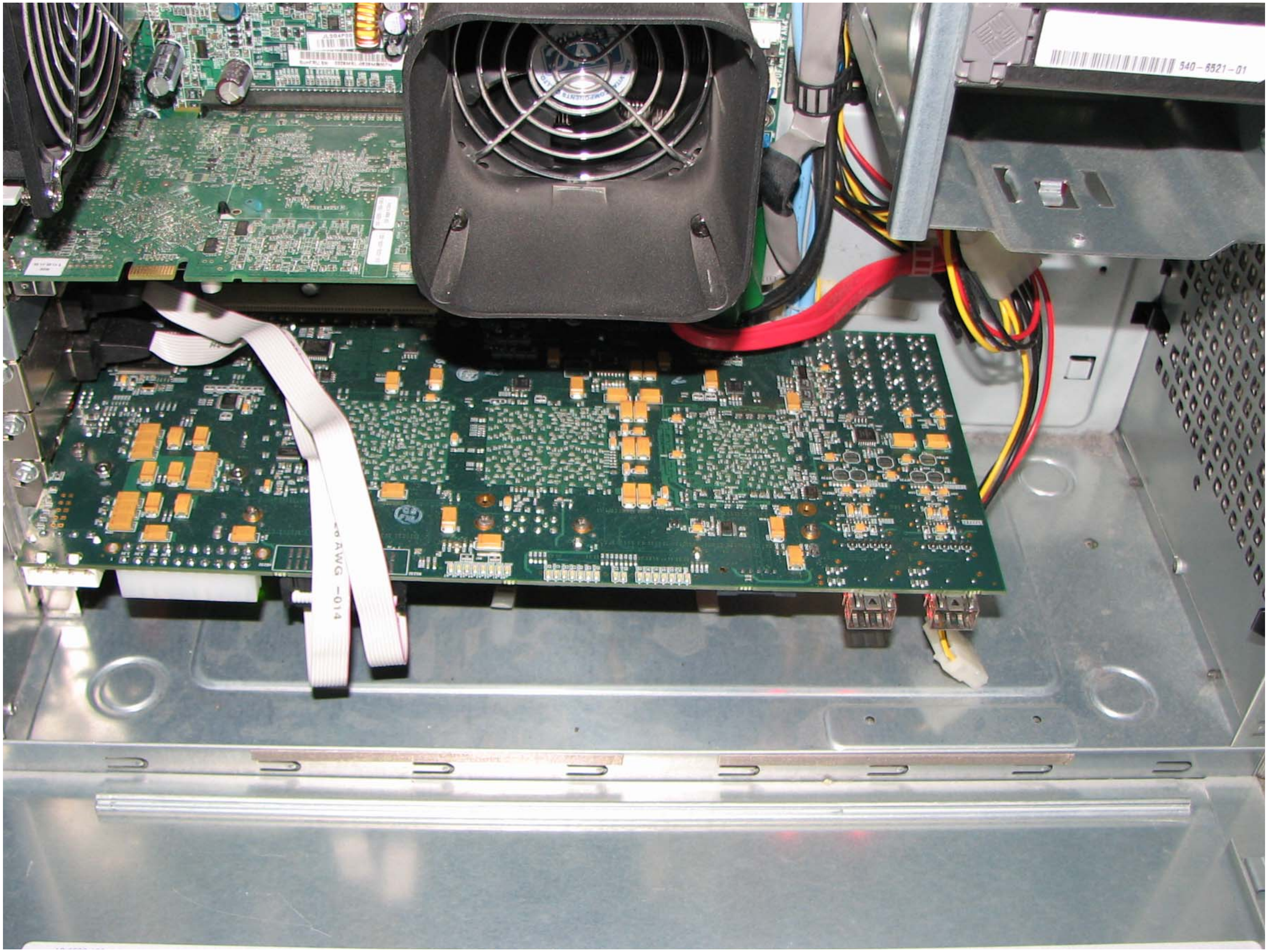
FPGA Prototype built n=4, #TCUs=64, m=8, 75MHz.

The system consists of 3 FPGA chips:
2 Virtex-4 LX200 & 1 Virtex-4 FX100(Thanks Xilinx!)

- Cache coherence defined away: Local cache only at master thread control unit (MTCU)
- Prefix-sum functional unit (F&A like) with global register file (GRF)
- Reduced global synchrony
- Overall design idea: no-busy-wait FSMs

Block diagram of XMT




540-6521-01
AWG -014

Some experimental results

- 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 QSort was 20M
- XMT enhancements: Broadcast, prefetch + buffer, non-blocking store, non-blocking caches.

