# Dealing with Hard Serial Bottlenecks

- Patrick H. Madden
  SUNY Binghamton Computer Science Dept.
- pmadden@acm.org

- Warning: slides are intentionally

## Sorry to keep beating this drum…

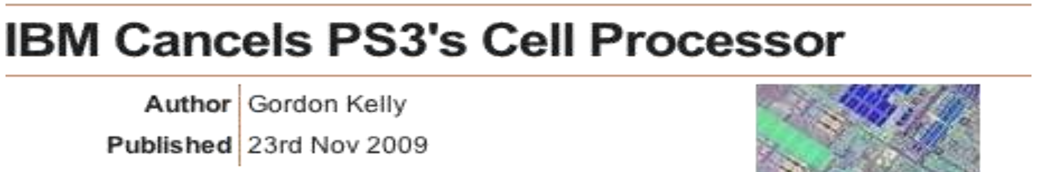
Amdahl Olukotun Arvind Gustafso

ICCAD '07 panel

… but this stuff is important.

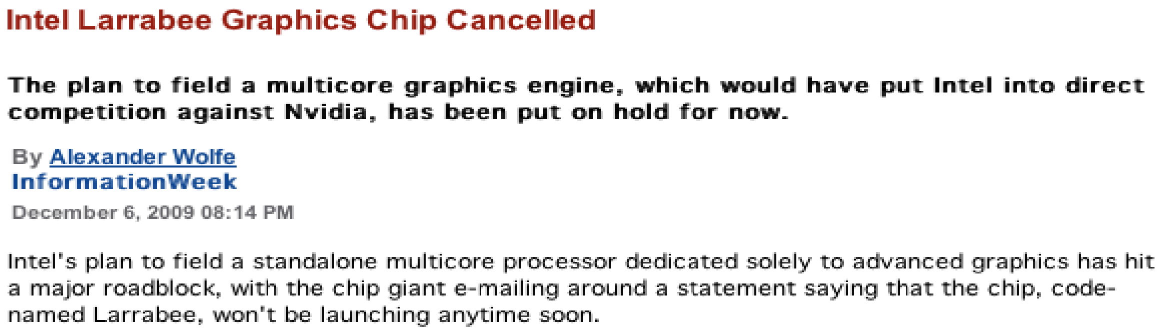
# It seems like only 5 years ago, most people thought I was insane….

- EDP05 talk – parallel is a recipe for disaster….



**IBM Cancels PS3's Cell Processor**

**Author** Gordon Kelly
**Published** 23rd Nov 2009

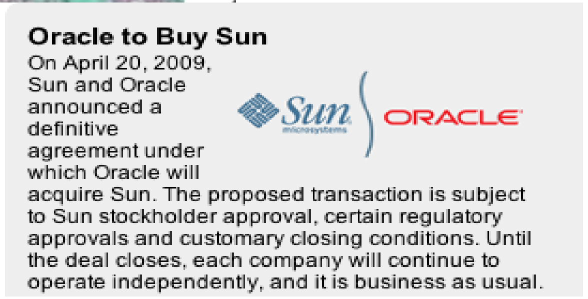


**Intel Larrabee Graphics Chip Cancelled**

**The plan to field a multicore graphics engine, which would have put Intel into direct competition against Nvidia, has been put on hold for now.**

By Alexander Wolfe
**InformationWeek**
December 6, 2009 08:14 PM

Intel's plan to field a standalone multicore processor dedicated solely to advanced graphics has hit a major roadblock, with the chip giant e-mailing around a statement saying that the chip, code-named Larrabee, won't be launching anytime soon.



**Oracle to Buy Sun**

On April 20, 2009, Sun and Oracle announced a definitive agreement under which Oracle will acquire Sun. The proposed transaction is subject to Sun stockholder approval, certain regulatory approvals and customary closing conditions. Until the deal closes, each company will continue to operate independently, and it is business as usual.

## The Future of Computing

INSIDE • TECHNOLOGY

- The A4 processor is a **single-core CPU**, making it either an ARM Cortex A8 or a single-core variant of the A9. Most likely, it's an A8.

