For General-Purpose Parallel Computing: It is PRAM or never

Uzi Vishkin

University of Maryland Institute for Advanced Computer Studies

A. JAMES CLARK SCHOOL of ENGINEERING

Short answer to John's questions

1. Make sure that your machine can look to the programmer like a PRAM. Without PRAM, evidence of dead-end.. assuming human programmers.
2. Possible to work out the rest: "PRAM-On-Chip" built at UMD; this presentation 10 yrs in <7 min

Envisioned general-purpose chip parallel computer succeeding serial by 2010 in 1997: Speed-of-light collides with 20+GHz serial processor. [Then came power ..]

View from our solution Several patents, but lots known.

A bit alarmed. Appear as architecture cluelessness. Especially for single task completion time. Must be addressed ASAP:

Architecture instability bad for business: why invest in long-term SW development if architecture is about to change.

Please do not stop with this workshop: Have coherent solutions presented ASAP. Examine. Pick winners. Invest in them.

Commodity computer systems

Chapter 1 1946—2003: Serial. Clock frequency: $\sim a^{y-1945}$

Chapter 2 2004--: Parallel. #"cores": $\sim d^{y-2003}$ Clock freq: flat.

Prime time is ready for parallel computing. But, is parallel computing ready for prime time: is there a general-purpose parallel computer framework 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.

Answer: YES. **PRAM-On-Chip@UMD is addressing (i)-(iv).** Performance programming is PRAM-like.

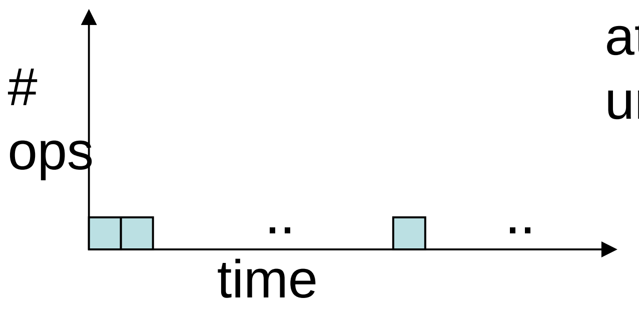
Rep speed-up [Gu-V, JEC 12/06]: 100x for VHDL benchmark.

Parallel Random-Access Machine/Model (PRAM)

Abstraction Concurrent accesses to memory, same time as one

ICS07 Tutorial: How to think algorithmically in parallel?

Serial doctrine

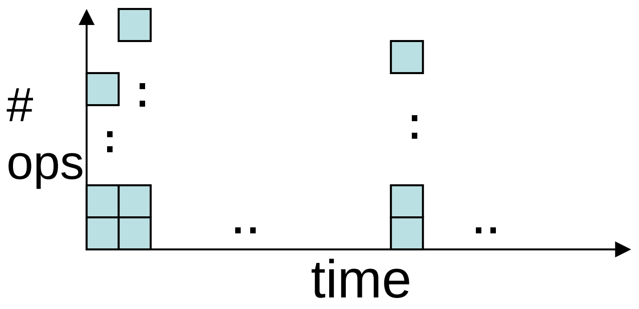


time = #ops

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

➔

Natural (parallel) algorithm



time << #ops

Where did the PRAM come from?

1960-70s: how to build and program parallel computers?

PRAM direction (my take)

