PRAM on Chip

What to do with all this hardware? Could the PRAM-On-Chip architecture lead to upgrading the WINTEL performance-to-productivity platform?

PRAM on Chip: A Quest for Not-So-Obvious Non-Obviousness

U. Vishkin, vishkin@umiacs.umd.edu.
With graduate students (Balkan, Berkovich, Dascal, Gu, Naishlos, Nuzman, Wen), post-doc (Kupershtok), micro-architecture (Franklin, Jacob), compiler (Barua, Tseng), VLSI/power (Qu), and graphics (Olano; student: Wang) experts.

My strategic research technical objective, since 1979:
Seek to improve general-purpose single task completion time by parallelism.

Highlights

• Technological opportunity - bandwidth and latencies orders of magnitude better on chip.
Note 1: Unprecedented bandwidth; better latencies.

• The PRAM (parallel random-access model) –
the "ultimate" for scalable parallel thinking/programmability.
BUT, was impractical for the 1990s multi-chips: overheads for managing parallelism too high.
Note 2: Could not be done before.

• PRAM-on-Chip – a **concrete** virtual-PRAM design...
NDA with Major Elec Des company led to prelim cost est.
Note 3: Practical! Building is in D stage!

• Wait a minute: but how do you manage parallelism?
Through reaxiomatization of von-Neumann's ("mathematical machines") 1946 design:
- Control (program counter + stored program) of computer systems same as in the 1940s. New: **upgraded**.
- "Billion transistor" chip, for: **memory, interconnect and low overhead management of parallelism**.

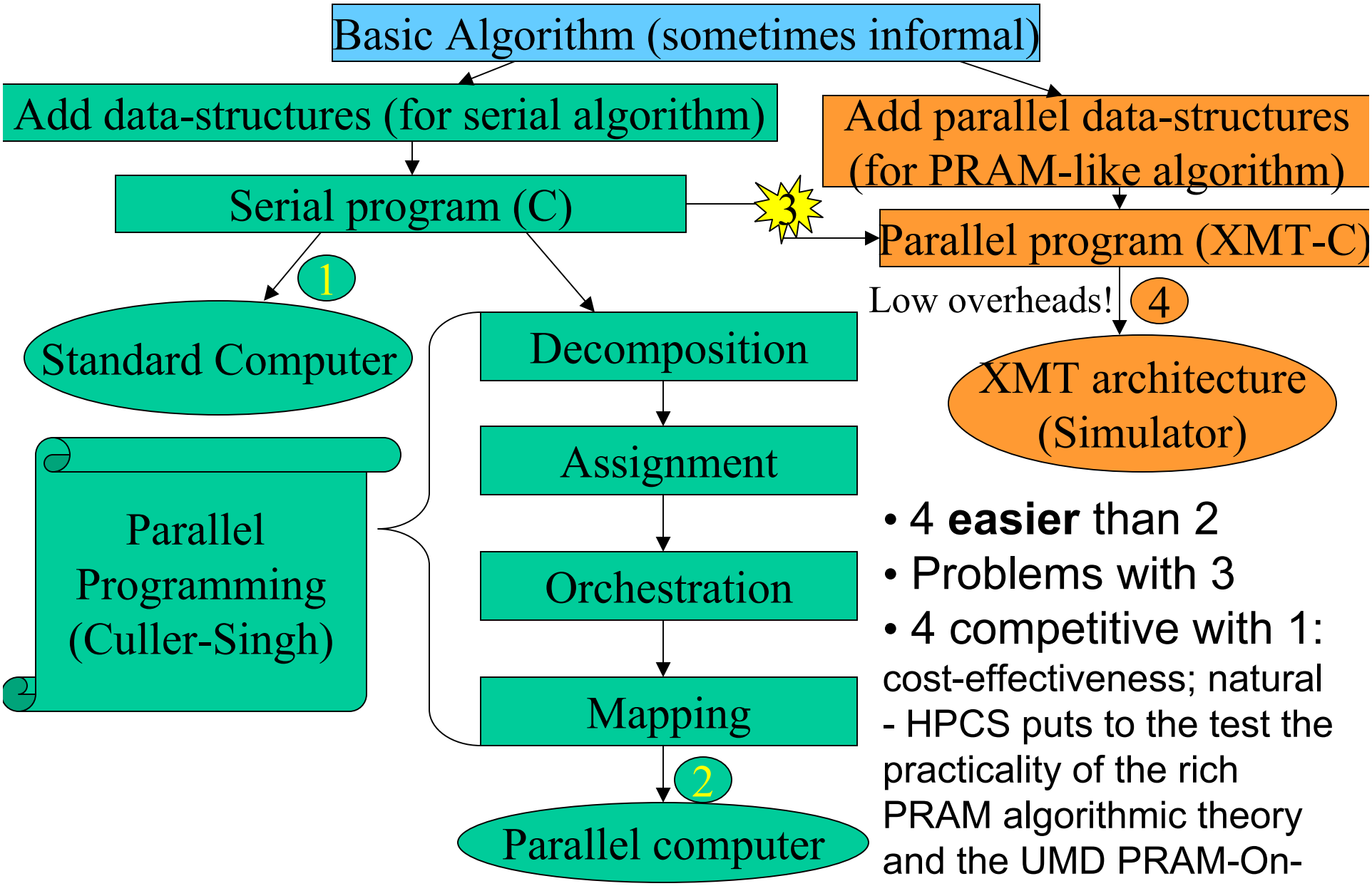
Note 4: Reaxiomatization for better performance.

• Speed-up: **X50-100** over serial for general-purpose applications. For: single task completion time.
Note 5: performance orders of magnitude better.

