

# Workshop on Theory and Many Cores

## May 29, 2009

Introduction, by Uzi Vishkin, University of Maryland

### Program committee

[Guy Blelloch](#), Carnegie-Mellon University

[Phil Gibbons](#), Intel

[Arch Robison](#), Intel

[Leslie Valiant](#), Harvard University

[Uzi Vishkin](#), University of Maryland

\*Slides are a bit loaded due to NSF request for self-contained slides

### Sponsors

[The University of Maryland Institute for Advanced Computer Studies \(UMIACS\)](#), [Electrical and Computer Engineering Department](#), and the [Center for Computational Thinking, Carnegie-Mellon University](#).

# Commodity computer systems

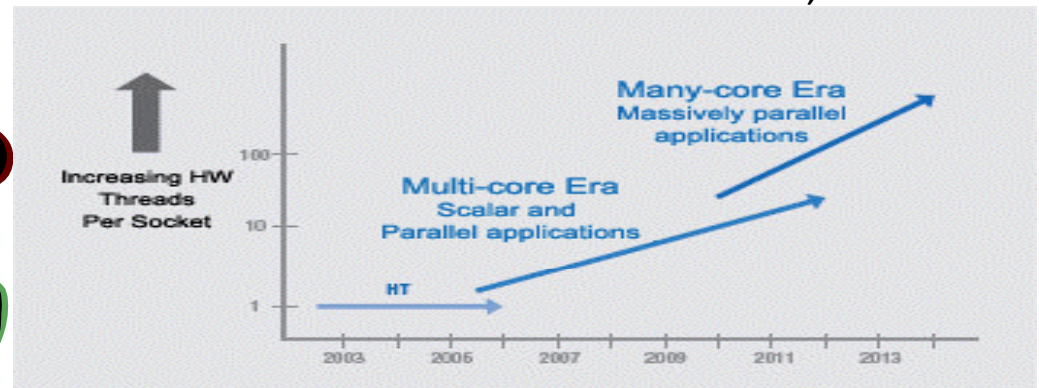
1946→2003 General-purpose computing: **Serial**. 5KHz→4GHz.

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. But, why is not everybody playing?**

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

Programmer's IQ? Flat..

Intel Platform 2015, March05



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

Letter grade 2003 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**”. Whose **F** was it: vendors' HW or SW people? Order of invited talks: SW to HW.

**What would be today's grade? If higher, is beyond better pain killers?**

# Emerging General-Purpose Many-core Computing Era

- Still in flux. Not yet clear how the many-core processor of the future will be built or programmed.
  - ➔ Aspiration: “convergence”. But: **How?**
- Two issues for which it is not yet clear what vendors propose and/or whether their solution is good enough: programmer’s model (algorithmic & programming thinking process) & scalability.

Non-conformist terminology:

- “parallel programmer’s model”: Total of 3 Google hits (all UV).  
“parallel programming model”: ~60K hits. Wikipedia: **A parallel programming model** is a set of software technologies to express parallel algorithms and match applications with the underlying parallel systems... The ultimate goal is to improve productivity of programming. Worded algorithms out of productivity: WHY?

Impact: has this mechanical focus reduced pressure on HW designers from planning around the programmer and his/her computational thinking as a constraint (and make needed progress)? Kept theorists away? and now not readily available for teaching.

# Possible roles of theory vis-à-vis Parallel Programming

- The *parallel programmer's model problem*: ease of programming and good performance. Arguably, the hardest nut.
  - What parallel algorithmic model? (In the past, theory endorsed PRAM algorithmics.) What role should the algorithms model play in parallel programming:
    - Like serial: provide useful knowledge. [Serial algorithms (model and analysis) are taught for the first time only in the 4<sup>th</sup> semester or later. ]
    - More integral part: stage in a “work flow”, enforcing “programmer’s model” rather than “programming model”.
- “Programmability”. “Not easy” → what’s an **F** grade among friends?! Are CS systems communities doing their best to measure or benchmark programmability? One exception was 2002+ DARPA HPCS (high productivity computer systems) program. Assumed HPC vendors will make progress. What difference did it make \$600M later?

Thoughts: 1. Qualitative: what is the list of things the programmer needs to account for in order to program your machine (algorithm concurrency, decomposition)? 2. Teachability necessary condition for programmability. Quantitative: Benchmark machines by: How teachable using them is?

Modeling & teaching how to think is what theorists do.

# More thoughts

General: Theory plays an important role in CS research and education. Parallelism: more involved and requires greater intellectual insight → many-core CS got to be tied even closer to theory.

To vendors: who will teach the parallel algorithms & data structures course so that people will be able to program your machine? Sooner: How will you figure out how teachable using your machine is? (Anecdote: NVidia's ex-CTO tried himself at UIUC).

