From Asymptotic PRAM Speedups To Easy-To-Obtain Concrete XMT Ones

Uzi Vishkin

For background and more information:
Using Simple Abstraction to Reinvent Computing for Parallelism, CACM, January 2011, pp. 75-85
& http://www.umiacs.umd.edu/users/vishkin/XMT/
Didn't want to repeat my 5/2009 T&MC workshop

Commodity computer systems

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

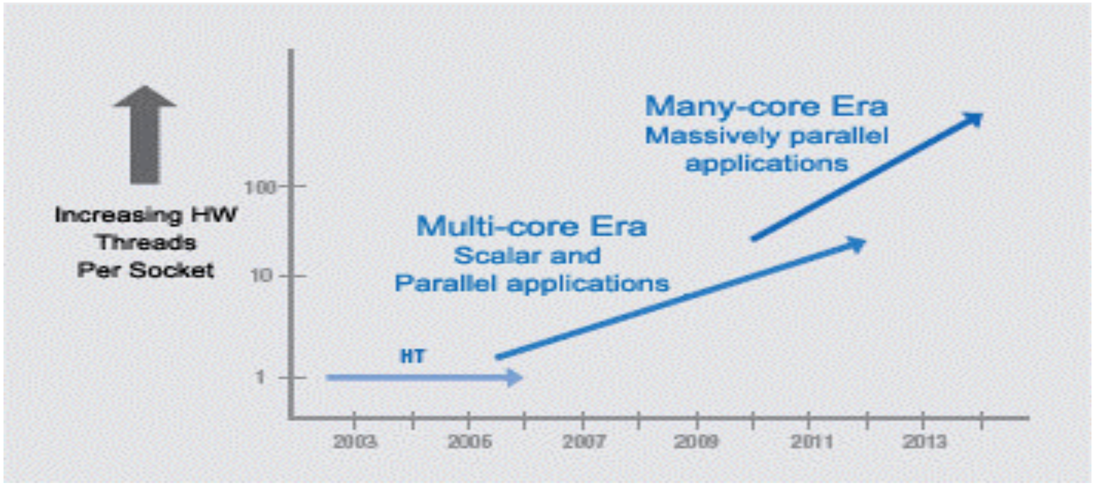
2004 General-purpose computing goes parallel.

Clock frequency growth flat. #Transistors/chip 1980→2011: 29K→30B!

#"cores": increase exp.

Intel Platform 2015, March05:

If you want your program to run significantly faster … you' re going to have to parallelize it



But, what about the programmer? "The Trouble with Multicore: Chipmakers are busy designing microprocessors that *most programmers can't handle*"—D. Patterson, IEEE Spectrum 7/2010

Only heroic programmers can exploit the vast parallelism in current machines – The Future of Computing Performance: Game Over or Next Level?, Report by CSTB, NAE 12/2010. **Conclusion** need new stack: algorithms to HW. In spite of no algorithms people on committee.

My 2020 Vision

- Unprecedented opportunity for **Chinese (or other) to dominate processors & systems** in mainstream computers and major devices

Is the game really over? Or, why is the success of parallel machines so limited?

- All parallel machines to date **failed** to generate a broad base of **application programmers** because they have been **too difficult to program**
- They **sidestepped ease-of-programming (EoP**) , or treat it as an afterthought. [Can adding architectural support be good enough?]
- Current parallel computing government/industry efforts, even as it is becoming mainstream, are at **risk** of investing in yet **another generation of too-hard-to-program** parallel machines
- [Counting out multi-core technology “pushers”] how big is current **demand for undergrad parallel programming education** from application SW developers? My impression: very **small**. Reason: future products unclear (ROI) (& too difficult to program).

Diagnosis **Mismatch** of algorithms/EoP and architecture

- Reward system matters. Consider reward of addressing one symptom at a time.. symptomatic medicines for life **versus** diagnosis ➔ cure

Who benefits from which

- Symptomatic medicine for life:
  - Farma companies
  - DGP
  - Wall Street
  - Washington
  - Even your Drs&spouses may get a free cruise to Hawaii
- Diagnosis and cure:
  - Your unpatriotic self

What do you think will happen?

Why most programmers can’t handle today’s machines? (cont’d)

Hypothesis: Flawed architecture foundation

- originated with ‘design-first figure-out-how-to-program-later’
- Where are the *rewards*?
  1. Funding for new general-purpose architectures: basically gone
  2. Originality-seeking publications culture ➔ mismatch provides rich opportunities; flawed system legitimate if vendor-backed

Will discuss one approach to do things **differently**

- **Idea** preempt the mismatch by a *back-to-the-future playback,* to just before the turn. Take a different one. Rebuilb and generate **evidence** that improvements by order-of-magnitude may be possible.
- *Reward alert*: Try to publish a paper boasting easy results ➔ Is the so badly needed EoP a lose-lose proposition?

