

The eXplicit MultiThreading (XMT) Easy-To-Program Parallel Computer: A PRAM-On-Chip Proof-of-Concept

Uzi Vishkin

1. PRAM-On-Chip is a **Legitimate contender** for defining **general-purpose many-core computing** (XMT + 2010s GPU replace past Pentium+GPU).
2. **NSF** needs to weigh in for HW diversity and true natural selection: have **industry-grade HW** available for research, education & comparison

XMT home page: www.umiacs.umd.edu/users/vishkin/XMT



A. JAMES CLARK SCHOOL *of* ENGINEERING

Abstract: As the heart of the field is being reinvented for parallelism the main technical challenge is timely convergence to an easy-to-program highly scalable many-core platform. The wealth of the parallel random-access machine/model (PRAM) theory of algorithms is well documented. The University of Maryland (UMD) explicit Multi-Threading (XMT) project has been driven by a PRAM-On-Chip vision, seeking to build an easy-to-program parallel computer comprising thousands of processors on a single chip, using a PRAM-like programming model. XMT has gone through extensive hardware (64-processor machine) and software prototyping. A software release allows experimentation with the XMT environment on any standard computer platform.

Interestingly, starting with a PRAM might not have been an obvious choice. Technology constraints guide us away from tightly coupled concurrency in programs; e.g., away from the PRAM and towards multi threading. On the other hand, relaxed concurrency in programs is notoriously difficult to design or analyze for correctness or performance. The XMT programming approach incorporates an elegant workaround. First, XMT provides a (refinement) ***“work flow” from a PRAM algorithm to an XMT program, and even to fine-tuning an XMT program*** for performance. Given a problem, a PRAM style parallel algorithm is developed for it using the Shiloach-Vishkin 1982 Work-Depth (WD) Methodology. All the operations that can be concurrently performed in the first round are noted, followed by those that can be performed in the second round, and so on. Such synchronous description of a parallel algorithm makes it easy to reason about correctness and analyze for work (the total number of operations) and depth (number of rounds). The XMT programmer is then expected to use the XMTC language (basically C with two additional commands) for producing a multi-threaded program. Reasoning about correctness or performance can **now be restricted to just comparison of the program with the WD algorithm, assuming that correctness and performance of the algorithm have been established, often a much easier task than directly analyzing the program.** This workaround allowed college freshmen and even high-school students solve the same problems they get in typical freshmen serial programming course assignments using (XMT) parallel programming.

Our proof-of-concept demonstrates that the limited diversity of many-core vendor hardware may miss an important opportunity for the timely convergence sought. Theory can play a key role for this or other opportunities.

Commodity computer systems

1946 → 2003 General-purpose computing: **Serial**. 5KHz → 4GHz.

2004 **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**

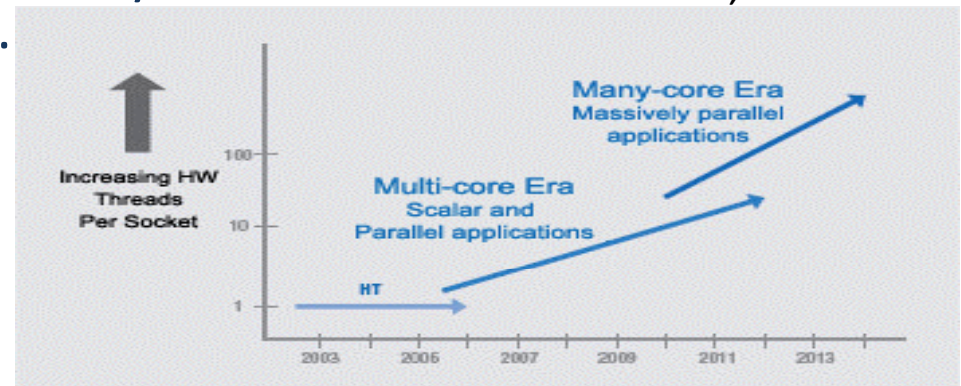
#Transistors/chip 1980 → 2011: 29K → 30B! General-purpose computing goes **Parallel**. #“cores”: $\sim d^{y-2003}$

But, what about the programmer?

Systems communities claim to be objectively guided by “the quantitative approach”.

Yet, they “forget” to quantify or benchmark the human factor. To whom can an approach be taught: graduate or middle-school students? Development time?

Intel Platform 2015, March05



40 years of parallel computing **Never** a successful general-purpose parallel computer: **Easy to program & good speedups**

Letter grade from NSF Blue-Ribbon Panel on Cyberinfrastructure: **F**.

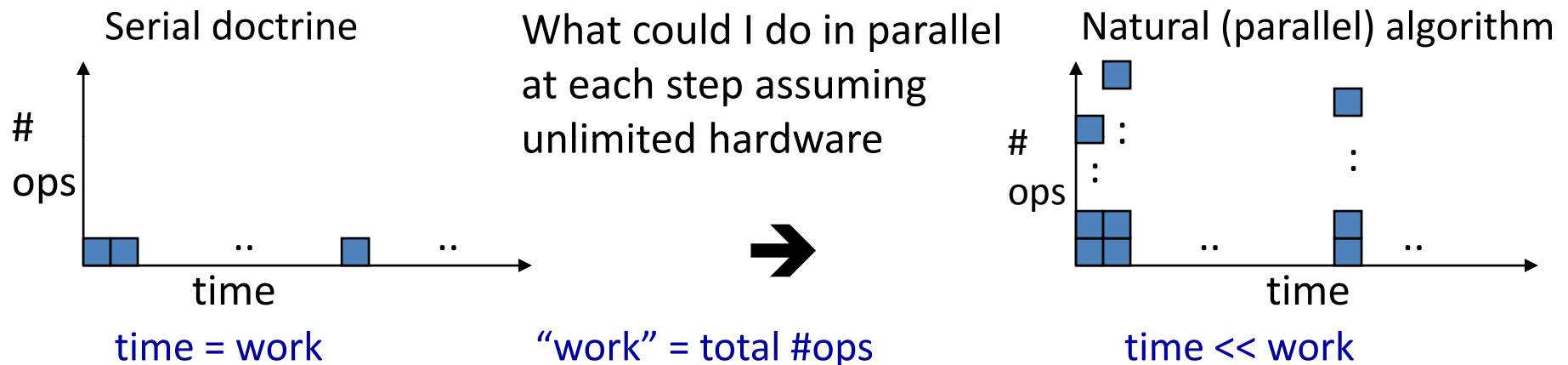
To many users programming existing parallel computers is “as intimidating and **time consuming as programming in assembly language**”.

What would be today's grade? If higher, is this a game changer?

Are theorist the canaries in the coal mine, whose absence is a worrying sign?