• Translate computing performance to "productivity":
- Performance programming (next slide).
- Application programming (following slide); e.g., simpler (standard) APIs for games+graphics yet good performance.
- Applications:
(i) Molecular simulation, e.g., drug design. Aspire: Nanosecond per step; feasible to simulate many more steps.
(ii) General purpose: "iceberg effect".
Note 6: Easier (BS, not PhD) performance programming.
Non-programmer (app expert) programmers.
Market/science/defense apps.

• "Trends in VLSI technology scaling demand that future computing devices be narrowly focused to achieve high performance and high efficiency", Opens: Smart Memories (Stanford), ISCA2000's. Heard from others.
Note 7: much higher level of abstraction in XMT. Easier to program. Argue: still good performance.

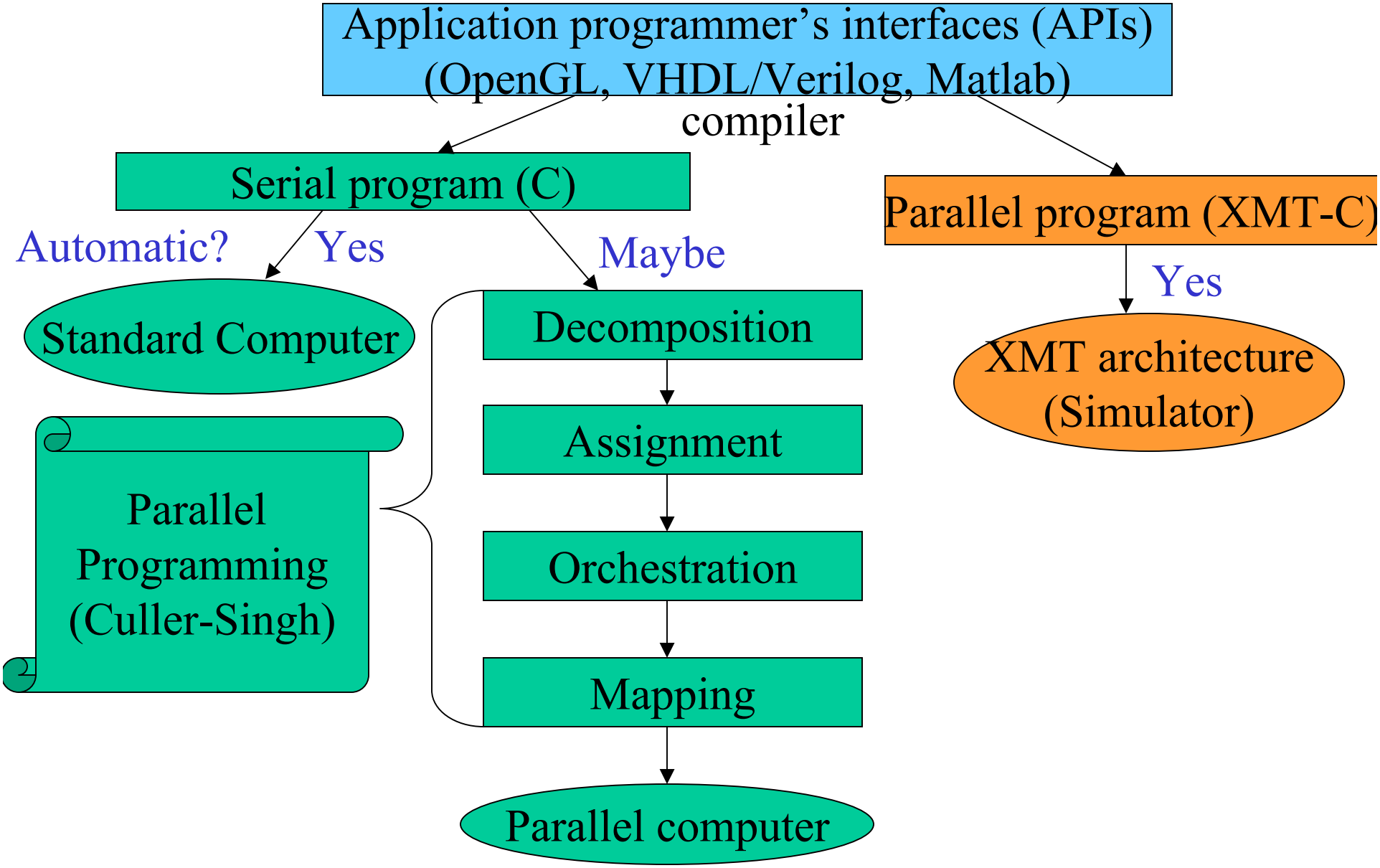
PERFORMANCE PROGRAMMING & ITS PRODUCTIVITY



- 4 **easier** than 2
- Problems with 3
- 4 competitive with 1:

cost-effectiveness; natural - HPCS puts to the test the practicality of the rich PRAM algorithmic theory and the UMD PRAM-On-Chip project

APPLICATION PROGRAMMING & ITS PRODUCTIVITY



The PRAM: parallel random-access (virtual machine) model

• Ideal PRAM: latency for arbitrary number of memory accesses, same as for one access.
• Premise: algorithmicist states (or, does not hide..) what can be done concurrently.
• Algorithmic knowledge-base *2nd only to serial algorithms*. Simplest parallel model.

Example

Given: (i) All commercial airports in the world. (ii) For each, all airports to which there is a non-stop flight
Task: Find the smallest number of flights from DCA to every other airport
Principle of PRAM algorithm
Parallel Step i: Given all the airports which require $i-1$ flights, find (concurrently!) all those that require i flights
Observe:
(i) "Concurrently": only change to serial algorithm
(ii) Inherent serialization: S - number of parallel steps
Total number of operations: T - O(number of flights)
Orders of magnitude gain relative to "serial":
$O(T/S)$ decisive also relative to coarse-grained par

An Overall Design Challenge

- Showed algorithm scalability.
- Hardware scalability: put more of the same
- ... but, how to manage parallelism coming from a programmable API?