A fool may throw a stone into a well which a hundred wise men cannot pull out

☺

- But they can write many papers about

- Will not be so bad if we accept the papers, but reject the system

But, how much can we gain in

- Performance and
- Ease-of-programming (EoP)

Rules if you believe in your own theory/architecture

1. Don't stop at asymptotics; IMO theory with no form of (eventual) validation remains 'fluffy'
2. Don't repeat EoP failure since 'not quantifiable'; do your best and show it

Will note some new$_1$ examples for performance

1 For more: Wed talk, by James Edwards

Approach

Stage 1 (community wide) **PRAM** theory: *p synchronous processors communicating in unit time through shared memory*

Sought to define parallel algorithmic thinking. Created unmatched wealth of parallel algorithms

Perception given above machines whose programming mandates decomposition-first: 'PRAM simplest parallel model' [e.g., TCPP 12/2010 curriculum meant simplistic, but were too polite]

Stage 2 **XMT** – explicit multithreading.

Make EoP a first-order architectural objective.

'Objecting' to TCPP: 'PRAM is *way too difficult*'; why care about processors and allocating them to jobs? Instead:

Any number of 'concurrent instructions' can execute next

Build system (HW, compiler, programming methodology). See what you can do on performance & EoP. Compare

Not Just Talking

Algorithms

PRAM parallel algorithmic theory. "Natural selection". **Latent**, though not widespread, **knowledgebase**

Work-depth.CACM11: "immediate concurrent execution (ICE)"

SV82 conjectured: Full PRAM algorithm just a matter of skill.

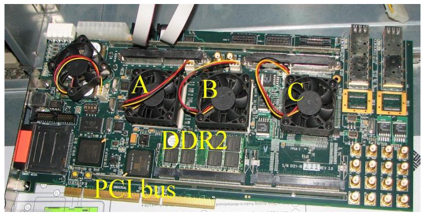
Used as framework in main PRAM algorithms texts: JaJa92, KKT01

Later: **programming** & **workflow**

Rudimentary yet stable **compiler**.

PRAM-On-Chip HW Prototypes

64-core, 75MHz FPGA of XMT

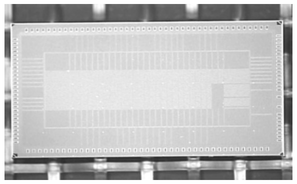


SPAA98..CF08

Toolchain

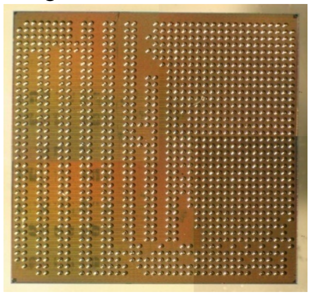
Compiler + simulator HIPS' 11

128-core intercon. network



IBM 90nm: 9mmX5mm, 400 MHz [HotI07]

•



FPGA design→ASIC

IBM 90nm: 10mmX10mm

150 MHz

Architecture scales to 1000+ cores on-chip

Are PRAM graph algorithms any good in practice?

- NSF/IEEE-TCPP curriculum: not really.

  No poly-logarithmic PRAM graph algorithms
- Very limited evidence to counter that:
- Graph algorithms in particular tend to be difficult to implement efficiently:
  - Biconnectivity [CB05] (12-processor Sun machine): <= 4X speedups after major change in [TarjanV85]. <1X otherwise
  - Max flow (hybrid GPU-CPU implementation): Speedups of up to 2.5x [HH10]

1st quantitative results 2nd context/significance 3rd EoP

Back to topic From Asymptotic PRAM Speedups To Easy-To-Obtain Concrete XMT Ones

XMT speedups. Comparing cycles with best serial algorithm and compiler on modern CPU:

- Connectivity [EV11] 39X-100X. Direct implementation of [SV82]
- Biconnectivity [EV11] 9X-33X. Direct implementation of [TV85]
- MaxFlow [CV-SPAA11] up to 108X. Balance push-relabel [GT88] with the original layered PRAM MaxFlow algorithm [SV82]. Then direct implementation.

Biconnectivity: most advanced problem in parallel algorithms texts

MaxFlow: ~ most advanced in serial algorithms texts

First XMT papers on a single problem.

No publication on biconnectivity yet on GPUs.