2 Paradigm Shifts

- Serial to parallel widely agreed.
- Within parallel Existing “decomposition-first” paradigm. Too painful to program. Hence: Express only “what can be done in parallel” (PRAM: Parallel Random-Access Model). Build machine around this.



Late 1970s- : THEORY: figure out **how to think algorithmically in parallel**

1997- : PRAM-On-Chip@UMD: derive **specs for architecture; design and build**

2 premises: (i) parallel algorithmic thinking. (ii) Specs first; contrast with:

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**”.

Implies: parallel HW followed **“build-first figure-out-how-to-program-later”**.”

Pre many-core parallelism: 3 main trusts

Improving single-task completion time for general-purpose parallelism was **not** the primary target of parallel machines:

1. Application-specific: e.g., computer graphics. Limiting origin. GPUs: great performance if you figure out how. Example: no interaction between threads; what to do with textbook parallel graph algorithms?
2. Parallel machines for high-throughput (of serial programs); hence, cache coherence, SMP, DSM. Only choice for “HPC” → Language standards, but many issues, e.g., **F** grade. Heard from HW designers (that dominate vendors): YOU figure out how to program (their machines) for locality.

→ Nothing fundamentally new in HW since 1990s. Serious lack of parallel machine diversity. What can a non-HW designers do?!

HW is like nature for Physics (Are vendors .. Gods of CS?)

Theory Always had its eyes on the ball. Started with a clean slate targeting single task completion time for general-purpose parallel computing.

3. PRAM and its extensive algorithmic theory. As simple as it gets. Ahead of its time: avant-garde. 1990s Common wisdom (LOGP): never implementable. Well: **we built it**.. Showed 100x speedups for 1000 processors. Also: taught to **grad students, seniors, freshmen, HS (&MS)**. 😊 **humans to humans**. Validated understanding & performance with programming assignments. Problems on par with serial courses. Students see immediate speedups.

Welcome to the 2009 Impasse

All vendors committed to multi-cores. Yet, their architecture and how to program them for single program completion time not clear

- The **software spiral** (HW improvements → SW imp → HW imp) – growth engine for IT (A. Grove, Intel); Alas, now broken!
- SW vendors avoid investment in long-term SW development since may bet on the wrong horse. Impasse bad for business.

For current students: Does CS&E degree mean: being trained for a 50yr career dominated by parallelism by programming yesterday's serial computers? **How can the same impasse & need to serve current students be mitigated for education?**

Answer “What can be done in parallel” common cognition for all approaches → **Can teach PRAM common denominator. The education enterprise has an actionable agenda for a time critical need. Isn't ours and the NSF's duty to act on it?**

Comments: 1. Is this a tie-breaker among approaches?

2. A machine is not easy-to-program if it is not easy-to-teach → education for parallelism has become a key **benchmark. Namely, **for parallelism, education is CS research.****

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 (grain or regularity) 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)

Key point: PRAM-On-Chip@UMD is addressing (i)-(iv).

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 2009?*]. **Device changed it all with #transistors on-chip:**

UP Highlights: eXplicit-multi-threaded (XMT) FPGA-prototype computer (not simulator): SPAA'07, CF'08; 90nm ASIC tape-outs: int. network, HotI'07, **XMT**. **1000 processors can fit on a single chip by mid-2010s.**

But, how come? Crash "course" on parallel computing

How much processors-to-memories bandwidth?

Enough: Ideal Programming Model (PRAM)

Limited: Programming difficulties

What else changed since the 1990s?

“Multi-Core Interconnects: Scale-Up or Melt-Down”

Panel discussion, Hot Interconnects, 2007, Stanford University

- Panel abstract: *As we anticipate 32, 64, 100+ processors on a single chip, the problem of interconnecting the cores looms as a potential showstopper to scaling. Are we heading for the cliff here, or will our panelists bring visions of interconnect architectures, especially those that work on-chip but not between chips, that will enable the scaling to continue?* Panelists from Google, Yahoo, others.

Summary Noted several issues with power consumption of multi-core architectures coming from industry:

- high power consumption of wide communication buses needed to implement cache coherence;
- basic nm complexity of cache coherence traffic (given n cores and m invalidations) and its implied huge toll on inter-core bandwidth; and
- high power consumption needed for a tightly synchronous implementation in silicon used in these designs.

Panel's conclusion: the industry must first converge to an easy-to-program highly-scalable multi-core architecture. These issues should be addressed in the context of such an architecture.

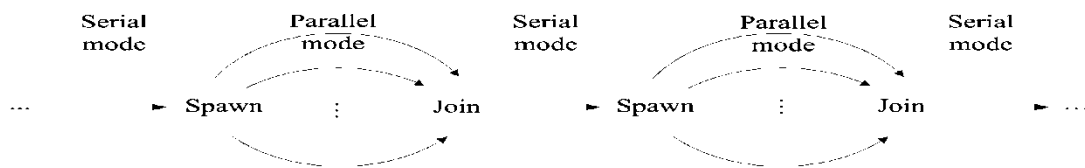
How does it work, and what should people know to participate

“Work-depth” Alg Methodology (SV82) State all ops you can do in parallel. Repeat.

Minimize: Total #operations, #rounds. Note: 1 The rest is skill. 2. Sets the algorithm

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

Unique First parallelism. Then decomposition



Legend: Level of abstraction Means

Means: Programming methodology Algorithms → effective programs.

Extend the SV82 Work-Depth framework from PRAM-like to XMTC

[Alternative Established APIs (VHDL/Verilog,OpenGL,MATLAB) “win-win proposition”]

Performance-Tuned Program minimize length of sequence of round-trips to memory + QRQW + Depth; take advantage of architecture enhancements (e.g., prefetch).

Means: Compiler: [ideally: given XMTC program, compiler provides decomposition: tune-up manually → “teach the compiler”]

Architecture HW-supported run-time load-balancing of concurrent threads over processors. Low thread creation overhead. (Extend classic stored-program+program counter; cited by 15 Intel patents; Prefix-sum to registers & to memory.)

All Computer Scientists will need to know >1 levels of abstraction (LoA)

CS programmer’s model: WD+P. CS expert : WD+P+PTP. Systems: +A.

Merging Example for Algorithm & Program

Input: Two arrays $A[1..n]$, $B[1..n]$; elements from a totally ordered domain S . Each array is monotonically non-decreasing.

Merging: map each of these elements into a monotonically non-decreasing array $C[1..2n]$

The partitioning paradigm

n : input size for a problem. Design a 2-stage parallel algorithm:

1. Partition the input into a large number, say p , of independent small jobs AND size of the largest small job is roughly n/p .
2. Actual work - do the small jobs concurrently, using a separate (possibly serial) algorithm for each.

Merging algorithm (cont'd)

Serial Merging algorithm

SERIAL – RANK(A[1 . .];B[1. .])

