Disruptive Long-term (~2018) R&D Goal:

Easy-to-program many-processor supercomputer

Uzi Vishkin

University of Maryland

www.umiacs.umd.edu/users/vishkin/XMT

UMIACS

University of Maryland Institute for Advanced Computer Studies

UNIVERSITY OF MARYLAND 1856

A. JAMES CLARK SCHOOL *of* ENGINEERING

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].

- Tribal lore, DARPA HPCS Development Time study: "Parallel algorithms and programming for parallelism is easy. What is difficult is the programming/tuning for performance."

- 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. Recall the Productivity in HPCS.

Preview

Aiming at strong speed-ups for single-task completion time, conceive an easy-to-program architecture opportunity.

- **Derive ICN "draft specs" (#terminals, bandwidth, latency).**
- **Suggest "draft specs" for enabling technologies to make architecture possible.**
- **2013: have enabling technologies (free space optics, plasmonics, waveguides, etc), mixed timing "ligaments" (asynch grant & work with S. Nowick, Columbia), do their magic.**
- **Build architecture.**

"Moon shot of HPC"

But, where to start?

By extrapolation (from an UMA approach)

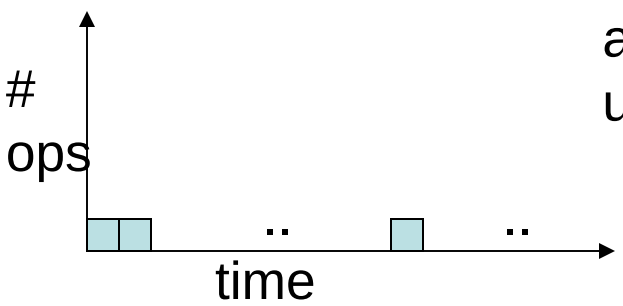
Easy-to-program 64-processor FPGA prototype (and 90nm tape-out)

- XMT: Explicit multi-threading. Goal: 1000 thread control units (processors) on-chip, multi GHz clock.
- Easy to build. Single grad student. No design experience. 2+ years from simulator to FPGA.
- Programmed so far by 35 HS students (23 taught by self taught teacher). Some did 5 grad course prog assignments.
- Plans with K12 Math Ed Profs, CS Ed Prof, and HS curriculum dev experts, inc. Middle School.
- Taught to non-CS Freshmen@UMD.
- Contrast with “Need PhD in CS to program current multi-cores” AMD (C. Moore)/Intel.

Parallel Random-Access Machine/Model (PRAM)

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

Serial doctrine

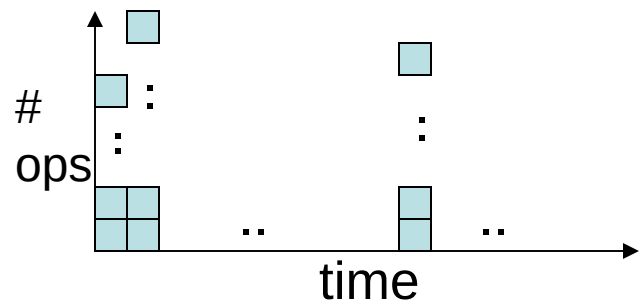


time = #ops

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

"In theory there is no difference between theory and practice but in practice there is" ➔

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

Preview (cont'd) use XMT as yardstick for 2018 roadmap

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?]; **Device changed it all:**

UP Highlights: eXplicit-multi-threaded (XMT) FPGA-prototype computer (not simulator), SPAA'07,CF'08; 90nm ASIC tape-outs: int. network, Hotl'07, XMT. **# on-chip transistors**

How come? crash "course" on parallel computing

How much processors-to-memories bandwidth?

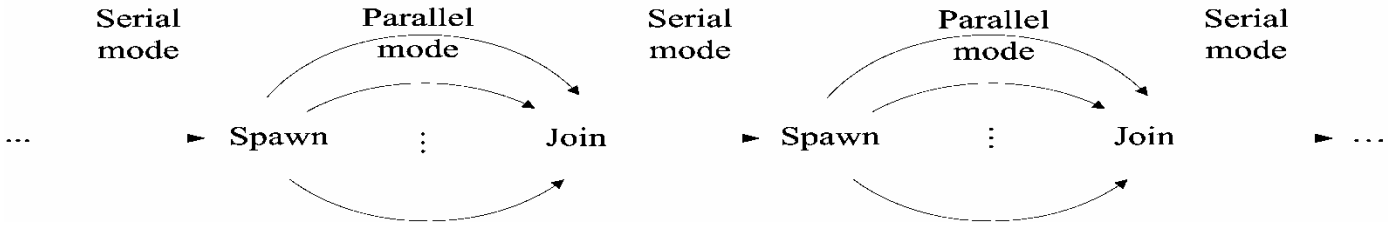
Enough: Ideal Programming Model (PRAM)

Limited: Programming difficulties

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.)