# Background Reading

- http://www2.dac.com/front_end+topics.aspx?article=17&topic=1
- “Rethinking Parallel” in the Technical Articles section of the new DAC web site
- Massively parallel computing makes no sense to me.  There is small scale parallelism everywhere (2, 4, maybe 8 cores), but massive parallelism is commercially useless

## Hard Serial Bottlenecks

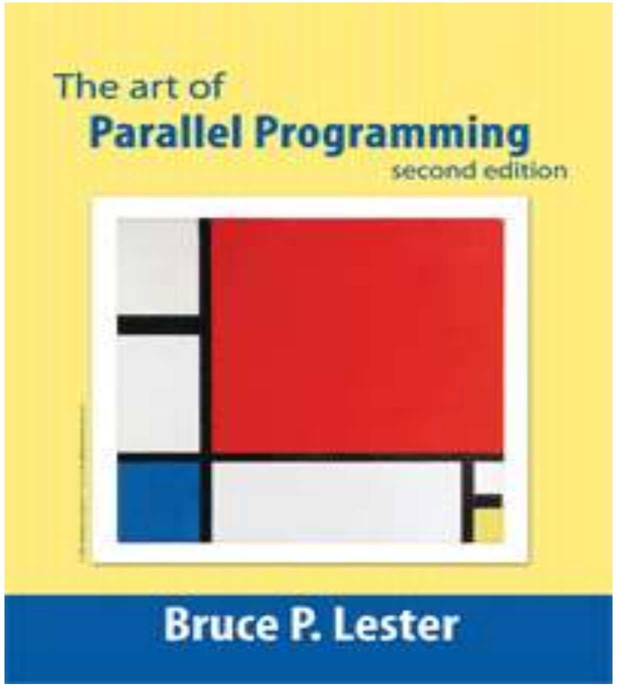
- Interesting problems where there are no competitive scalable solutions (and extracting any speedup is difficult)
- Why do we care? Amdahl’s Law… If 10% of our application relies on solving problems that are serial in nature, we get a maximum of 10X speedup.
  - We can make more cores… they’re just useless.
- Where can we find these serial bottleneck problems?
  - Surprisingly often in papers by parallel computing experts, who are too busy with their bold visions to see how wrong they are.

## Hard Serial Bottlenecks

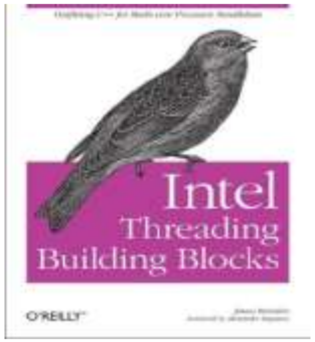
- Garland, DAC08: GPGPU parallel shortest paths algorithm
  - Parallel Bellman-Ford: O(E * V)
  - Serial Dijkstra approach: O(E + V log V)
  - **Serial is faster(!)**
- Jamsek, ASPDAC09: GPGPU rectangle overlap for lithography applications
  - Parallel brute force: O(n^2)
  - Serial computational geometry approach: O(n log n)
  - **Serial is faster(!)**

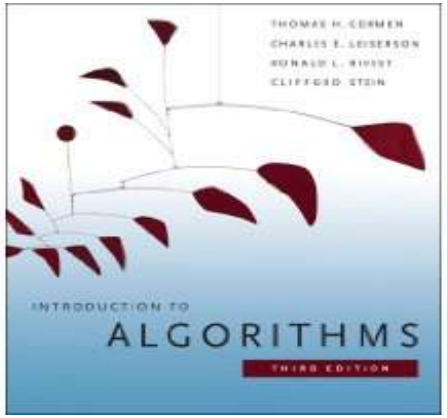
# More Examples

- Sorting
  - Lester advocates a parallel rank sort: O(n^2)
  - Serial quicksort: O(n log n)
  - **Serial is faster!**
- Shortest path
  - Same mistake as Garland DAC paper
  - **Serial is faster!**