Quant results in GPU context

- Compared:
  1. XMT running a PRAM algorithm with few or no modifications
  2. A multi-core CPU or GPU. Either heavily modified version - same PRAM algorithm, or another algorithm
- Connectivity [EV11]:

- 2.2x-4x over an optimized GPU implementation that greatly modified the original algorithm.
- GPU: Non-trivial paper
- PRAM algorithm: one of 6 programming assignments in standard PRAM algorithm class; even done by couple of 10th graders, Blair HS, MD

- BFS [HotPAR10]:

- 5.4x speedups over optimized GPU implementation
- 73x for large radius (low average parallelism)

More evidence on **ease of obtaining speedups**

Breadth-first-search (BFS)

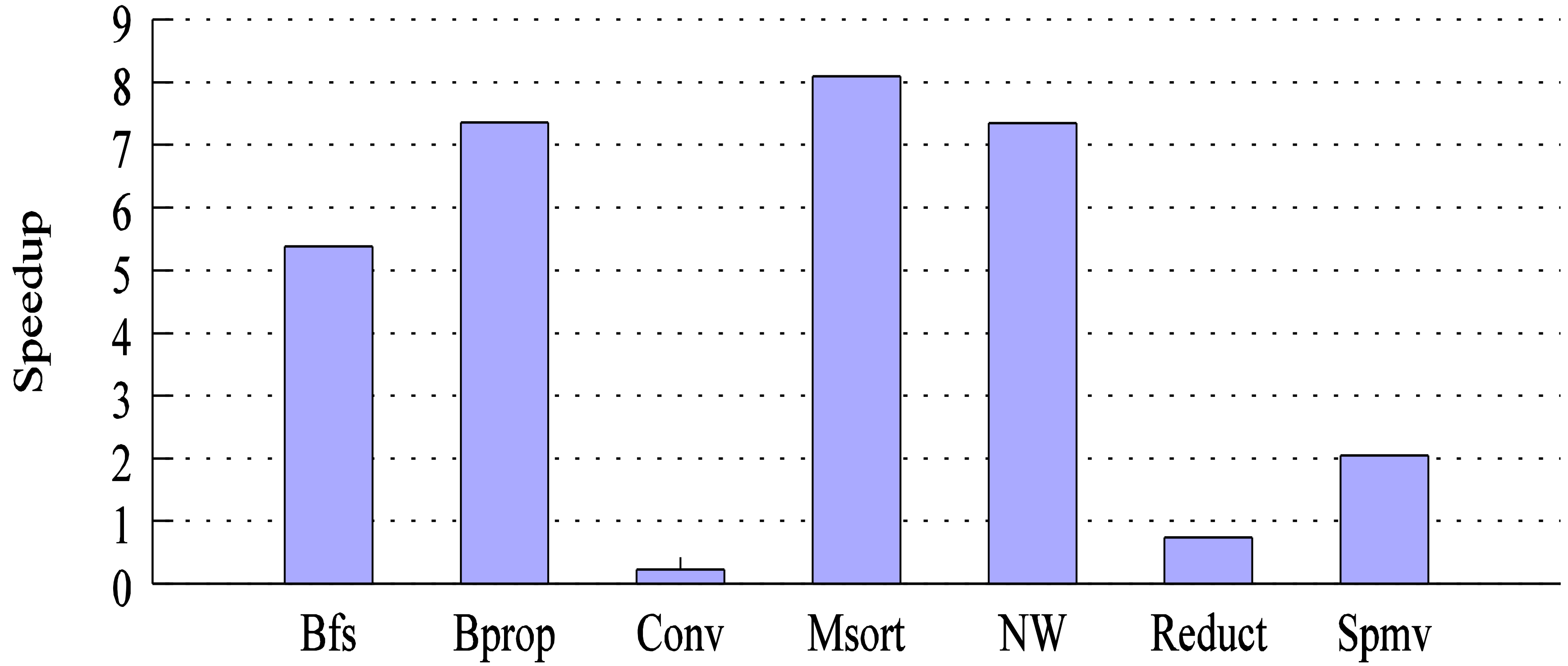
- 40+ students in fall 2010 joint UIUC/UMD course
- <1X speedups using OpenMP on 8-processor SMP
- 8x-25x speedups on 64-processor XMT FPGA prototype.

But, what's the big deal of 64 processors beating 8?

- Silicon area of 64 XMT processors ~= 1-2 SMP processors

But, what is the performance penalty for easy programming?

Surprise benefit! vs. GPU [HotPar10]



1024-TCU XMT simulations vs. code **by others** for GTX280. < 1 is slowdown. Sought: similar silicon area & same clock.

Postscript regarding BFS

- 59X if average parallelism is 20
- 111X if XMT is … downscaled to 64 TCUs

Problem acronyms

BFS: Breadth-first search on graphs

Bprop: Back propagation machine learning alg.