Spectrum of Explicit Multi-Threading (XMT) Framework

Algorithms $--> $ architecture $--> $ implementation.

- XMT: strategic design point for fine-grained parallelism
- New elements are added only where needed

Attributes

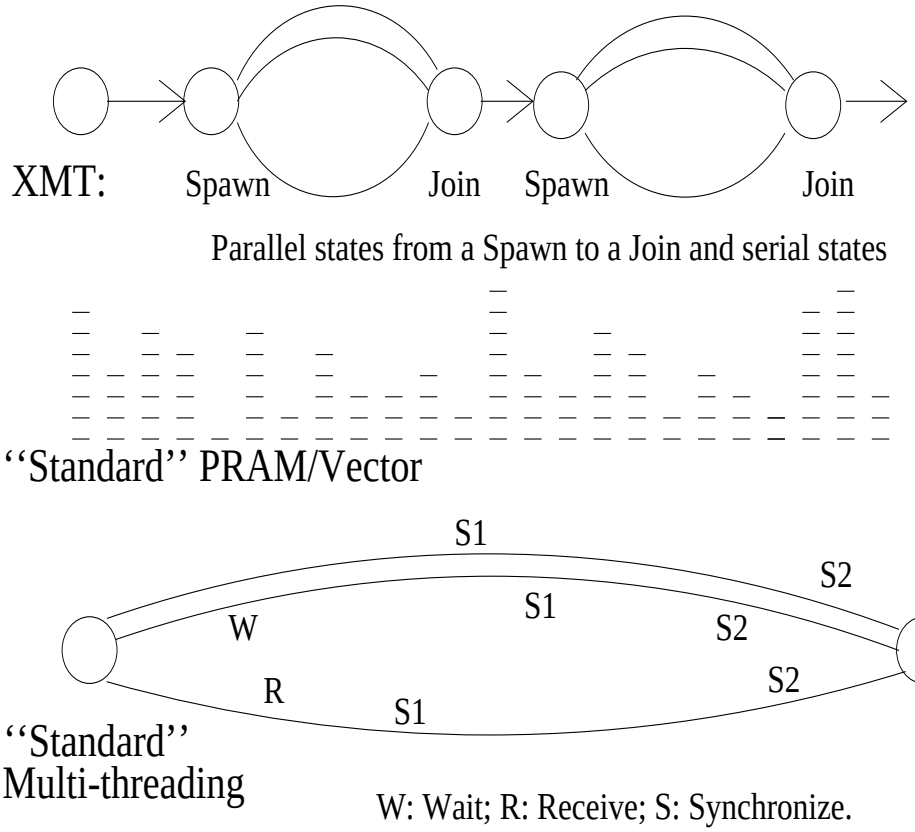
- Holistic

A variety of subtle problems across different domains must be addressed:

- Understand and address each at its correct level of abstraction

Snapshot: XMT High-level and assembly languages

For contrast: vector and standard multi-threading

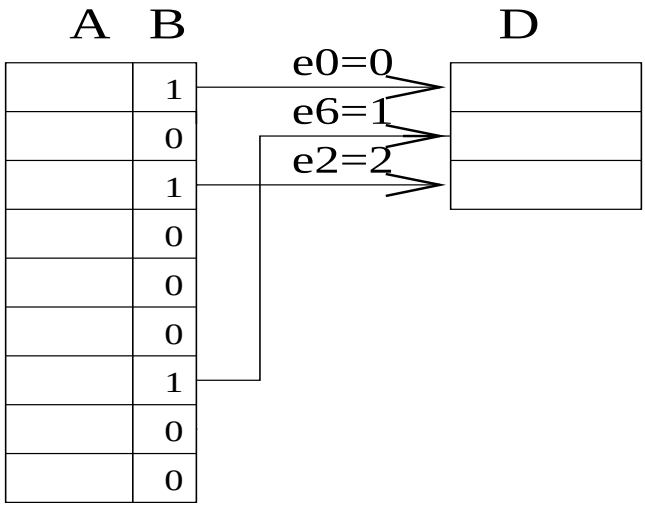


Cartoon Spawn creates threads; a thread progresses at its own speed and expires at its Join. Synchronization: only at the Joins. So, virtual threads avoid busy-waits by expiring. New: Independence of order semantics (IOS).

Programming interface: XMT-C and Assembly

The array compaction (artificial) problem

Input: Array $A[1..n]$ of elements; binary array $B[1..n]$.
Map in some order all $A(i)$, where $B(i) = 1$, to array C.



The array compaction problem.
Array B: bit values for the compaction.
For the program below: e0, e1 and e6
are the e values for threads 0, 2 and 6; x is 3.

XMT-C: High-level language

Single-program multiple-data (SPMD) extension of standard C. Includes Spawn and PS - a multioperand instructions.

Algorithm level (high-level program)

```
int x = 0;
Spawn(0,n)   /* Spawn n threads; $ ranges 0 → n − 1 */
{ int e = 1;
  if (B[$] == 1)
     { PS(x,e);
       D[e] = A[$] }
  }
n = x;
```

Notes: (i) PS is defined next (think F&A). See results for e0,e2, e6 and x. (ii) Using C-style scoping, Join instructions are implicit.

XMT Assembly Language

Standard assembly language, plus 3 new instructions: Spawn, Join, and PS.