Starting from A(1) and B(1), in each round:

1. compare an element from A with an element of B
2. determine the rank of the smaller among them

Complexity: $O(n)$ time (and $O(n)$ work...)

“Surplus-log” parallel algorithm for Merging/Ranking

for $1 \leq i \leq n$ pardo

- Compute RANK(i,B) using standard binary search
- Compute RANK(i,A) using binary search

Complexity: $W=(O(n \log n), T=O(\log n)$

Linear work parallel merging: using a single spawn

Stage 1 of algorithm: Partitioning for $1 \leq i \leq n/p$ pardo [$p \leq n/\log$ and $p \mid n$]

- $b(i) := \text{RANK}(p(i-1) + 1, B)$ using binary search
- $a(i) := \text{RANK}(p(i-1) + 1, A)$ using binary search

Stage 2 of algorithm: Actual work

Observe Overall ranking task broken into $2p$ independent “slices”.

Example of a slice

Start at $A(p(i-1) + 1)$ and $B(b(i))$.

Using serial ranking advance till:

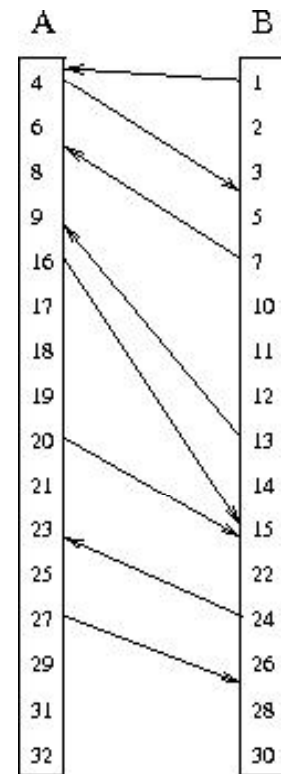
Termination condition

Either some $A(p_i+1)$ or some $B(j_p+1)$ loses

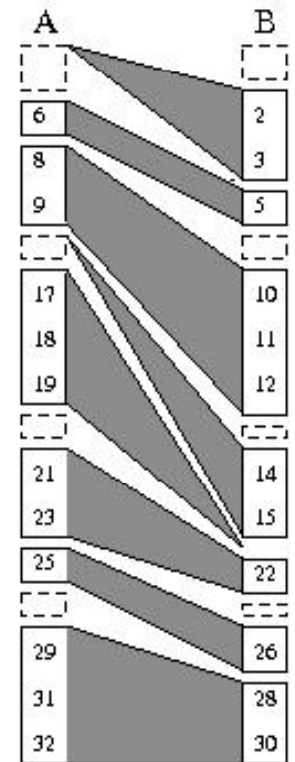
Parallel program $2p$ concurrent threads using a **single spawn-join for the whole algorithm**

Example Thread of 20: Binary search B.

Rank as 11 (index of 15 in B) + 9 (index of 20 in A). Then: compare 21 to 22 and rank 21; compare 23 to 22 to rank 22; compare 23 to 24 to rank 23; compare 24 to 25, but terminate since the Thread of 24 will rank 24.



Step 1
partitioning



Step 2
actual work

Linear work parallel merging (cont'd)

Observation $2p$ slices. None larger than $2n/p$.

(not too bad since average is $2n/2p=n/p$)

Complexity Partitioning takes $W=O(p \log n)$, and $T=O(\log n)$ time, or $O(n)$ work and $O(\log n)$ time, for $p \leq n/\log n$.

Actual work employs $2p$ serial algorithms, each takes $O(n/p)$ time.

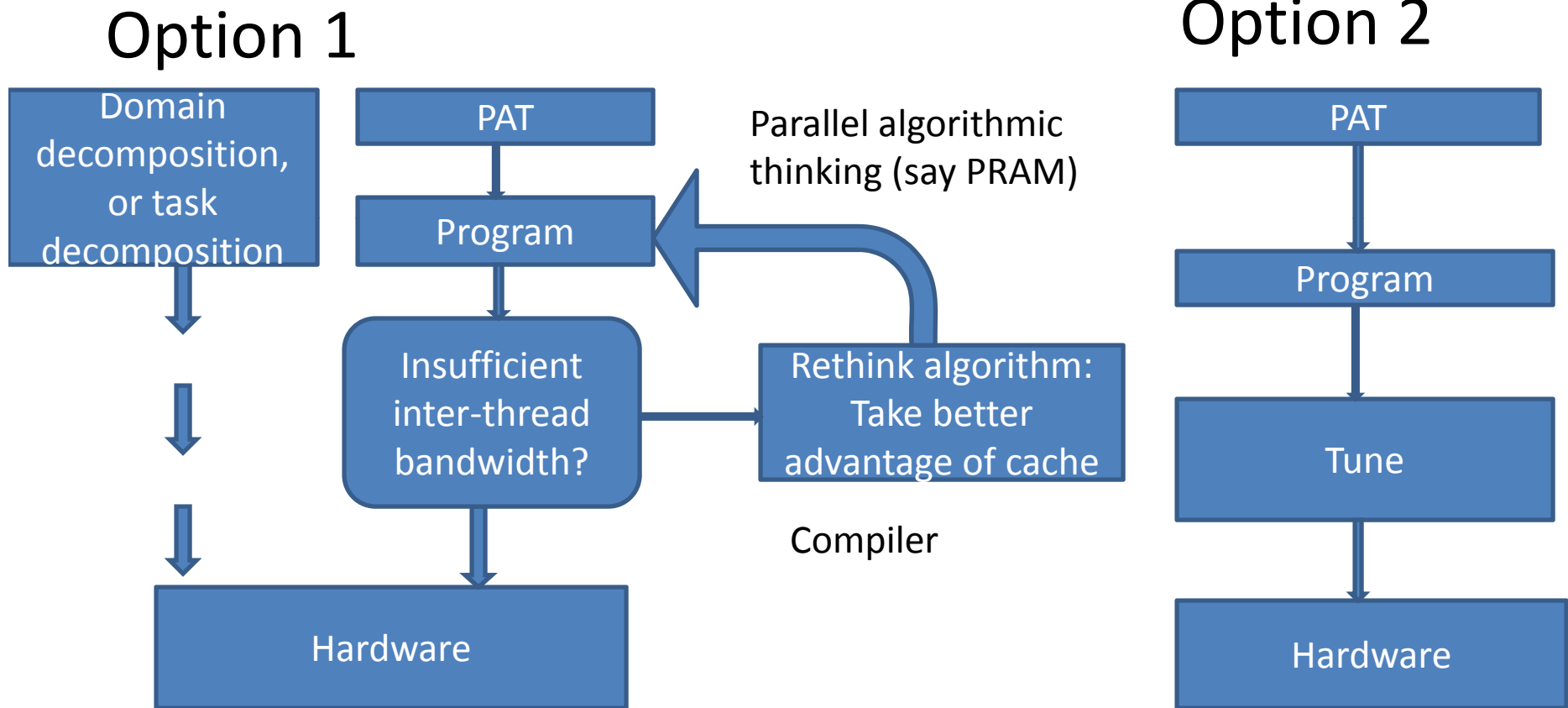
Total $W=O(n)$, and $T=O(n/p)$, for $p \leq n/\log n$.