1979- : figure out how to think algorithmically in parallel

1997- : use this in specs for architecture; design and build

The PRAM Rollercoaster ride

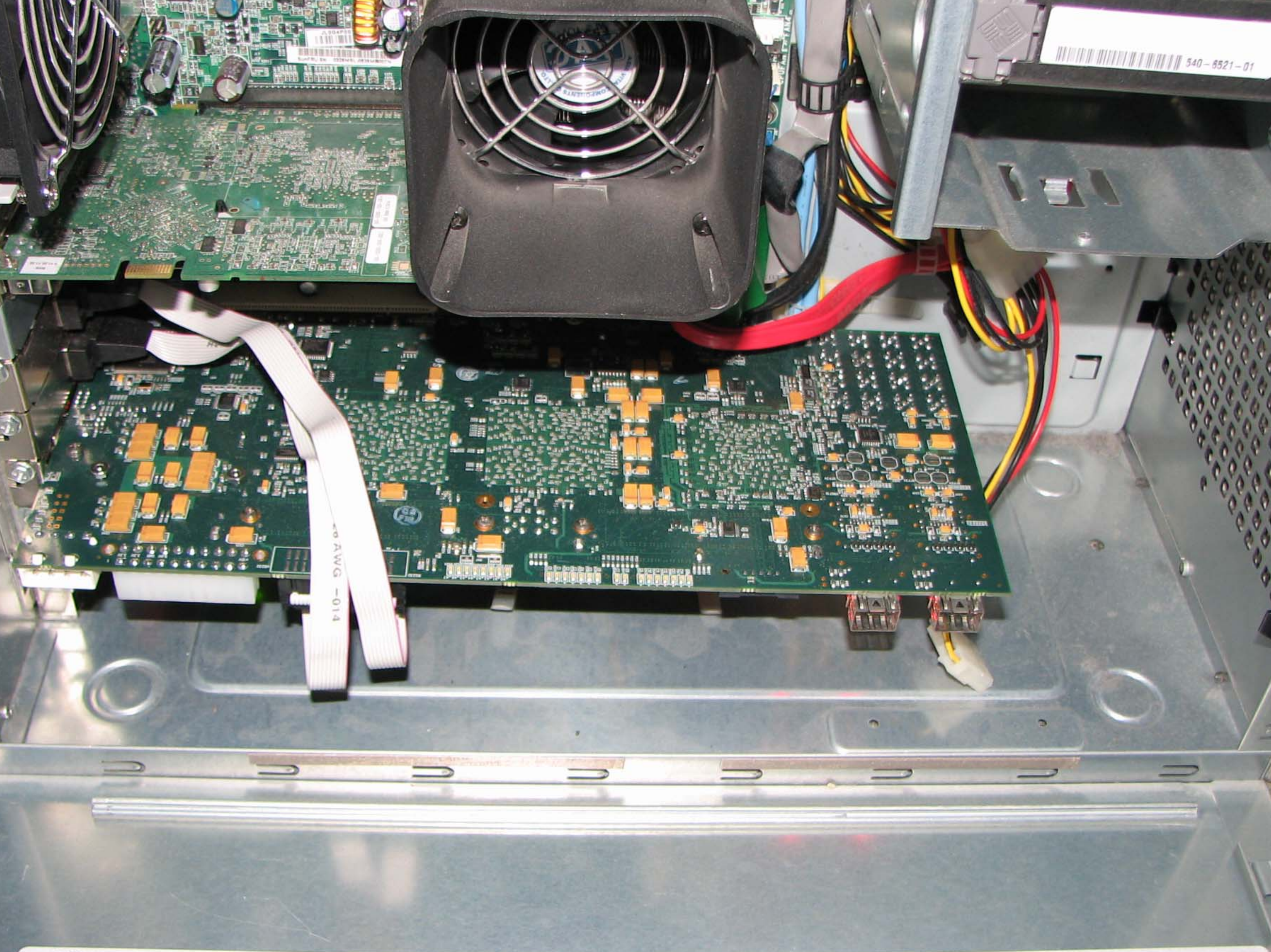
Late 1970's Dream

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". BUT, even the 1993+ despair did not produce proper alternative→ Not much choice beyond PRAM!

UP Dream coming true? eXplicit-multi-threaded (XMT) computer; realize PRAM-On-Chip vision: FPGA-prototype (not simulator), SPAA'07:


540-6521-01
28 AWG -014

What is different this time around?

crash course on parallel computing

– How much processors-to-memories bandwidth?

Enough	Limited
Ideal Programming Model: PRAM	Programming difficulties

In the past bandwidth was an issue.

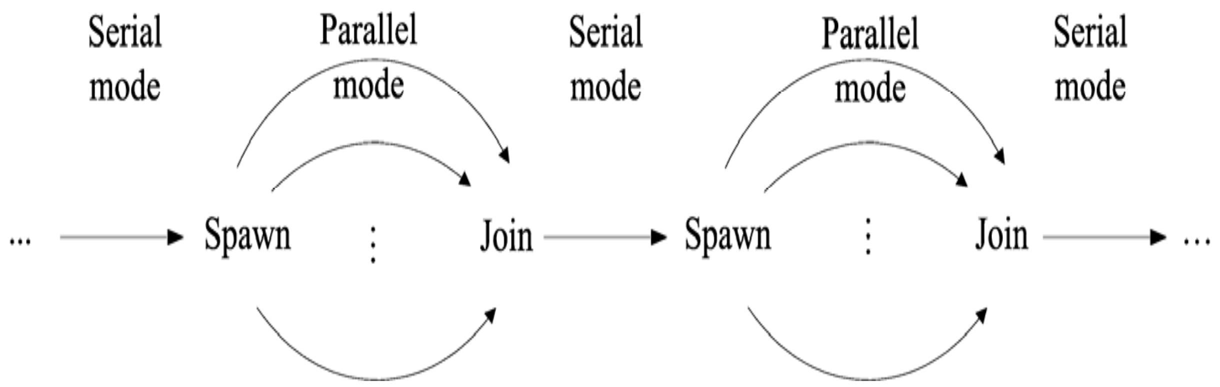
XMT: enough bandwidth for on-chip interconnection network. [Balkan,Horak,Qu,V-HotInterconnects'07: 9mmX5mm, 90nm ASIC tape-out—"Layout-accurate"]

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. Extra push needed is much smaller than you would guess.