The PS multi-operand instruction

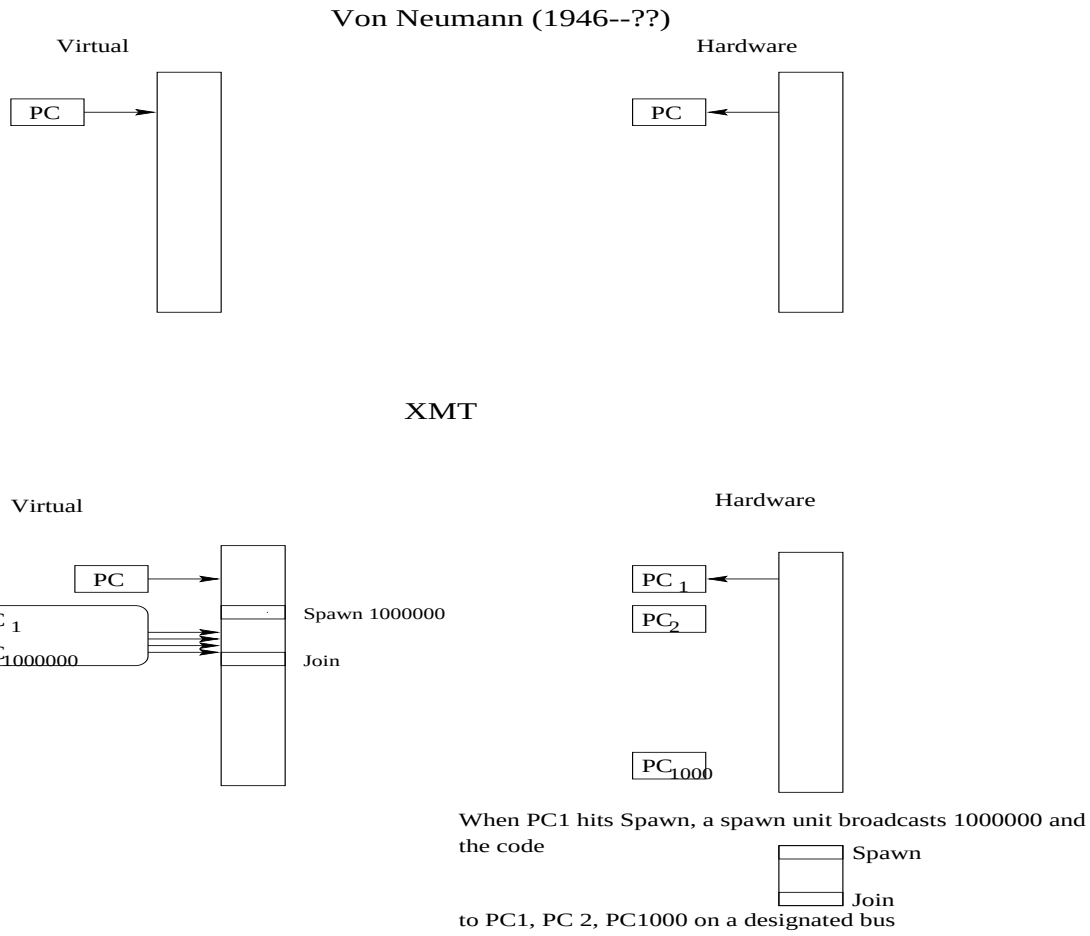
New kind of instruction: *Prefix-sum* (PS).
Individual PS, $PS\ Ri\ Rj$, has an inseparable ("atomic") outcome: (i) Store $Ri\ +\ Rj$ in Ri, and (ii) store original value of Ri in Rj.
Several successive PS instructions define a multiple-PS instruction. E.g., the sequence of k instructions:
$PS\ R1\ R2$; $PS\ R1\ R3$; ...; $PS\ R1\ R(k+1)$
performs the prefix-sum of the *base* $R1$ and the *elements* $R2, R3, ..., R(k+1)$ to get: $R2\ =\ R1$;
$R3\ =\ R1\ +\ R2$; ...; $R(k+1)\ =\ R1 + ... + Rk$;
$R1\ =\ R1 + ... + R(k+1)$.
Idea: (i) Several ind. PS's can be combined into one multi-operand instruction. (ii) Executed by a **new multi-operand PS functional unit**.

Mapping PRAM Algorithms onto XMT

(1) PRAM parallelism maps into a thread structure
(2) Assembly language threads are not-too-short (to increase locality of reference)
(3) the threads satisfy IOS

Machine extensions to the von Neumann model

Program counter + stored program:
– a remarkable Darwinistic success story.
Pragmatic: Extend rather than head-on competition.