IMPORTANT: Correctness & complexity of parallel program

Same as for algorithm.

This is a big deal. Other parallel programming approaches do not have a simple concurrency model, and need to reason w.r.t. the program.

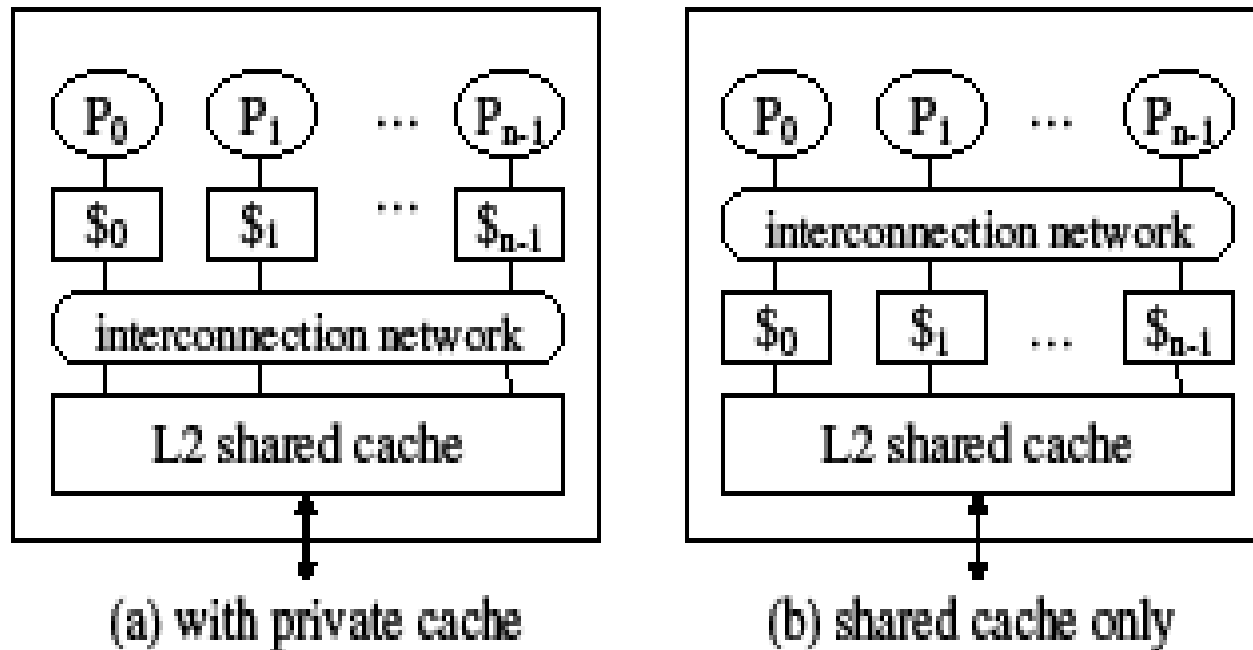
Workflow from parallel algorithms to programming versus trial-and-error



Is **Option 1 good enough** for the parallel programmer's model?
Options 1B and 2 start with a PRAM algorithm, but not option 1A.
Options 1A and 2 represent workflow, but not option 1B.

Not possible in the 1990s.
Possible now.
Why settle for less?

Memory architecture alternatives



Good news Fits on chip. Allows high-bandwidth interconnection network.

Left side: Makes sense for high-throughput. Uses cache coherence. Significant toll on bandwidth, power, and chip resources (each processor needs more silicon)

Right side: Amortizes memory resources → more processors. Unless in local registers (also “read-only caches” & “prefetch buffers”) need round-trip across ICN. Sequence of round-trips to memory a bigger issue.

Remaining constraint

Off-chip bandwidth remains limited. Still **less than for high-throughput, since one program.**

Synchrony versus ??

Synchrony Pros

- PRAM algorithmic theory
- Reasoning about correctness. Correlates with determinism
- Predictable timing of computation

Synchrony Cons

- Memory: 1 to 400 clocks → separate average-case from worst-case
- Tighter synchrony → more power.

Asynchrony?

- Asynchrony too difficult for algorithms and architecture. What can we do?

→ Reduced synchrony

- Balance (synchronous) PRAM algorithms and a no-busy-wait finite-state-machine (NBW FSM) “ideal”.
- Role for HW enhanced prefix-sum (F&A)
- Start with Arbitrary CRCW PRAM algorithms. Set concurrent threads that are as long-as-possible. Each advances at its own speed without letting another thread cause it to busy-wait. When all fails, join (every threads expires)
- Disciplined non-determinism.

How I updated my basic intuition

I first expected parallel systems to operate through punctual orchestration. “Swiss train system paradigm”: set clock when train leaves the station. Example: Cray 1970s vector machines. Cray was brilliant, but technology stood in the way. Expensive cooling was a warning sign. I went through 180-degree conceptual transformation between my SPAA97 and SPAA98 papers. I now expect progress to occur as distributedly as possible.

Accommodating a fixed number of processors (thread controls units, TCUs)

Issues

- Optimizing two parameters (work and time) → Can have more than one “best” parallel algorithm: $W1 < W2 < W3$ but $T1 > T2 > T3$. “building blocks”.
- Constants matter
- Cases: (i) Saturated TCUs, and (ii) unsaturated TCUs
- Ideally: programmer should not rewrite for more TCUs
- Responsibilities of programmer, compiler, hardware:

Programmer

- Algorithm and its XMTC code
- All building blocks

Compiler

- Assemble building blocks into ready to run code, given #TCUs, input size, and HW parameters
- Clustering threads

Hardware

- Initially, allocate virtual threads to TCUs, and then dynamically as TCUs become available
- No-busy-wait finite-state-machine (NBW FSM) ideal.

Realistic Role for theory

Wing: Computational thinking is often at multiple levels of abstraction (LoA)

One of these levels got to be the theory of parallel algorithms (reflecting parallel algorithmic thinking.)

The “how does it work” slide demonstrates our PRAM-like approach. If you restrict attention to one layer of this multi-layer onion, it becomes similar to other approaches, but .. misses the point. (The XMT language layer is one of the least innovative in XMT, but gets much attention.)

Need the vertical integration to make sense and understand in what ways it is different (unique).

Roles for theory (will be elaborated in panel presentation):

The algorithms (WD) LoA is traditional PRAM theory.

Performance tuning is in line with Experimental Algorithms. Much work is needed in developing understanding of what gives best performance in implementation.