Snapshot: XMT High-level language

XMTC: Single-program multiple-data (SPMD) extension of standard C.
Arbitrary CRCW PRAM-like programs.
Includes Spawn and PS - a multi-operand instruction. Short (not OS) threads.
To express architecture desirables present PRAM algorithms as:
[ideally: compiler in similar XMT assembly; e.g., locality, prefetch]



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).

PRAM-On-Chip

Specs and aspirations

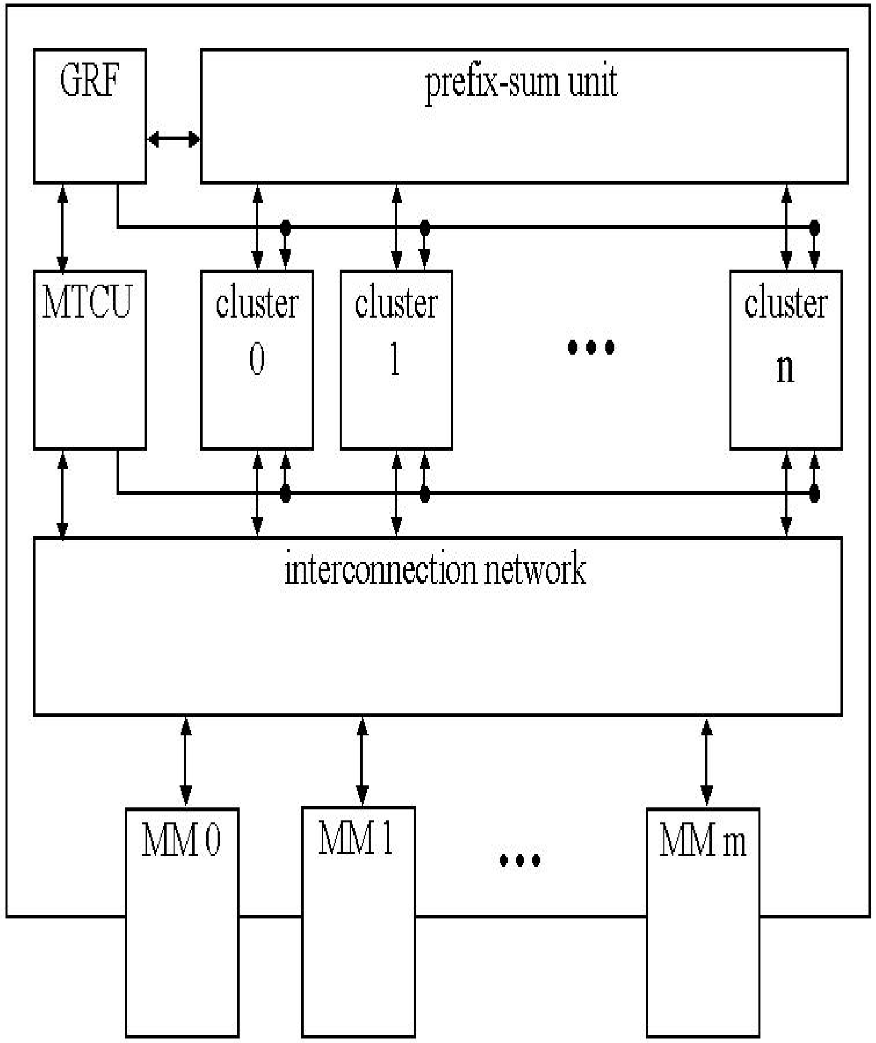
n=m	64
# TCUs	1024

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

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

- 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



Experience with new FPGA computer

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

New computer used: to validate past speedup results.

Zooming on Spring'07 parallel algorithms 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), "I am excited about one day having an XMT myself! "

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

Compare with

Build-first figure-out-how-to-program-later architectures.

Lack of proper programming model: programmability.

Painful to program decomposition step in other parallel programming approaches.

(Appearance of) Industry cluelessness.

J. Hennessy 2007: "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"

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"

More “keep it simple” examples

Algorithmic thinking and programming

- PRAM model itself; and the following plans:
- Work with motivated high-school students, Fall’07.
- 1st semester programming course. Recruitment tool: “CS&E is where the action is”.
- Undergrad parallel algorithms course.

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.