XMT Wall clock time (in seconds)

App.	XMT Basic	XMT	Opteron
M-Mult	179.14	63.7	113.83
QSort	16.71	6.59	2.61

Assume (arbitrary yet conservative)

ASIC XMT: 800MHz and 6.4GHz/s

Reduced bandwidth to .6GB/s and projected back by 800X/75

XMT Projected time (in seconds)

App.	XMT Basic	XMT	Opteron
M-Mult	23.53	12.46	113.83
QSort	1.97	1.42	2.61

Experience with new FPGA computer

Included: basic compiler [Tzannes,Caragea,Barua,V].

New computer used: to validate past speedup results.

Spring'07 parallel algorithms graduate class @UMD

- Standard PRAM class. 30 minute review of XMT-C.
- Reviewed the architecture only in the last week.
- 6(!) significant programming projects (in a theory course).
- FPGA+compiler operated nearly flawlessly.

Sample speedups over best serial by students Selection: 13X. Sample sort: 10X. BFS: 23X. Connected components: 9X.

Students' feedback: "XMT programming is easy" (many), "The XMT computer made the class the gem that it is", "I am excited about one day having an XMT myself! "

11-12,000X relative to cycle-accurate simulator in S'06. Over an hour ➔ sub-second. (Year➔46 minutes.)

Experience (cont'd)

Fall'07 Informal Course to High Schoo students

- Dozen students: 10 MB, 1 TJ, 1 WJ.
- Motivated. Capable. BUT: 1-day tutorial. Follow-up with 1 weekly office hour by undergrad TA.
- Some (2 10th graders) 8 programming assignments, including 5 of 6 in grad class.

Conjecture: Professional teacher, 1-hour/day, 2 months can teach general above-average HS students

Spring'08 General UMD-Honors course

- How will programmers have to think by the time you graduate.

Spring'08 Senior-Year parallel algorithms course

First time: 14 students

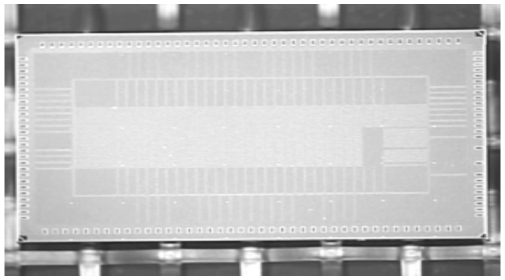
XMT architecture and ease of implementing it

Single (hard working) student (X. Wen) completed synthesizable Verilog description AND the new FPGA-based XMT computer (+ board) in slightly more than two years. No prior design experience.

➔faster time to market, lower implementation cost.

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 tapeout



Bare die photo of 8-terminal interconnection network chip IBM 90nm process, 9mm x 5mm fabricated (August 2007)

  - 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 tapeout this year? 4-5 grad students working

XMT Development (cont'd)

- Compiler Done: Basic. To do: Optimizations. Match HW enhancement.
  - Basic, yet stable, compiler completed
  - Under development: 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.

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

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

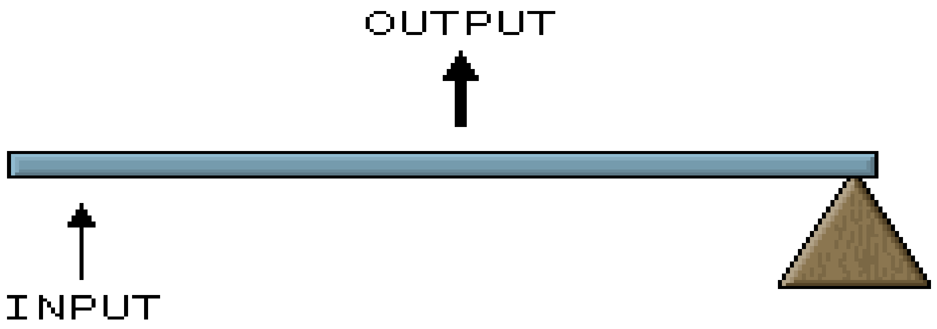
None has a competitive parallel programming model, or supports a broad range of APIs

- Streaming: XMT can emulate (using prefetch). 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.

Streaming&cell: some nice speed-ups.

Summary of technical pathways (Revisit)

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?

Positive record

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

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, M.N. Horak, G. Qu, and U. Vishkin. Layout-Accurate Design and Implementation of a High-Throughput Interconnection Network for Single-Chip Parallel Processing. Hot Interconnects, Stanford, CA, 2007.