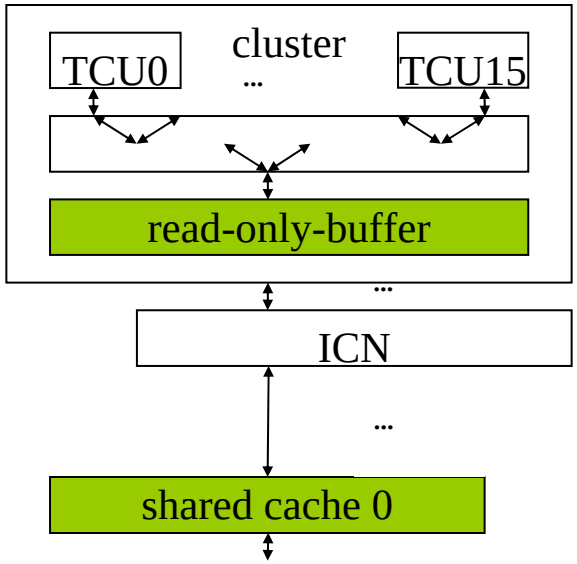
Discussion

- Program for parallelism. Must realize tension with locality. Ride a bike with feet on the ground. Programming for locality has limited HPC. Counter intuitive to some architects:

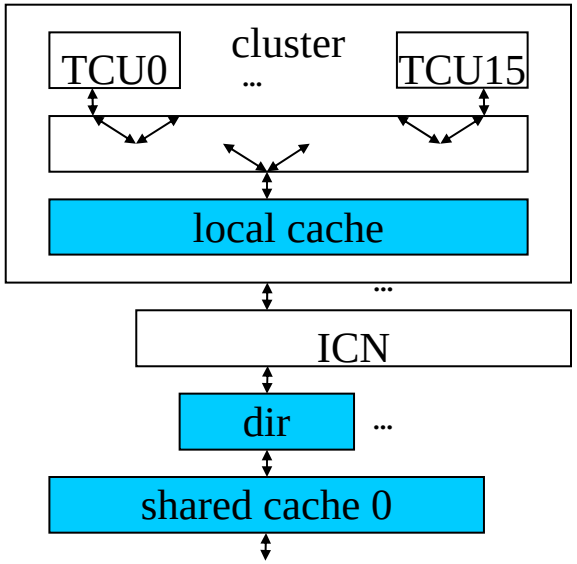
Story time (applicable to on-chip parallelism & on-chip caches)
Several current courses. Each has a text. The library has 1 copy. Should the copies be: (i) reserved at the library, (ii) reserved at a small library where the department is, or (iii) allow borrowing and then satisfy requests when needed

- The PRAM "level of cognition (LOG)": lowest common denominator (LCD) among parallel approaches => PRAM LOG necessary condition, regardless who eventually wins the ongoing battle on programming model(s) for future multi-cores.
- Current standard practice produces 22-year (CSE graduate) dinosaurs; going to 50-year career, dominated by parallelism, have only learned to program serial machines. LCD argument ➔ teach PRAM & PRAM-like programming.
- 10th graders succeed as 12th graders; Non-major Freshmen succeed as Senior majors. Less bad habits to undo.
- Makes possible: same model as for commodity systems
- Believe that MPI can be supported.

Comparison between XMT S and XMT P



(a) read-only-buffer, no cache coherence mechanism



(b) local cache, directory based cache coherence system

	Read-only buffer in XMT S	Private cache in XMT P
Coherence	No	Yes
Size	8KB	32KB
Latency	2 cycles	2 cycles
Associativity	Direct map	2-way
Granuality	1 word	8 word

Read latencies in XMT S and XMT P

8 kernel benchmarks are tested in 4 different size of XMT S and XMT P. (8,16,32,64-cluseter)

Read from	# of cluster	XMT S	XMT P
Prefetch buffer	All	1	1
RO buffer or local cache[1]	all	4	4
Shared cache (SC)[2] (miss in RO buffer or local cache)	8	21	24
	16	25	28
	32	29	32
	64	33	36
Off-chip memory	all	150+SC	