# And even more....



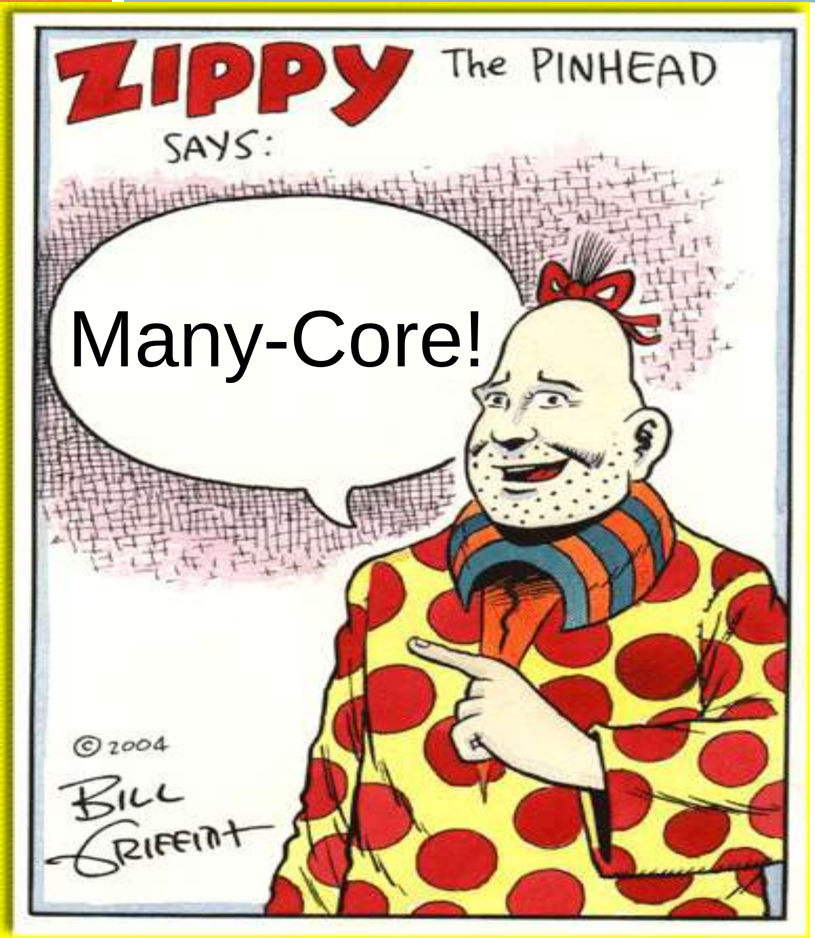


- Reinders TBB book (Intel), latest edition of CLRS text
  - Parallel recursive Fibonacci: O(2^n)
  - Serial alternatives: closed form, or O(n) dynamic programming
  - **Serial is faster!**
- Note: CLRS is absolutely clear about what a bad idea the parallel version is, and Reinders notes the “more efficient” approach in a side-bar

## This is a strange world

- Leading research groups are publishing material that is obviously, horrifically flawed
  - It’s gone through peer review, editors, program committees, and down the gullets of an audience one would expect to be at least a bit skeptical.
  - The errors are not subtle.  They’re right up front, in your face, and should be stuff that any person with an undergrad degree in computer science can catch.
- *What kind of science are we doing?*
  - If all it takes to get funding or a publication is to cook up some bogus results, what incentive is there to actually make an advance?
  - What exactly is going on with these “parallel advocates?”

# Are Parallel Advocates Stupid?



Many seem to have degrees from “good” schools. They can’t all be stupid.