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 proto-typing 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 (Eds), Handbook of Parallel Computing, CRC Press. To appear. 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, Jan. 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 as well as odd and ends

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)

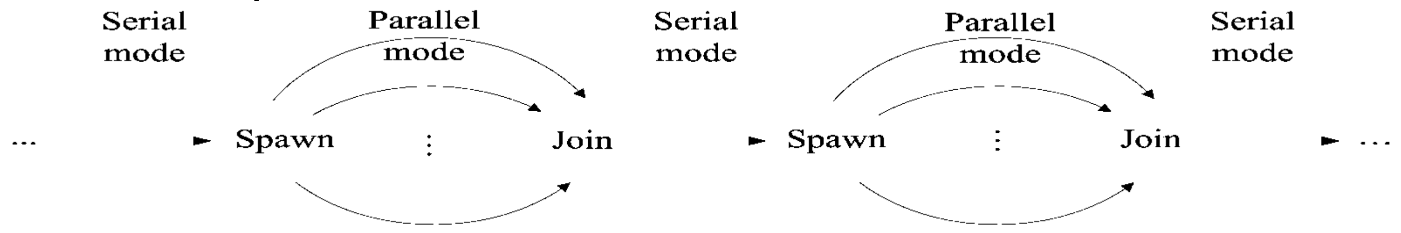
- PRAM Theory
  - Assume latency for arbitrary number of memory accesses is the same as for one access.
  - Full overheads model (like serial RAM).
  - Model of choice for parallel algorithms in all major algorithms/theory communities. No real competition!
  - Main algorithms textbooks included PRAM 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

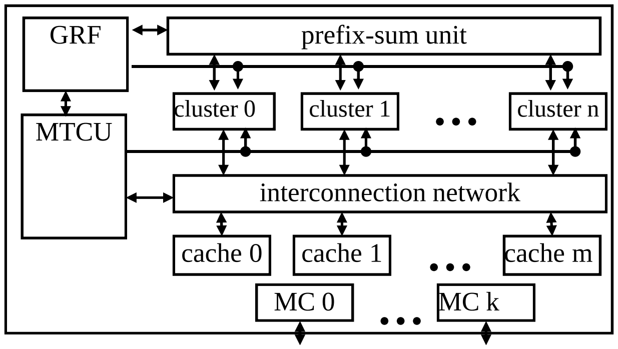
AMD Opteron 2.6 GHz, RedHat Linux Enterprise 3, 64KB+64KB L1 Cache, 1MB L2 Cache (none in XMT), memory bandwidth 6.4 GB/s (X2.67 of XMT)

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

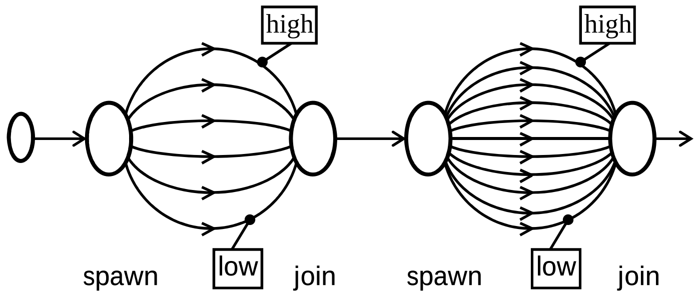
Note: First commitment to silicon. “We can build”.

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

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.