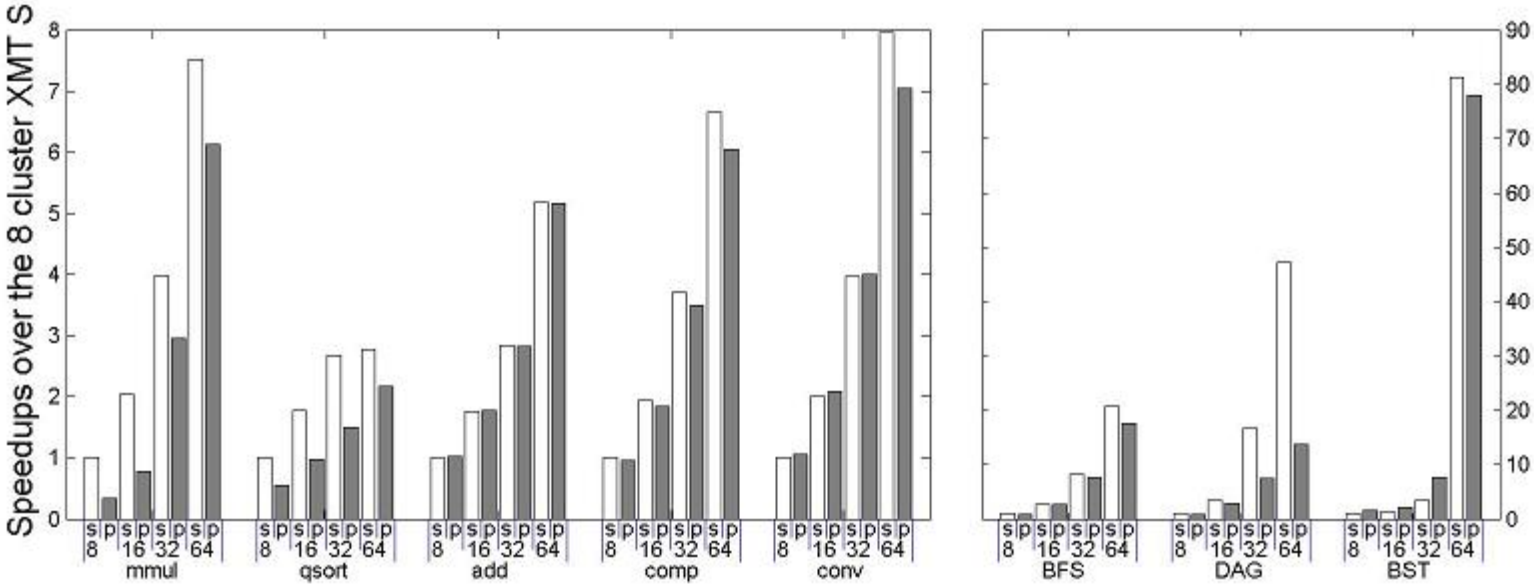
1 Since RO buffer o local cache is shared by 16 TCUs in a cluster, arbitration and result delivery takes 1 cycle each.

2.Assumed there is no contention in the interconnection network and shared cache module. XMT S needs extra 3 cycles for directory access and additional FIFOs.

Kernel benchmarks

App.	Brief description	Input size
mmul	Matrix multiplication	128x128
qsort	Quick sort	100K
BFS	Breadth first search	V=100K,E=1M
DAG	Finding longest path in a directed acyclic graph	V=50K,E=600K
add	Array summation	3M
comp	Array compaction	2M
BST	Binary search tree	2.1M nodes 16K keys
conv	convolution	Image:200x200 Filter: 16x16