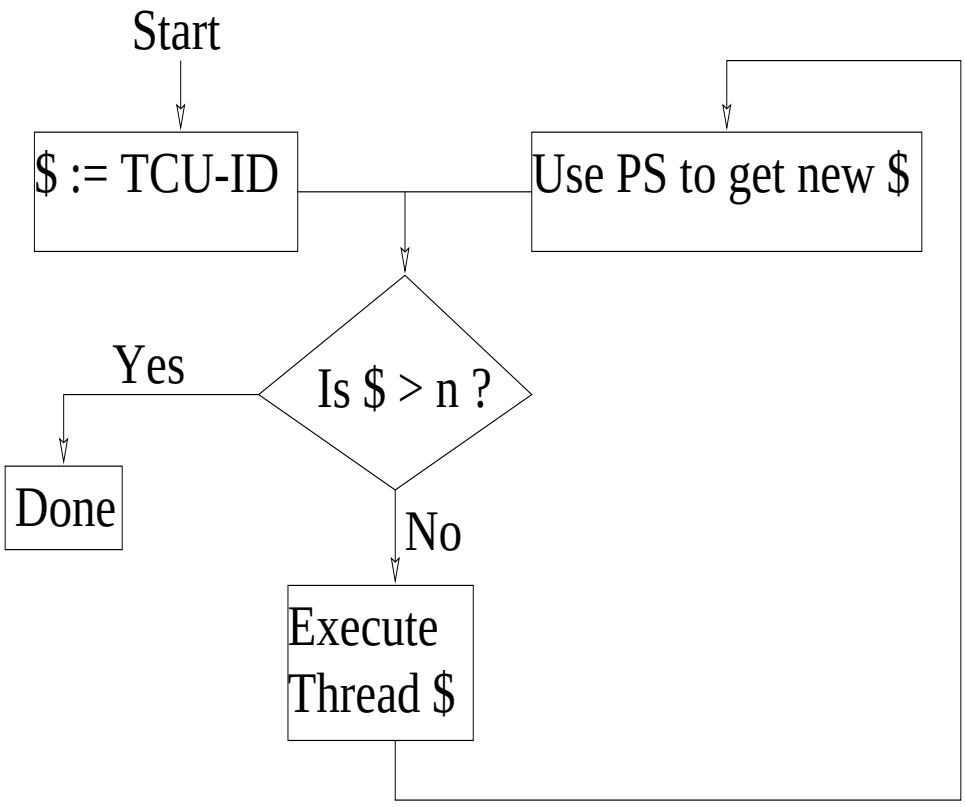


If successful, Encyclopedia Britannica will need to update its Computer entry...

Snippet of a machine execution model

Given: a Spawn of n threads. The hardware determines which virtual thread to execute next in a distributed fashion

The program of a thread control unit (TCU)



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

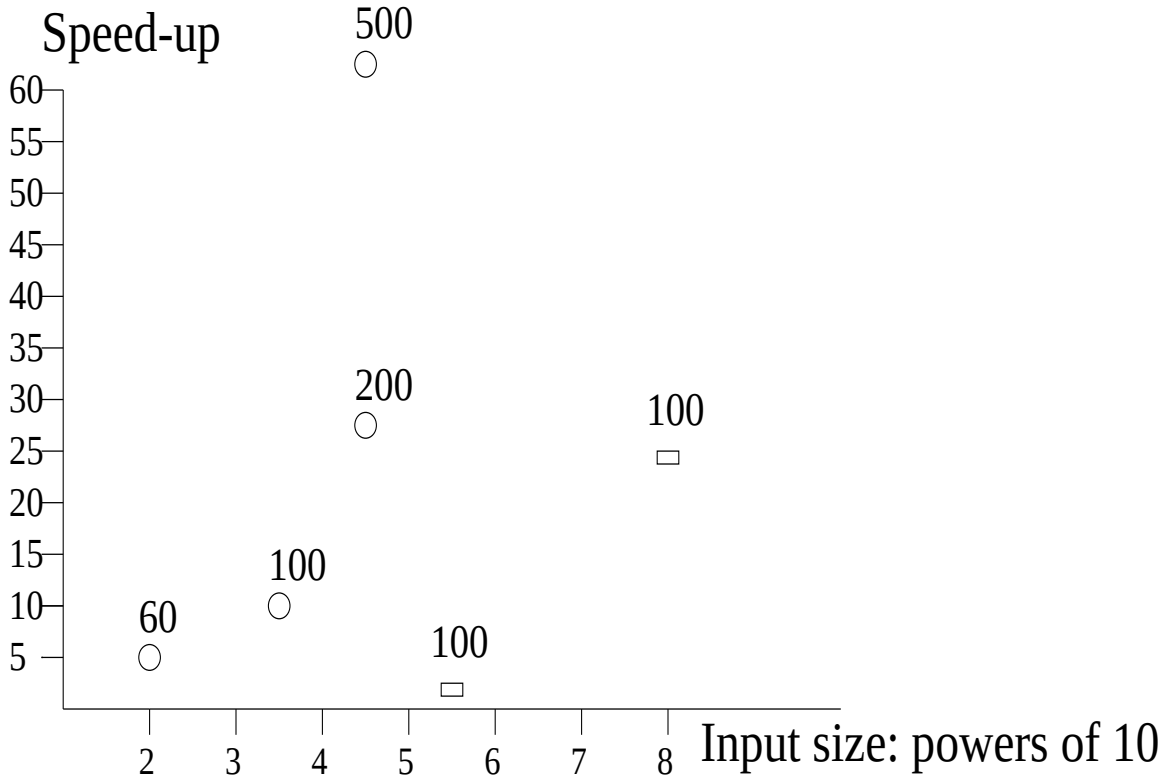
• High bandwidth memory architecture.
- Use hashing to partition the memory and avoid hot spots.
- Understood, BUT (needed) departure from current practice.

• High bandwidth on-chip interconnects

• Allow infrequent global synchronization (with IOS).
Attractive: lower energy.