Conv: Image convolution kernel with separable filter

Msort: Merge-sort algorith

NW: Needleman-Wunsch sequence alignment

Reduct: Parallel reduction (sum)

Spmv: Sparse matrix-vector multiplication

Ease of Programming

- **<u>Benchmark</u>** Can **any** CS major program your manycore?

Cannot really avoid it!

<u>Teachability demonstrated so far for XMT [SIGCSE' 10]</u>

- **To freshman** class with 11 non-CS students. Some prog. assignments: merge-sort*, integer-sort* & sample-sort.

Other teachers:

- Magnet HS teacher. Downloaded simulator, assignments, class notes, from XMT page. **Self-taught**. **Recommends**: Teach XMT first. Easiest to set up (simulator), program, analyze: ability to anticipate performance (as in serial). Can do not just for embarrassingly parallel. Teaches also OpenMP, MPI, CUDA. See also, keynote at CS4HS' 09@CMU + **interview with teacher**.

- **High school & Middle School** (some **10 year** olds) students from underrepresented groups by HS Math teacher.

*Also in Nvidia's Satish, Harris & Garland IPDPS09

<u>Teachability</u> almost as good as quantitative - Time to confuse us with facts

- At what stage can you teach the programming of your approach

MS	Intro to PRAM
9-10th grade	Conjecture: Basic PRAM
11-12th grade	[Limited validation]
Lower division undergrad	
Upper division undergrad	Where are: BSP, CUDA?
Grad core	
Grad advanced	

‘Dark secret’, EduWrkshp@PPoPP’11: majority teachers of parallelism never require hard speedups. Try to find in TCPP curriculum.

- Allows claims for order-of-magnitude improvement

Middle School Summer Camp Class Picture, July' 09 (20 of 22 students)

What is the reason for low speedups and bigger programming efforts on other platforms?

- Not caused by inefficient algorithms or their implementation
- Rather: by the mismatch between the algorithmic model and the architecture
- Strengthens the case for XMT as an efficient, general-purpose, easy-to-program many-core

In summary,
Claim 1 Easier parallel programming is feasible

- Current curricula: unconvincing speed-ups, if at all. Hodgepodge of topics that fail to get across ideas
- Finding a new programming model appears to be the holy grail for some. Hard to believe that after 40 years of research there is still some hidden programming model that once discovered will solve all productivity problems? What about models whose success was already shown in some subfields such as PRAM?

Claim 2 Ease-of-Programming requires a separate program

Real success of parallel computing

i.e., parallelism becomes the dominant computing paradigm

cannot happen before ease-of-programming (EOP) becomes a 1st class concern for parallel architecture

Many long agreed that EOP is very important. Evidence:

1980s	PRAM – simple algorithms model
1991	'Parallel SW crisis'
2002-2011	DARPA HProductivityCS – development-time + run-time
2011	CSTB: only heroic programmers succeed. Need new 'stack'

So, why didn't it happen?

EOP tends to be consumed by other objectives. Easier to measure performance and power. Wed EoP to one of them ➔ forget about EoP

HPCS: $650M, but not even a benchmark

New: ExaScale and other (CSTB): # FLOPS + power ➔Non starter for EOP

➔ EoP requires a **separate** program

Imagine two algorithms for the same problem: ParAlg and SerAlg

Suppose that:

- W(ParAlg) = c W(SerAlg), and 10 <= c <= 100
- The breakdown of the time ParAlg, T(ParAlg) in total includes a fraction which is serial and its serial execution amounts to 1% of T (SerAlg)

What speedup relative to SerAlg on a uniprocessor can you expect from ParAlg on:

- 16-core CPU?
- Cutting edge GPU?

For more information: Please hear James Edwards on Wednesday

Conclusion

- All parallel machines to date failed to generate a broad base of application programmers because they have been too difficult to program
- All current government and industry efforts known to us in the domain of parallel computing, even as it is becoming mainstream, are at investing in yet another generation of too-hard-to-program parallel machines
- Rather than sidestep ease-of-programming, or treat it as afterthought, PRAM appears to have been the only algorithmic theory, and XMT the only active holistic research project in the community today, that handle ease-of-programming of many cores as a first class constraint
- For perspective See Ran Ginosar's Parallel Computing course, Fall 2010, http://webee.technion.ac.il/courses/048874/ [VLSI/Plurality startup]
- Insufficient competition in the West opened an unprecedented opportunity for order-of-magnitude improvement
- Opportunity for ambitious competitor (with deep pockets) to take over
- 2020 Vision: **Chinese (or Indian) dominate processors & systems** in mainstream computers and major devices,

  unless warning on need for EoP-driven systems is acted upon