To theorists: Many-core is where the action in CS is. In the past, theorists found parallelism attractive, and you probably will too.

To funding agencies: Can the Federal IT investment risk relying only on HW coming down the pike from vendors? Can more diversified HW investment mitigate risk of another F?

Publishability Pitfall: Paper on solving Problem A on Machine X, but not on Machine Y may mean: no solution for Machine Y, OR too easy for Machine Y.

One objective for T&MC: point out opportunities beyond parallels to the role of theory for serial computing. Next slide:

# Possible roles of theory vis-à-vis Computer systems

## **Descriptive** (2-4 earlier talks?)

- Take system(s) provided by vendors as given.
- Model/understand them.
- Work towards getting the most (performance) out of them.

[There was no need to do more with serial systems.]

**What is “right” for you?**

**What is “right” for others?**

It is up to each of you. T&MC objective: better informed outlook.

Some good questions to ask in this space: Has it been tried before? Did it work (acceptance, implementability)? Have conditions changed?

## **Prescriptive**

40+ yrs: **F** on programmer’s model. Legitimate impatience: Wait longer for vendors?

- Theory: Mitigate in a model (PRAM).
- Provide specs for a new system. UMD: HW prototype to meet specs.

Vendors: we will build advanced ASIC system if shown speedups on apps/workloads that can be developed only after .. advanced ASIC system is built.

➔ Realizing HW diversity (so that “natural selection” can pick the best solution) requires NSF funding.

Theorists: Model, assess & compare HW and HW options. Voice opinion.

# Workshop on Theory & Many Cores

## Program ,fay 29, 2009

8:30 Light breakfast

9:00 Welcome and brief introduction

9:15 Parallel Algorithm Issues in TBB, Arch Robison, Intel

10:00 Coffee Break

10:30 A Bridging Model for Multi-Core Computing, Leslie Valiant, Harvard University

11:15 Theory: Asleep at the Switch to Many-Core, Phillip B. Gibbons (Intel Research Pittsburgh)

12:00 Lunch

1:30 Opportunities in Many Core Algorithms, Guy Blelloch, Carnegie-Mellon University

2:15 The PRAM-On-Chip Proof-of-Concept, Uzi Vishkin, University of Maryland

3:00 Coffee break

3:30 Optimal Speedup on a Low-Degree Multi-Core Parallel Architecture (LoPRAM), Reza Dorrigiv, Alejandro López-Ortiz, and Alejandro Salinger, University of Waterloo

3:40 Algorithm design for multicore processors, Theory and applications, Peter Krusche and Alexander Tiskin, The University of Warwick.

3:50 A Model of Computation for Map Reduce, Howard Karloff, AT&T Labs, Siddharth Suri , and Sergei Vassilvitskii, Yahoo! Research

4:00 Parallel Phase Model for Manycore Parallel Machines, Zhaofang Wen, Sandia National Labs, and Junfeng Wu, Syracuse University

4:10 Parallel External Memory Model, Lars Arge, University of Aarhus, Michael Goodrich, and Nodari Sitchinava, University of California – Irvine

4:20 Short break

4:30 Panel discussion: What will be the role of parallel algorithmic thinking in getting many core computing to stable state, and once it gets there? What will be the similarities and differences in the role that parallel algorithms will play for parallel programming relative to serial algorithms and serial programming? What are the key algorithmic/theory questions to address? What will become (or already are) the theory "classics" worth teaching for the long term and what are temporary artifacts of day or obsolete assumptions? How can we get reignite interest in parallelism in the theory community?

# For entertainment

Depth-first search in parallel Given a connected undirected graph  $G(V,E)$  and some node  $v \in V$ , we all know the depth-first search (DFS) method. DFS visits its vertices in the following recursive order. For some vertex  $u$ , performing  $\text{DFS}(u)$  consists of the following: if vertex  $u$  was visited before then call  $\text{DFS}(u)$  ends; otherwise, visit vertex  $u$  and then call  $\text{DFS}(w)$  in sequence for every vertex  $w$  that is adjacent on  $u$ . Depth-first search begins by calling  $\text{DFS}(v)$ . Assume  $|V|=n$  and  $|E|=m$ .

**Show how to run DFS on a PRAM in  $O(n)$  time and  $O(n + m)$  work.**

If you need a hint, go to Exercise 36 in:

[www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf](http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/classnotes.pdf)

Compare with: Some problems are very difficult to parallelize, although they are recursive. One such example is the [depth-first search](http://en.wikipedia.org/wiki/depth-first_search) of graph. [http://en.wikipedia.org/wiki/Parallel\\_algorithm](http://en.wikipedia.org/wiki/Parallel_algorithm)