Empirical results

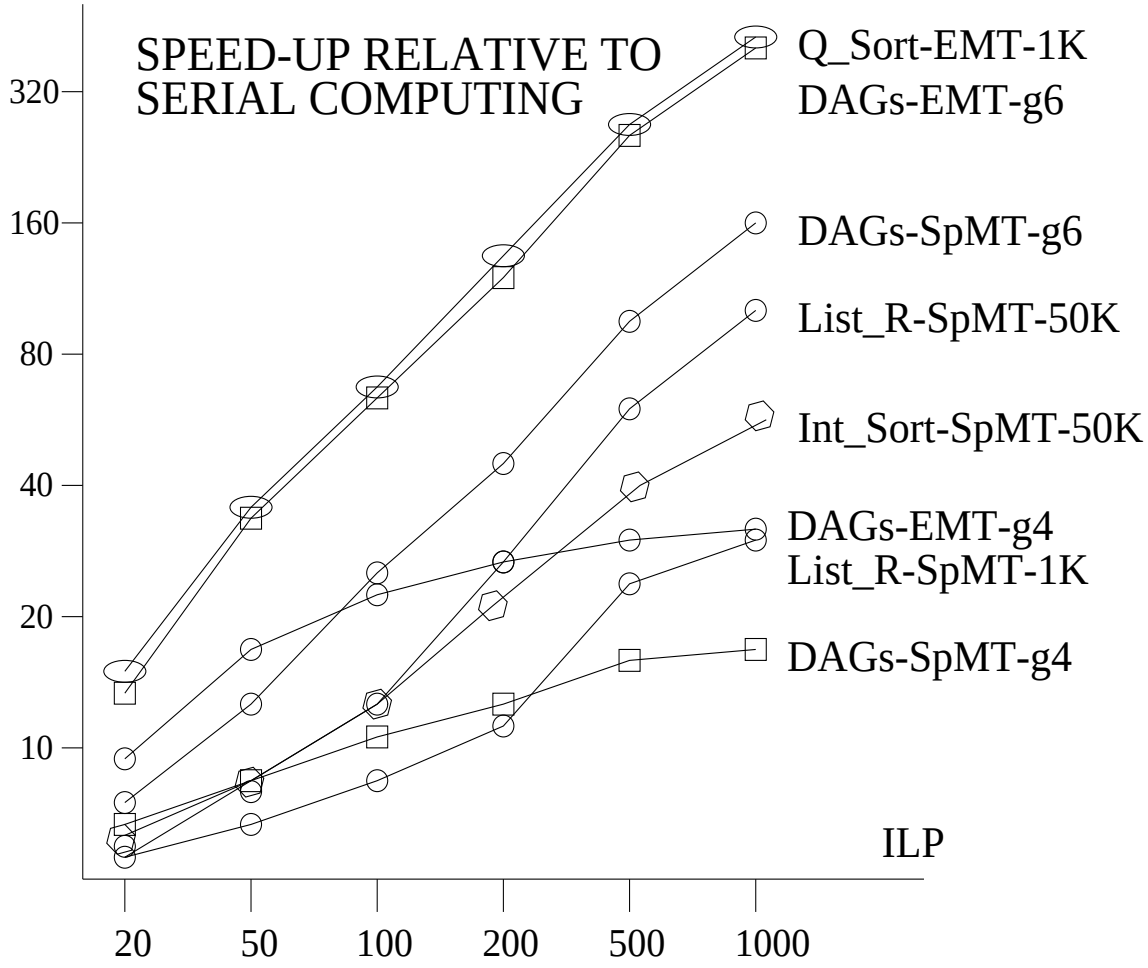
• **Parallel computing versus XMT**. List ranking results for the Intel Paragon reported in Sibeyn-97 versus XMT results.



LEGEND: 100 ▭ Means Multi-processing with 100 processors.

200 ○ Means XMT with ILP of 200

- **Relative to serial computing**.



4 last transparencies: much more data

Conclusion

Objectives
- General-purpose computing
- Single task completion time

Predicaments in the past
- High overheads for managing parallelism in multi-chip multiprocessing. Dictated difficult "decomposition-first" parallel programming.
- Good returns on using on-chip growth for caches.
Now diminishing...
- Architecture techniques? Mined out. Popular micro-procs: max #instructions issued per clock–flat. Decade-long stagnation!

Outcomes 2-3 years: speed-ups > 30. 3-5 years: speed-up > 100.
Some applications (productivity):
- Molecular simulations (e.g., for drug design, protein folding). Suppose 10^{14} steps need to be simulated. A rate of step per nanosecond could be possible. Exciting: the whole simulation takes a day instead of 1000 days with multi-chip multiprocessing.
- Hardware-enhanced software development kits (SDKs)

for some application domains (e.g., graphics, desk-top data bases). Expert programmers will implement easy to use APIs. Such APIs alleviate barriers to entry to creative content producers who will generate greater demand; note: Intel's Vtune, Sony PS2, etc., appear to lead in the opposite direction.
- Applications are generally unlimited!
An "Iceberg Effect" in high-performance general-purpose computing systems: only a small fraction of the actual applications are visible at build-time.

The WINTEL paradigm upgrade catch-22
1. Intel: but who will develop OS and SDKs/APIs?
2. Microsoft: who will build?
3. No action? performance improvement from VLSI only!

hurting both (and IT as a whole).
Needed prototype study: Applications, programmer's productivity, architecture. Successful? involve the other.

Documentation www.umiacs.umd.edu/ vishkin/XMT/+
First generation: SPAA'98, WAE'99 (special issue).
Second generation: MTEAC'00 at MICRO (MTEAC Best Paper Award), HIPS'01, SPAA'01 (and special journal issue, 2003)

"Sermon" part of the talk: a proposed characteristic of "contribution"

You are told about a new technical idea and react: "but this is so obvious, I wonder how I missed it".
Problem: since hindsight is 20/20, how to reason about non-obviousness when it is not-so-obvious?

Found out recently: the US patent law practice provides an insight. That law protects inventions that meet three requirements: non-obviousness (most challenging to establish), utility, and novelty.
Legal definition of (non-)obviousness: A patent may not be obtained though the invention is not identically disclosed, if the differences relative to prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art. But how is non-obvious demonstrated?
Using circumstantial evidence:
- experts stated disbelief in what the invention allows.
- unaddressed need.

"PRAM is unrealistic" statements showed to patent examiner that PRAM-On-Chip was non-obvious.

virtues of not-so-obvious non-obviousness

You can get a patent for it.
Simple. Many will be able to understand it.
Likely impact: may find its way to a textbook/encyclopedia, people may actually end up using it.
That is, more people may actually be interested in understanding it and... they will actually succeed.

Backup slides:

Legend of the Lazy Performance Programmer

Current practice (some repetition):
- SP2 non trivial programming
- Intel's Vtune
- Write your own memory manager, Game Developer, 2/2002
- Current parallel programming: decomposition-first. Programmability problems

The ideal: non-"programming" APIs. Many new products for games/graphics. Performance needed.