## Are Parallel Advocates Corrupt?

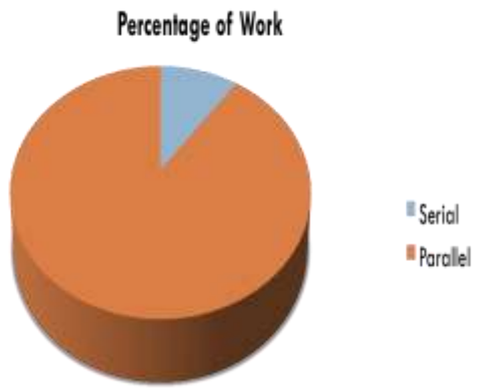
Intentionally fabricating results as a way to scam money out of the government and investors? It happens in science, but this sort of fraud would be difficult to sustain for 50 years.

## I think they're mostly deluded…

MONTY PYTHON'S
LIFE OF BRIAN
MONTY PYTHON'S LIFE OF BRIAN
GRAHAM CHAPMAN JOHN CLEESE
TERRY GILLIAM ERIC IDLE
TERRY JONES MICHAEL PALIN

*For over a decade,* ***prophets*** *have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit*

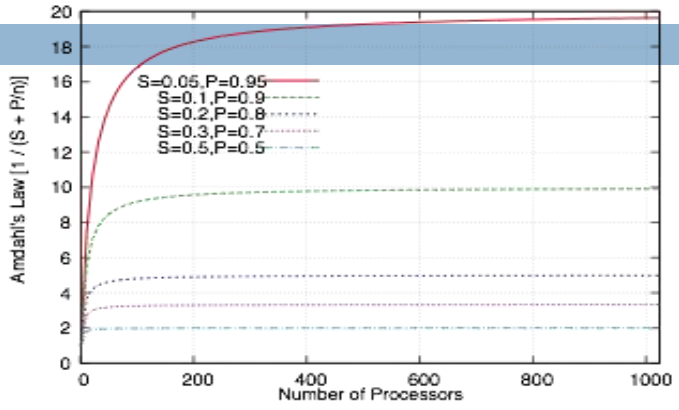
# Amdahl's Law



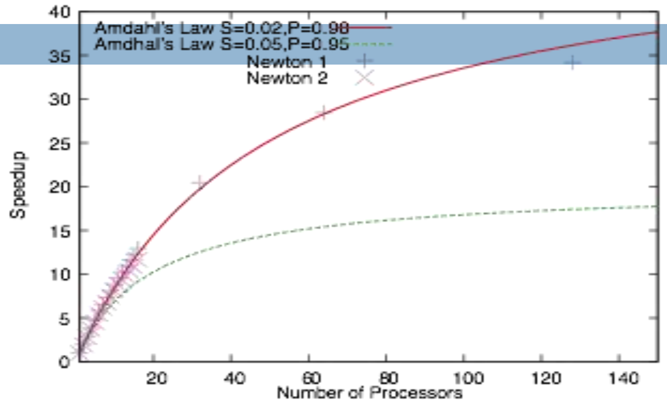
- The serial portion is real. Anyone who can't see this is in deep denial.

- Maximizing performance hinges on MINIMIZING the serial bottlenecks
- We can't expect linear speedup here; it's not sexy or flashy, but it is critical to do.
- The eternal problem: it's easy to get a bogus linear speedup, and has been encouraged by funding agencies, tenure and program committees…
- Why is Amdahl's Law so hard to understand?
  - **It is difficult to get a man to understand something when his job depends on not understanding it** – Upton Sinclair