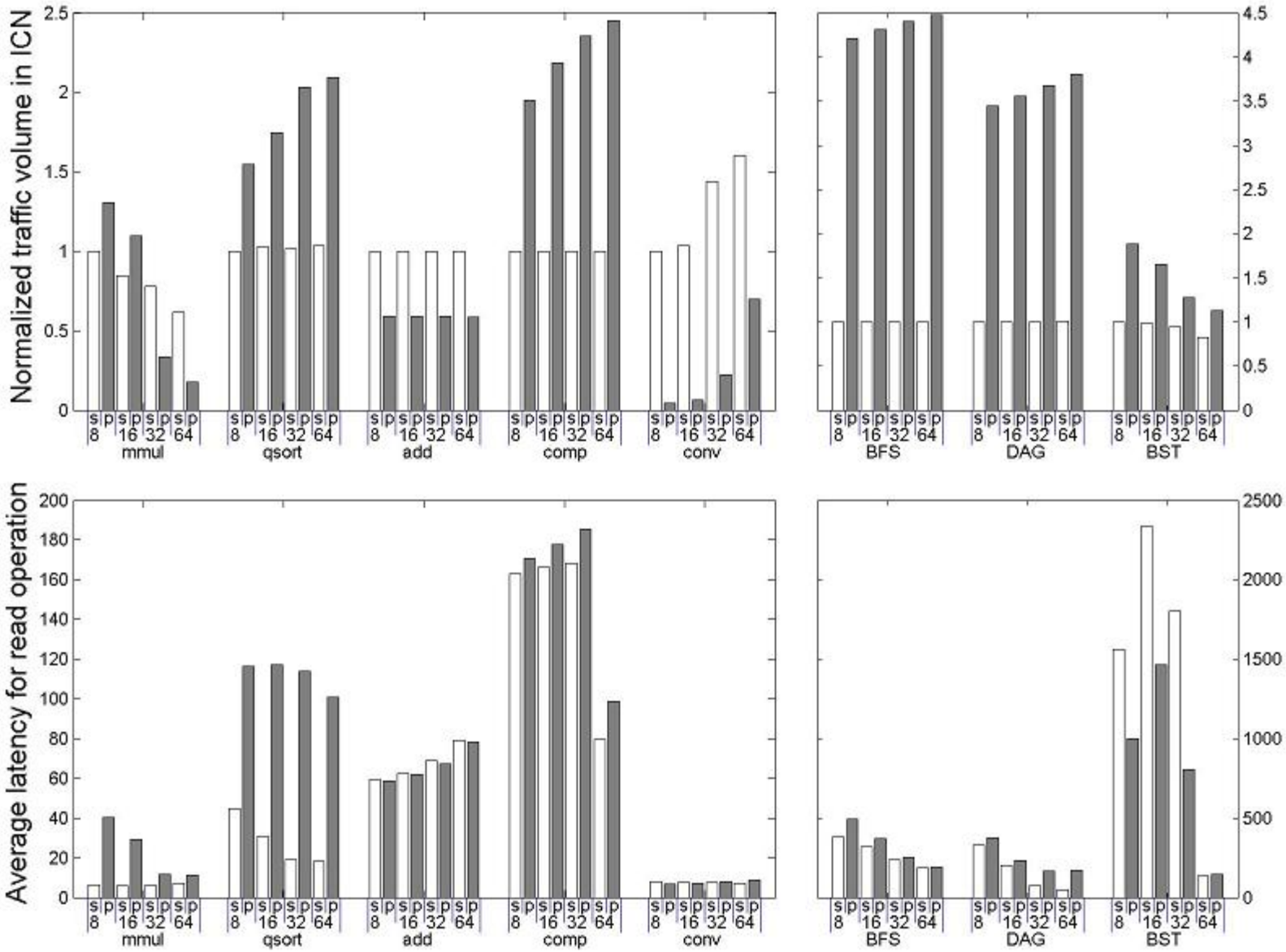
Speedups of the benchmarks over the eight cluster XMT S



Average speedups of XMT S over XMT P (same number of clusters)

	mmul	qsort	add	comp	conv	BFS	DAG	BST
average	2.039	1.682	0.995	1.063	1.103	1.009	1.983	0.692

Traffic in IN and Average Latency



Possible ICN draft specs for 250K processors in 8K clusters (32/cluster)

- Sufficient bandwidth?
- Can you achieve 60 clocks latency?

☺Mental poker: x250K processors, x8K cluster, $60+3\log_2 x$ clocks latency

Optoelectronic folks Can you provide that?

Discussion

Can send bit across/beyond the globe; why not nearby?

Seek to share NRE. E.g., have NIH fund beyond HW as platform for drug design. Bipartisan support.

Other approaches only sound-risk averse by relating to current multi-core; however, future multi-core will have to change for: Programmability & scalability.

Please stay tuned

- Will release shortly XMTC compiler + XMT architecture simulator

<u>Some more references</u>

SPIE'04&07 papers: Waveguides & Plasmonics

(w. I. Smolyaninov & C. Davis)

July 16, 2008 happy day at UMD:

Synchronous mesh of tree defense: A.O. Balkan (ASAP06, HotI07,DAC08)

Asynch (incl. mixed timing) defense: M.N. Horak + new DA NSF grant. Both with S. Nowick

www.umiacs.umd.edu/users/vishkin/XMT

Some comments on power

- Cache coherence: great way to melt your chip
- GALS design (in part due to IOS)
- Asynch: very promising on low power
- Hashing at end points reduces load on ICN

Some pros and cons of private caches

Pros

- Better performance for coarse-grained programs
- Short latency for local cache hit
- Hardware-supported cache coherence protocol

Cons

- Hardware-supported cache coherence protocol (power)
- Complicated hardware, limited scalability
- Inefficient for certain types of memory access pattern, like false sharing

Some pros and cons of shared caches

Pros

- Simple hardware, good scalability.
- Efficient use of silicon. (No multiple copies)
- Fine-grained parallelism.
- Easy-to-program. (Locality is less important)

Cons

- Long cache access latency