The need ("handwaving")

Even the world best race car builder needs to rely on ships to transport his/her cars. The "ship" here takes you from the 1946 strategic design point to the new one (buildability of a first design), and the explicit parallel programmability (theory and results). The "race car" are all the elegant static and dynamic optimizations, and the productivity enhancing SDKs/APIs. You need a ship (out-of-the-box mean of transportation) to cross water (the paradigm upgrade) to put your race car to use

An EDA company wants to make some money; what's the big deal?

[CS-99] Parallel Computer Architecture asserts:

- the PRAM algorithmic model is DESIRABLE, but NOT feasible (for the 1990s type multi-chip MPPs)
- a virtual-PRAM machine (that can look to the programmer like the PRAM) becoming FEASIBLE will be a BREAKTHROUGH for general-purpose computing

What about other alternatives? e.g. streaming

1. Case studies: (a) the recent (official) Open GL Shading Language is in a strict SPMD (XMT-like) model. As conditional execution and looping unfold differently for different threads, the linear flow of control of streaming architectures should have a problem dealing gracefully with the computation. Crucial to next generation reactive quality visualization (NSF-STTR).
(b) Similar issues appear to dominate HDLs.
(c) Even more problematic with richer data-dependent (pointer-based) flow. Question: inspiring GUPS?
2. Using PRAM terminology, let W be the total number of operations of an algorithm and T its running time. To a first approximation, running times on both XMT or a streaming architecture should behave like $c(W/T) + dT$ (reminiscent of Amdahl's Law). We generally expect that $d_{streaming} \gg d_{XMT}$ and that there will be less of a difference with respect to c values. That is, if the dT term dominates, XMT will do significantly better. Streaming: great for some niche applications. But, will there be enough parallelism beyond that?

But others upgrade multi-chip multi-processing
Compare with unichip-centered upgrade

Will be glad to emulate/import.
So far, I have seen no reason why XMT should fall behind;
not even spatial locality...

But, What Is Different?

High-end parallel computer systems boast: number of operations-per-second. Overall, indeed great in scaling up the total number of operations per given time.

However:
They fall behind even cheap high-end commodity serial platforms when it comes to running a standard program, unless they can extract from it a large amount of coarse-grained parallelism–a task known generally as the "parallel software crisis".

PRAM-On-Chip detours this known programmability impediment to broadening the applicability of current parallel architectures. Architecture is designed directly for the amounts of hardware that can fit on a single chip by 2006. Not incrementally upgrading decades-old architectures.
With its powerful patented support for very fine-grained, irregular parallelism it comes ahead of both current serial and parallel platform for a broad range of applications. Simulated program executions have shown dramatic performance gains over conventional processor architectures. Could have applications for interactive visualiza-

tion and virtual reality, CAD, control of switch fabric, ballistic missile defense, weather forecasting, radar processing, weapons design, as well as high-end simulations such as ones in current and future computationally demanding drug design applications, especially where the total number of rounds that needs to be simulated in a given time exceeds current parallel platforms.

Experimental Methodology

- Simulator
  - SimpleScalar parameters for instruction latencies
  - 1, 4, 16, 64, 256 TCUs
- Configuration:
  - 8 TCUs per cluster
  - 8K L1 cache
  - banked shared L2 cache 1MB
- Programs rewritten in XMT
  - Speedups of parallel XMT program compared to best serial program
    - parallel applications: scalability to high levels
    - speedups for less parallel, irregular applications

First Application Set

Domain	Program	source
Scientific Computation	1.jacobi	
	2.tomcatv	SPEC95
Linear Algebra	3.mmult	Livermore Loops
	4.dot	Livermore Loop
Database	5.dbscan	[]
	6.dbtree	MySQL
Image processing	7.convolution	[]