Final thought: Created our own coherent planet

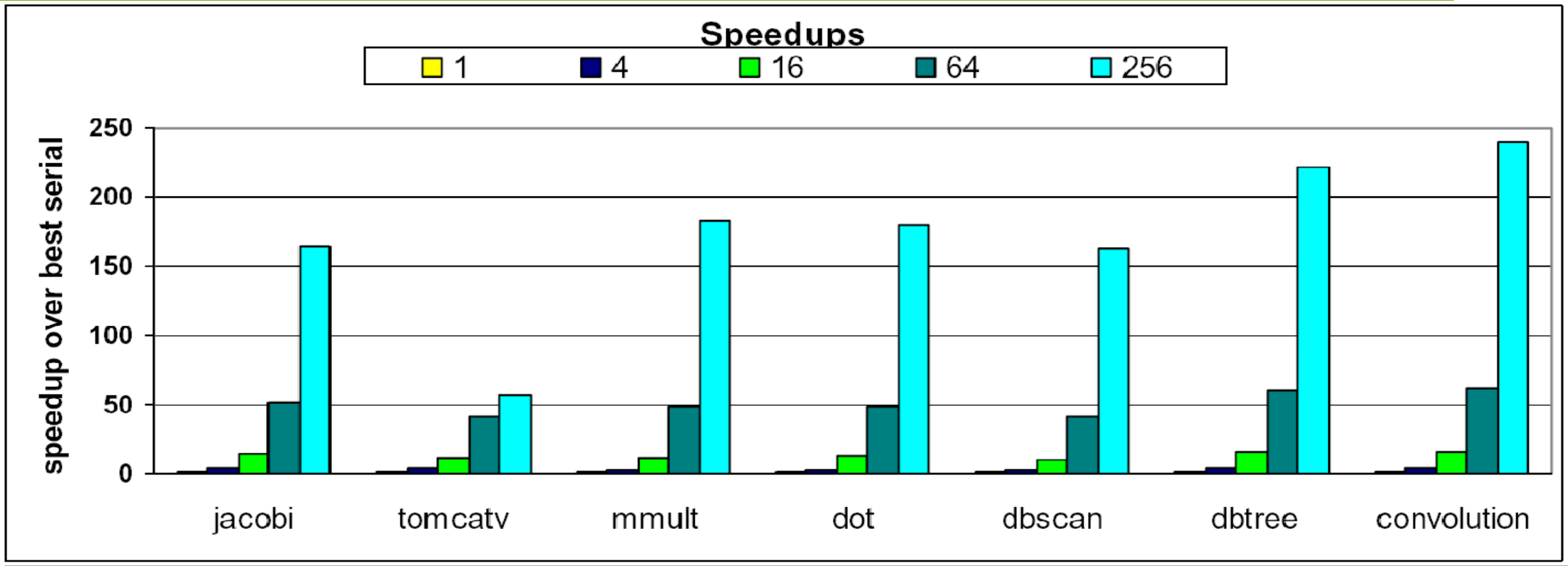
- When was the last time that a professor offered a (separate) algorithms class on own language, using own compiler and own computer?
- Colleagues could not provide an example since at least the 1950s. Have we missed anything?

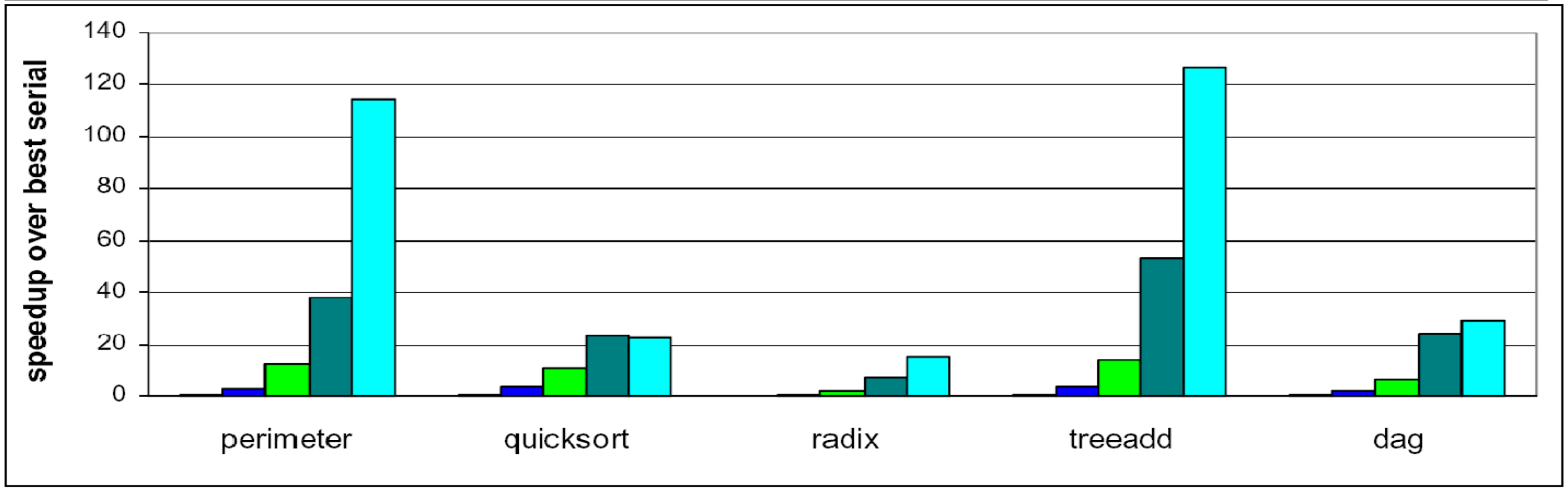
<u>Teaching</u>:

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

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”

Questions to profs and other researchers

Why continue teaching only for yesterday's serial computers? Instead:

1. Teach parallel algorithmic thinking.
2. Give PRAM-like programming assignments.
3. Have your students' compile and run remotely on our FPGA machine(s) at UMD.

Compare with (painful to program) decomposition step in other approaches.

Will you be interested in:

- Such reaching
- Open source access to compiler
- Open source access to hardware (IP cores)

Please let me know: vishkin@umd.edu