Model, assess and compare HW, and HW options.

Model reasoning about multiple levels of abstraction.

PRAM-On-Chip Silicon: 64-processor, 75MHz prototype

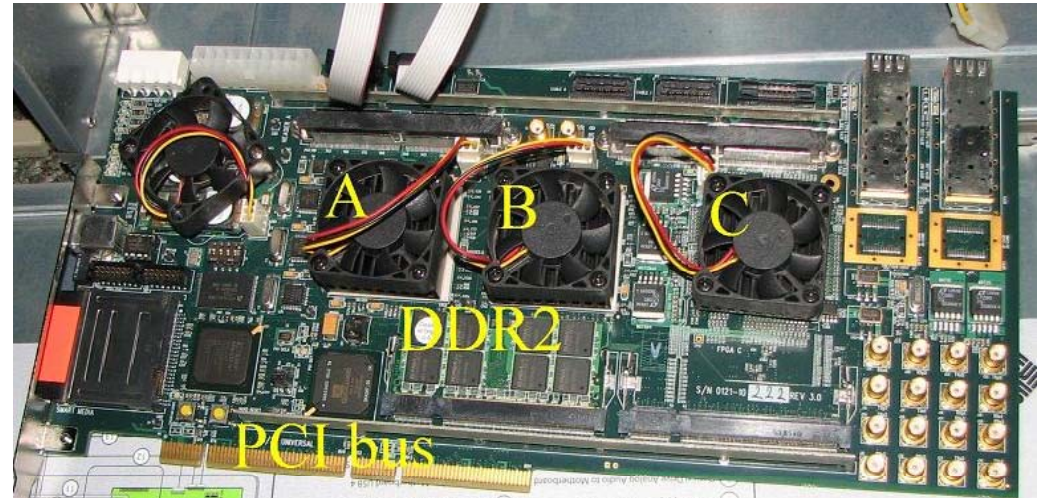
Specs and **aspirations**

n=m	64
# TCUs	1024

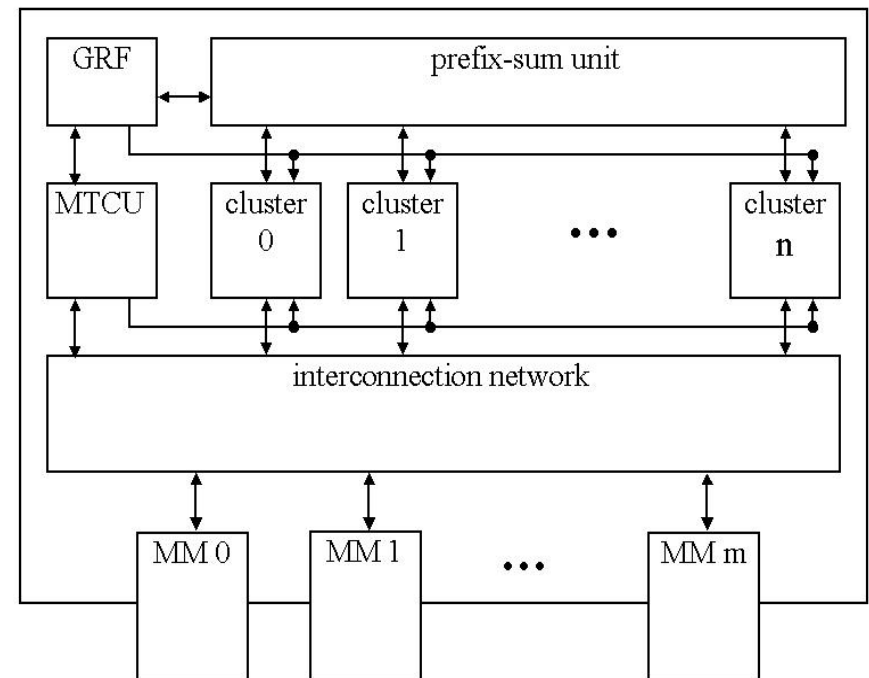
- Multi GHz clock rate
- Get it to **scale to cutting edge technology**
- Proposed **answer to the many-core era:**
“successor to the Pentium”?

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

- The system consists of 3 FPGA chips:
- 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



PRAM-On-Chip vs. Intel Core 2 Duo: Representative speedups w.r.t. cycle count (SPAA'09)

PROGRAM	Small	Large
Sparse Matrix Vector Mult	6.7	3.3
FFT	9.51	2.51
Quicksort	13.07	7.75

University project versus state-of-the-art industry processor. Problems and comparison methodology proposed by P. Dubey, Intel. Comparison favors the Intel design, since ASIC silicon area of a single Intel core is the same as the compared 64-processor XMT design. Comparisons in bold are more reflective of scalability potential: for the small inputs caches of both designs were sufficient. For large inputs, the Intel much larger caches made a difference. There is no reason that the cache size of a future XMT will be smaller.

	Small		Large	
PROGRAM	N	Footprint	N	Footprint
SparseMatVect	22K	200KB	4M	33MB
FFT	8K	192KB	8.76	96MB
Quicksort	100K	781KB	13.89	153MB

Experience with new FPGA computer

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

New computer used: to validate past speedup results.

Parallel algorithms graduate classes @UMD

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

2007 Sample speedups over best serial by students Selection: 13X.

Sample sort: 10X. BFS: 23X. Connected components: 9X.

2009 Programming assignments: Sparse Matrix-Vector Mult, Integer Sort, Sample Sort, Merge Sort, Summation with fine tuning (determine k for k-ary tree based on LSRTM), Find Spanning Forest in Graphs.

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.)

Freshmen Parallel Algorithmic Thinking Course, Spring'09

Strong enrollment. Majority non-majors. “How will programmers have to think by the time you graduate”. 80% of graduate course material was reviewed. No formal analysis (correctness/complexity).

Programming assignments: Binary tree prefix-sum, Randomized median finding, Integer Sort, Sample Sort, Merge Sort. Problems on par with first serial programming course.

Experience with K12 Students

Fall'07: 1-day parallel algorithms tutorial to **12 HS students**. Some (2 10th graders) managed 8 programming assignments, **including 5 of the 6 in the grad course**. Only help: 1 office hour/week by undergrad TA. No school credit. Part of a computer club after 8 periods/day.

One of these 10th graders: “I tried to work on parallel machines at school, but it was no fun: I had to program around their engineering. With XMT, I could focus on solving the problem.”

Dec'08-Jan'09: 50 HS students, **by self-taught HS teacher**, TJ HS, Alexandria, VA

Spring'09: Inner City Magnet, Baltimore Poly Institute.

Summer'09: Middle-School Summer Camp.