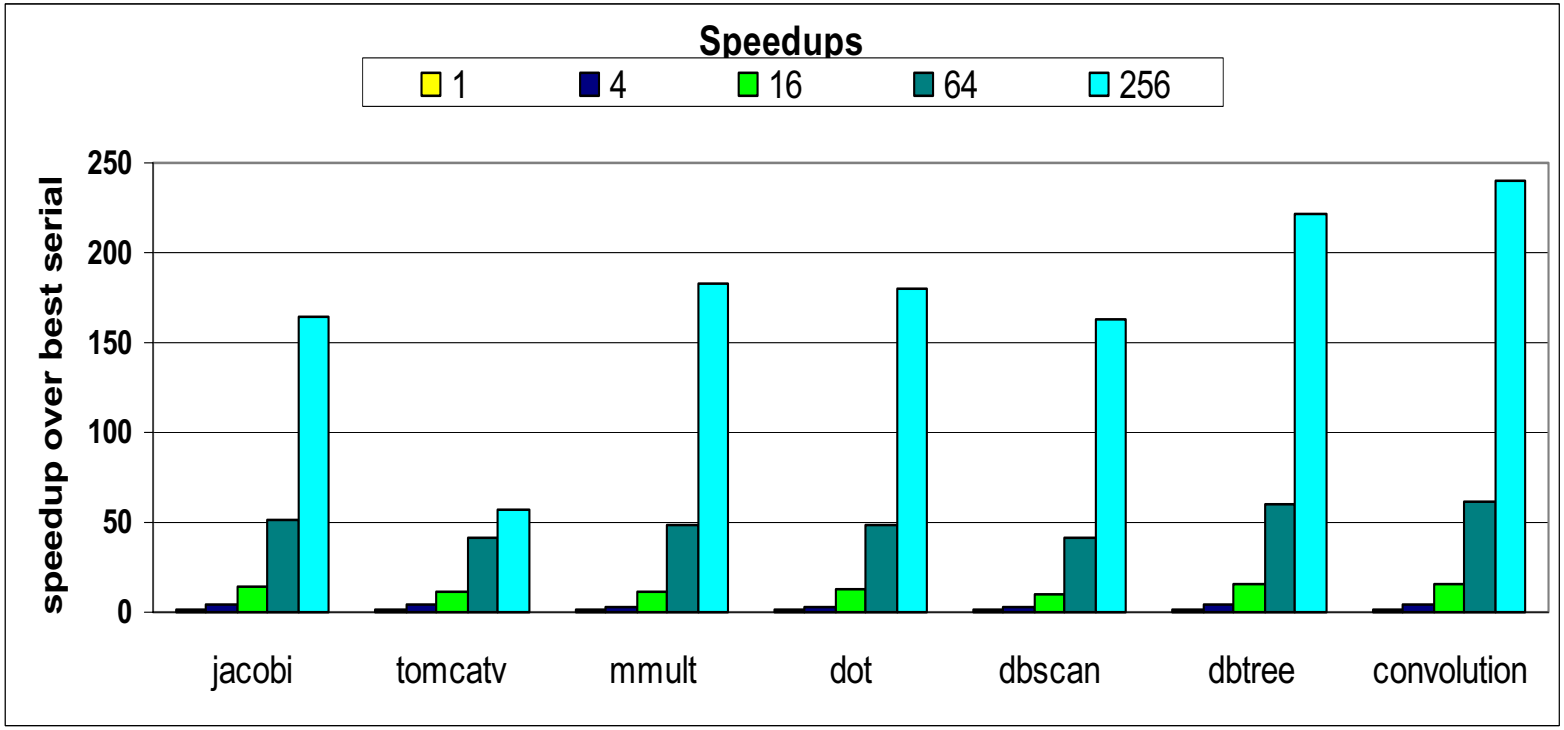
- Computation:
  - regular,
  - mostly array based,
  - limited synchronization needed

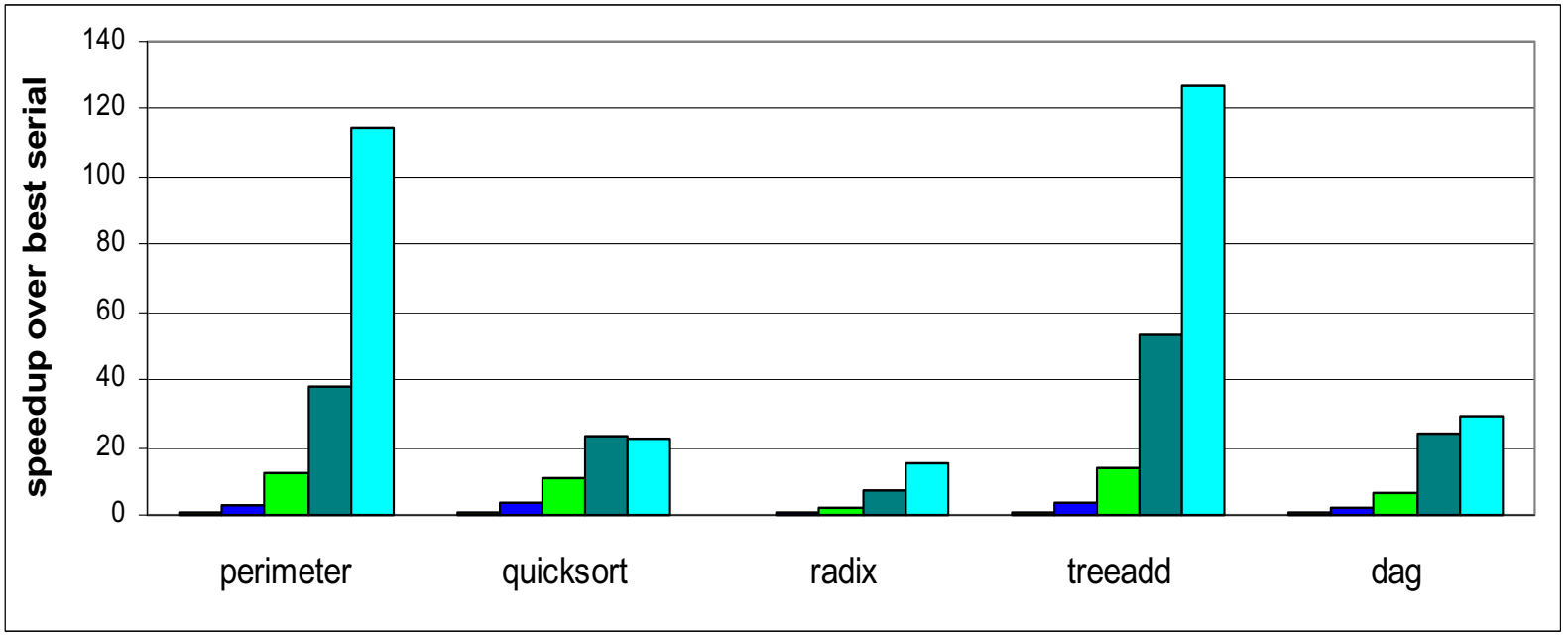
Second Application Set

Domain	Program	source
Sorting Algorithms	1.quicksort	
	2.radixsort	(SPLASH)
graph traversal	3.dag	
	4.treeadd	Olden
image processing	5.perimeter	Olden

- Computation:
  - irregular,
  - unpredictable
  - synchronization needed

Summary


Speedups
1
4
16
64
256
speedup over best serial
250
200
150
100
50
0
jacobi
tomcatv
mmult
dot
dbscan
dbtree
convolution


speedup over best serial
140
120
100
80
60
40
20
0
perimeter
quicksort
radix
treeadd
dag