Conclusion

Any successful general-purpose approach must (also) answer: what will be taught in the algorithms class? Otherwise dead-end

I concluded in the 1980s: For general-purpose parallel computing it is PRAM or never. Had 2 basic options: preach or do

PRAM-On-Chip: Showing how PRAM can pull it is more productive & fun.

Significant milestones toward getting PRAM ready for prime time. IMH0: Now, just a matter of time (& money)

Naming Context for New Computer

http://www.ece.umd.edu/supercomputer/

Cash award.

FPGA Prototype of PRAM-On-Chip:
1st commitment to silicon

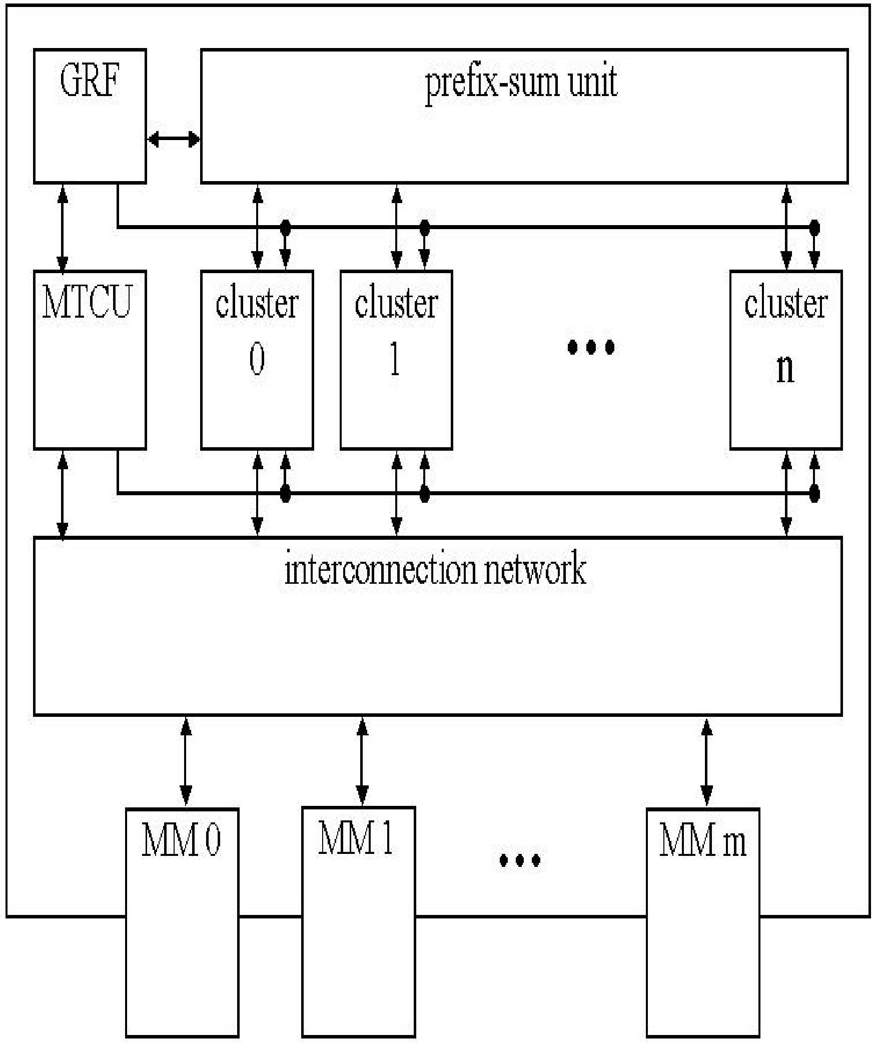
[FPGA prototyping: “can build”.]

Specs of FPGA system: n=4; m=8

Clock rate	75 MHz
Memory size	1GB DDR2
Mem. data rate	2.4GB/s
Number of TCUs	64 (4 X 16)
Shared cache size	256KB (32 X 8)
MTCU local cache	8KB

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

Block diagram of XMT



Back-up slides: 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

Nature of XMT Enhancements

Question Can innovative algorithmic techniques exploit the opportunities and address the challenges of multi-core/TCU?
Ken Kennedy's answer: And can we teach compilers some of these techniques?
Namely: (i) identify/develop performance models compatible with PRAM; (ii) tune-up algorithms for them (can be quite creative); (iii) incorporate in compiler/architecture.

Back-up slide:
Explanation of Qsort result

The execution time of Qsort is primarily determined by the actual (DRAM) memory bandwidth utilized. The total execution time is roughly = memory access time + Extra CPU time

6.4GB/s is the maximum bandwidth that memory system provides. However, the actual utilization rate depends on the system and application.

So, XMT seem to have achieved higher bandwidth utilization than AMD.