NEW: Software release & Course Material

Use **your own computer** for programming on an XMT environment and experimenting with it. Includes:

[\(i\) Cycle-accurate simulator of the XMT machine](#)

[\(ii\) Compiler from XMTC to that machine](#)

Extensive on-line course material. **Can teach and self-study parallelism anywhere, anytime:**

[\(iii\) Tutorial + manual for XMTC \(150 pages\)](#)

[\(iv\) Class notes on parallel algorithms \(100 pages\)](#)

[\(vi\) Video recording of full semester course \(30+ hours\), Spring 2009](#)

[\(vii\) Video recording of 9/15/07 HS tutorial \(300 minutes\)](#)

Next Major Objective

Industry-grade chip and production-quality compiler. Requires 10X in funding. Hope to get funding that will allow everybody here to get free or low-cost XMT machines.

Why Federal Funding

Vendors

Vendors want to see real applications developed and tested **before building in ASIC a machine powerful enough to.. develop and test such applications”**.

Go to Washington

Submit a proposal to say NSF: experienced engineers will (be hired and) build an industry-grade ASIC chip; the community can use it for application-development, research and education.

Paradigm shift sought: Reinvent computing for parallelism.

Peer-review funding mode: Paradigm shift by consensus.

T. Kuhn, Structure of Scientific Revolutions: Oxymoron! .. sigh

How to overcome a least-objectionable-first review system when a coherent school-of-thought (legitimate contender) is as good as it gets?

The NSF will need to resolve that for (the badly needed) greater HW diversity.

Conclusion

- Under work: function calls and nesting
- XMT goes after any amount/grain/regularity of parallelism you can find.
- 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”. XMT: “No more excuses. Just do it!”
- XMT is both complementary and a legitimate alternative for manycores:
 1. XMT+Theory: no competition on introducing “thinking in parallel” (to lower div. undergrads, and earlier). Same for methodology & ease of programming.
 2. Legitimate contender for defining general-purpose many-core computing (XMT with 2010s GPU replace past Pentium+GPU).
 3. NSF needs to weigh in for HW diversity and true natural selection: have industry-grade HW available for research, education & comparison

Participants

Grad students:, George Caragea, James Edwards, David Ellison, Fuat Keceli, Beliz Saybasili, Alex Tzannes, Joe Zhou. Recent grads: Aydin Balkan, Mike Horak, Xingzhi Wen

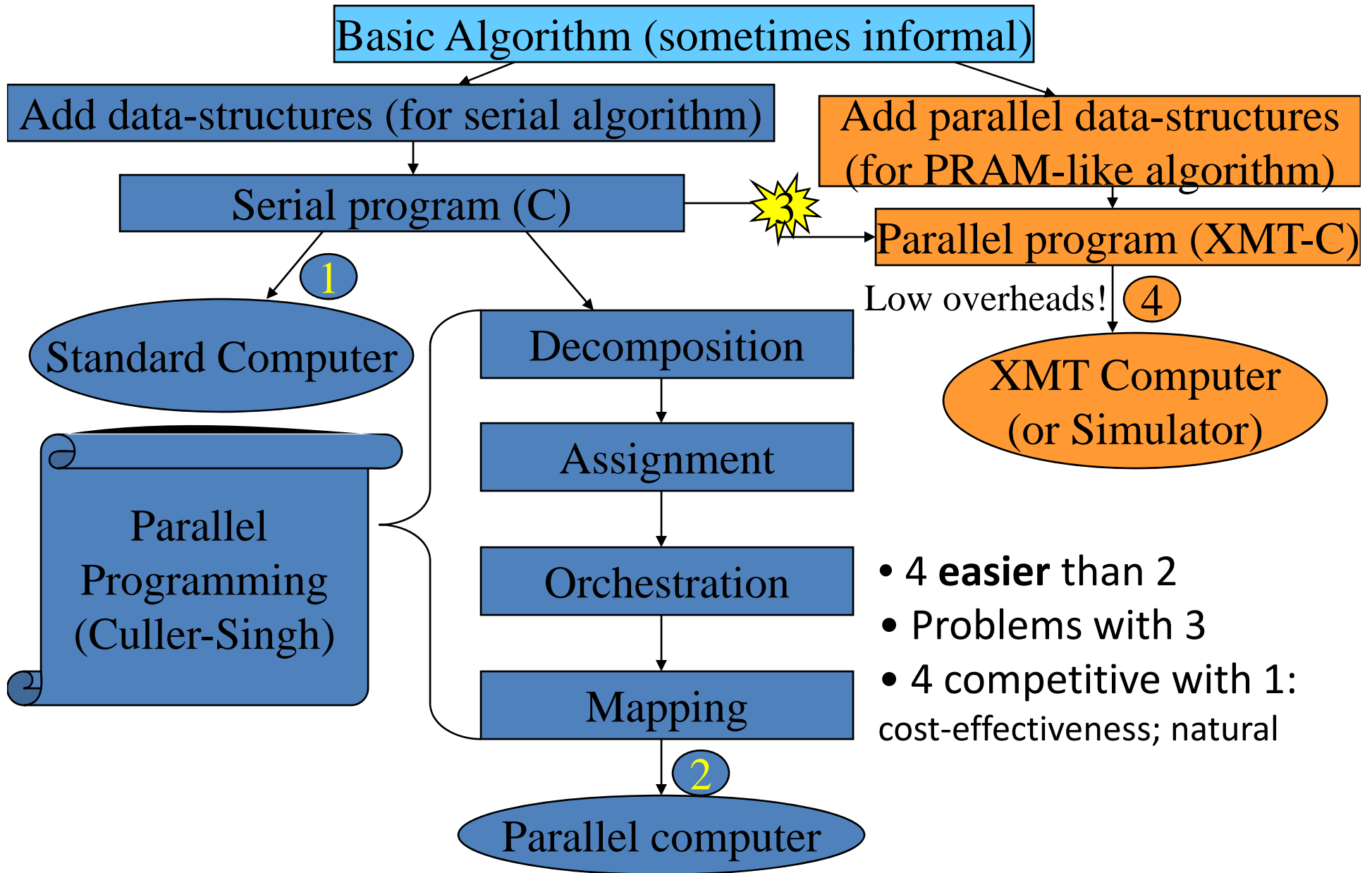
- Industry design experts (pro-bono).
- Rajeev Barua, Compiler. Co-advisor of 2 CS grad students. 2008 NSF grant.
- Gang Qu, VLSI and Power. Co-advisor.
- Steve Nowick, Columbia U., Asynch computing. Co-advisor. 2008 NSF team grant.
- Ron Tzur, Purdue U., K12 Education. Co-advisor. 2008 NSF seed funding

K12: Montgomery Blair Magnet HS, MD, Thomas Jefferson HS, VA, Baltimore (inner city) Ingenuity Project Middle School 2009 Summer Camp, Montgomery County Public Schools

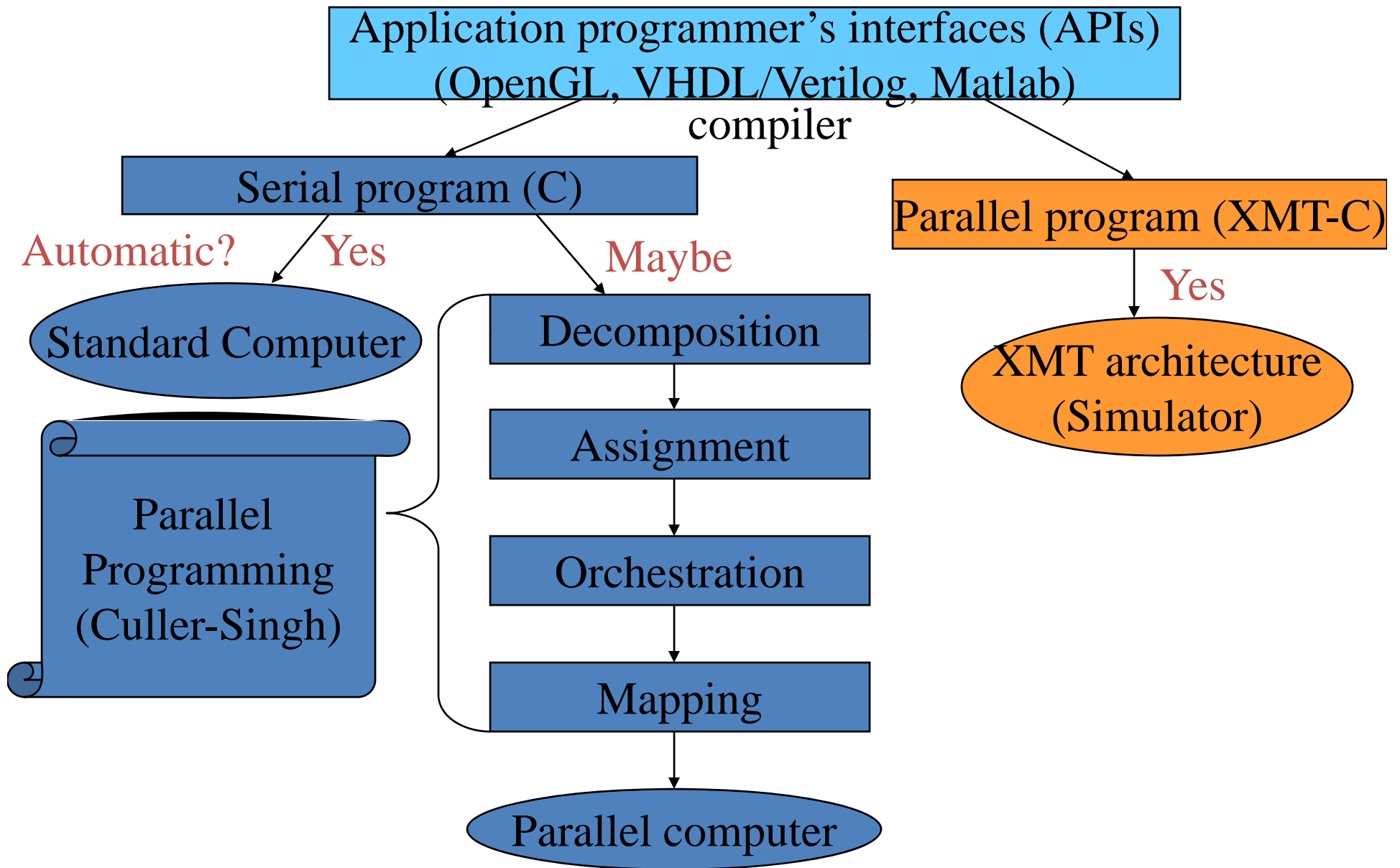
- Marc Olano, UMBC, Computer graphics. Co-advisor.
- Tali Moreshet, Swarthmore College, Power. Co-advisor.
- Bernie Brooks, NIH. Co-Advisor.
- Marty Peckerar, Microelectronics
- Igor Smolyaninov, Electro-optics
- Funding: NSF, NSA 2008 deployed XMT computer, NIH
- Industry partner: Intel
- **Reinvention of Computing for Parallelism. Selected for Maryland Research Center of Excellence (MRCE) by USM. Not yet funded. 17 members, including UMBC, UMBI, UMSOM. Mostly applications.**

Backup slides

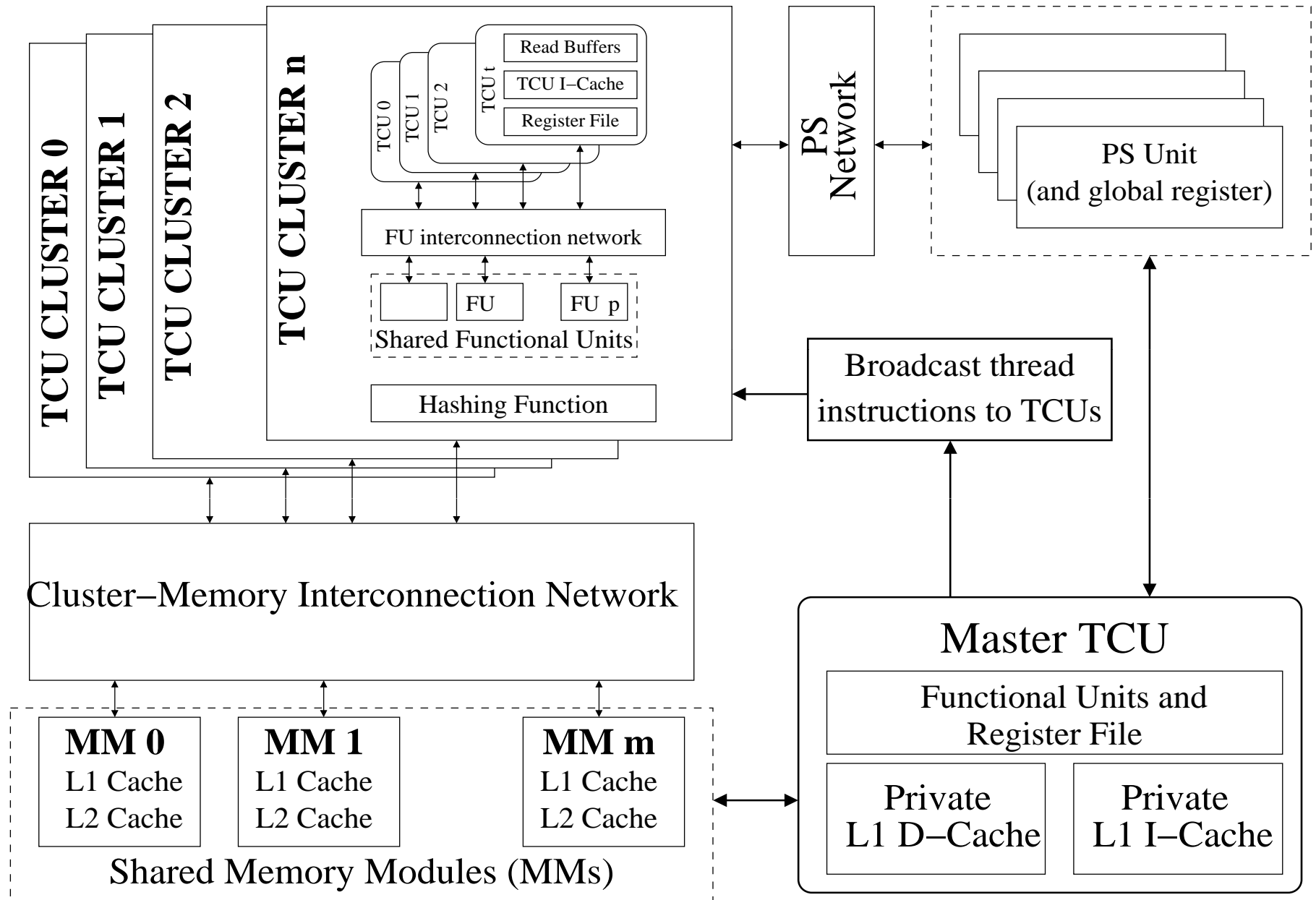
PERFORMANCE PROGRAMMING & ITS PRODUCTIVITY



APPLICATION PROGRAMMING & ITS PRODUCTIVITY



XMT Block Diagram – Back-up slide



How-To Nugget

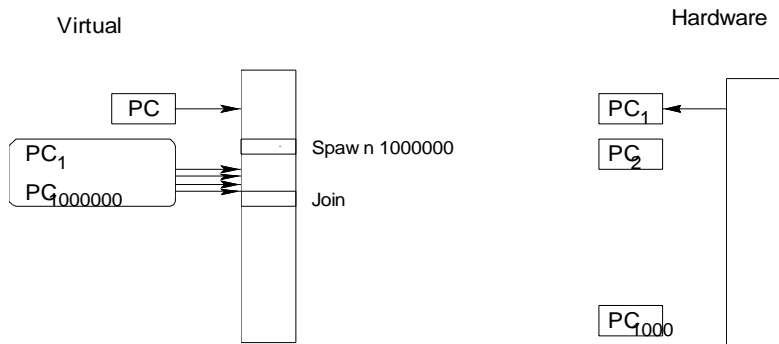
Seek 1st (?) upgrade of program-counter & stored program since 1946

Virtual over physical:
distributed solution

Von Neumann (1946--??)



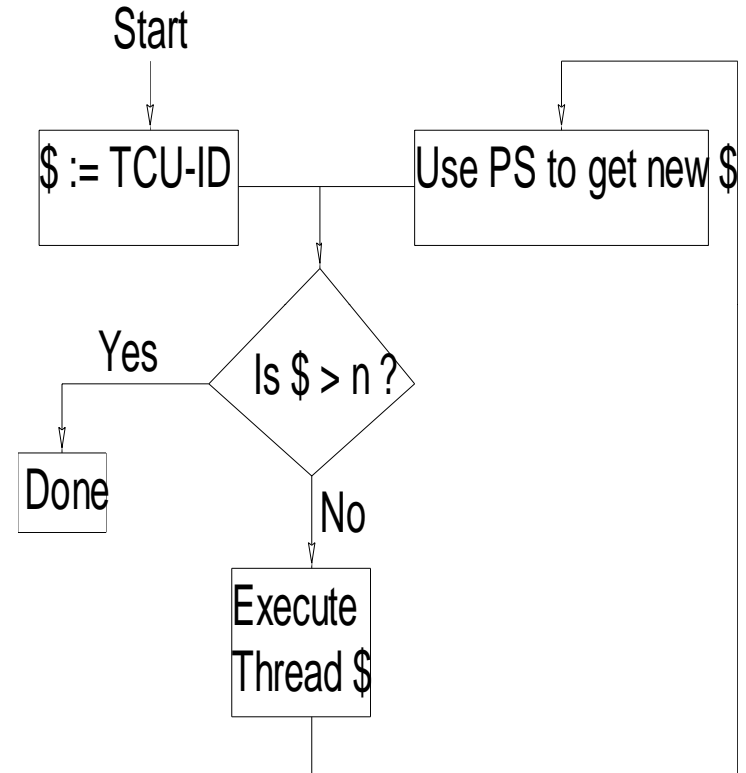
XMT



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



to PC₁, PC₂, PC₁₀₀₀ on a designated bus



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.

Some supporting evidence (12/2007)

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.

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

- Simulation of 1024 processors: 100X on standard benchmark suite for VHDL gate-level simulation. for 1024 processors [Gu-V06]
- Silicon area of 64-processor XMT, same as 1 commodity processor (core)

Naming Contest for New Computer

→ Paraleap

chosen out of ~6000 submissions

Single (hard working) person (X. Wen) completed synthesizable Verilog description AND the new FPGA-based XMT computer in slightly more than two years. No prior design experience. Attests to: **basic simplicity of the XMT architecture → faster time to market, lower implementation cost.**

XMT Development – HW Track

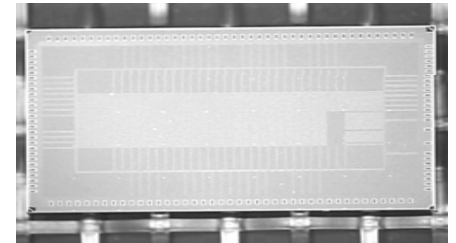
– Interconnection network. Led so far to:

❑ 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)



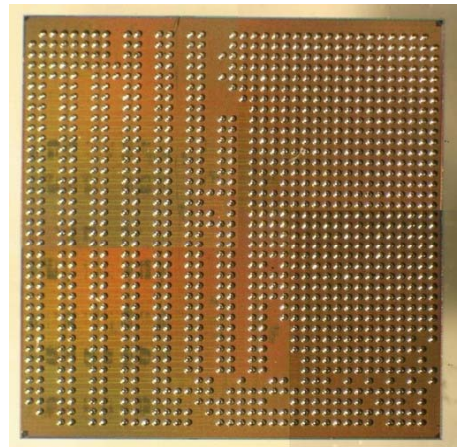
– Synthesizable Verilog of the whole architecture. Led so far to:

❑ 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

❑ 1st ASIC prototype—90nm 10mm x 10mm

64-processor tapeout 2008: 4 grad students



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+2 ASICs

Final thought: Created our own coherent planet

- When was the last time that a university project 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?

For more info:

<http://www.umiacs.umd.edu/users/vishkin/XMT/>