# Realistic Speedup

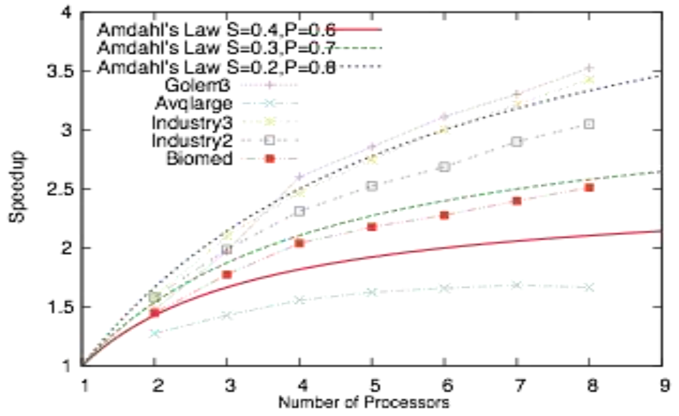


**(a) Amdahl's Law**

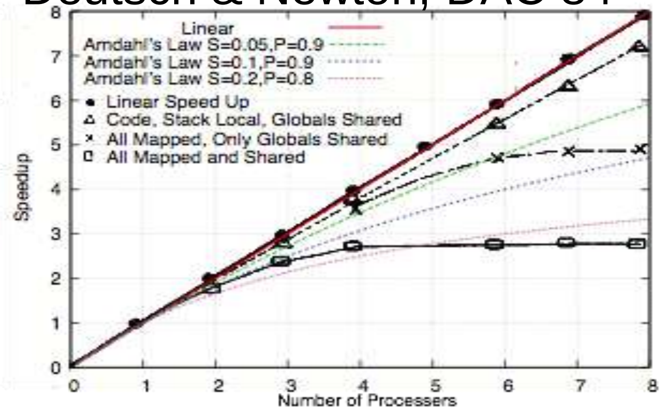


**(b) Circuit simulation[18]**

Deutsch & Newton, DAC 84



**(c) Recursive bisection placement [41]**

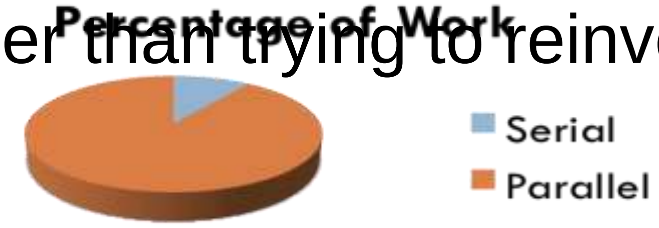


**(d) Linear programming [38]**

Khundakjie, ..., Madden, IEEE Clusters 2001 Jones & Schwarz, Computing Surveys '80

# Research Focus

- Critical to reduce the size of the "hard serial bottlenecks" while keeping the computational efficiency of the best algorithms
  - Looking for solutions with broad impact, not a one-off
  - Leverage algorithms with the best computational complexity rather than trying to reinvent the wheel



# Data Structure Co-Processing

- Many of the best algorithms rely on data structures
  - In particular, priority queues, trees, hash tables, and so on, require some compute time for insert, delete, ….
  - Opportunity to extract some parallelism; when inserting an item, the main thread doesn't need to wait for the data structure to reorganize
- Operations on the data structure may take tens or hundreds of cycles
  - We've looked at implementations on standard multi-core; while it works, we leave a lot on the table
  - Need low-latency connection, tight integration between the algorithm and a secondary processor

# Data Structure Co-Processing

- Data structures offer a clean separation between types of computational tasks, and offloading of work is certainly not new
- Questions from the early part of this research:
  - Is there enough potential for parallel work to offset the overhead of synchronizing with a secondary processor
  - How much of the work is "data structure," and how much is "main thread"
  - Is it possible to get a meaningful speedup here, to chip away at the hard serial bottlenecks?
- The integration of a data structure and it's algorithm can be very tight; we need to have efficient integration.

## Algorithm Test Bed

- Single source shortest path – a hard serial bottleneck
  - Bellman-Ford is out, due to computational complexity
  - No apparent alternative algorithms (e.g. Delta Stepping)
  - Dijkstra’s Algorithm
    - This is the best known algorithm, and is tightly integrated with a priority queue data structure
    - If it’s possible to extract some speedup here, it’s likely we can use the same trick on other tough problems
    - Our experiments use a basic binary heap for illustration; the idea applies to other types of data structures

# Dijkstra’s Algorithm

```
Initialize distances to infinity;
S = 0;
Q = Vertices;
while (Q not empty)
  u = ExtractMin(Q);
  for (each v adjancent to u)
    relax(u, v);
```

ExtractMin needs to reorganize the heap. Further, the relax step may update the distances to some vertices (also changing the structure of the heap). The key idea is to move this sort of work to a secondary processor.

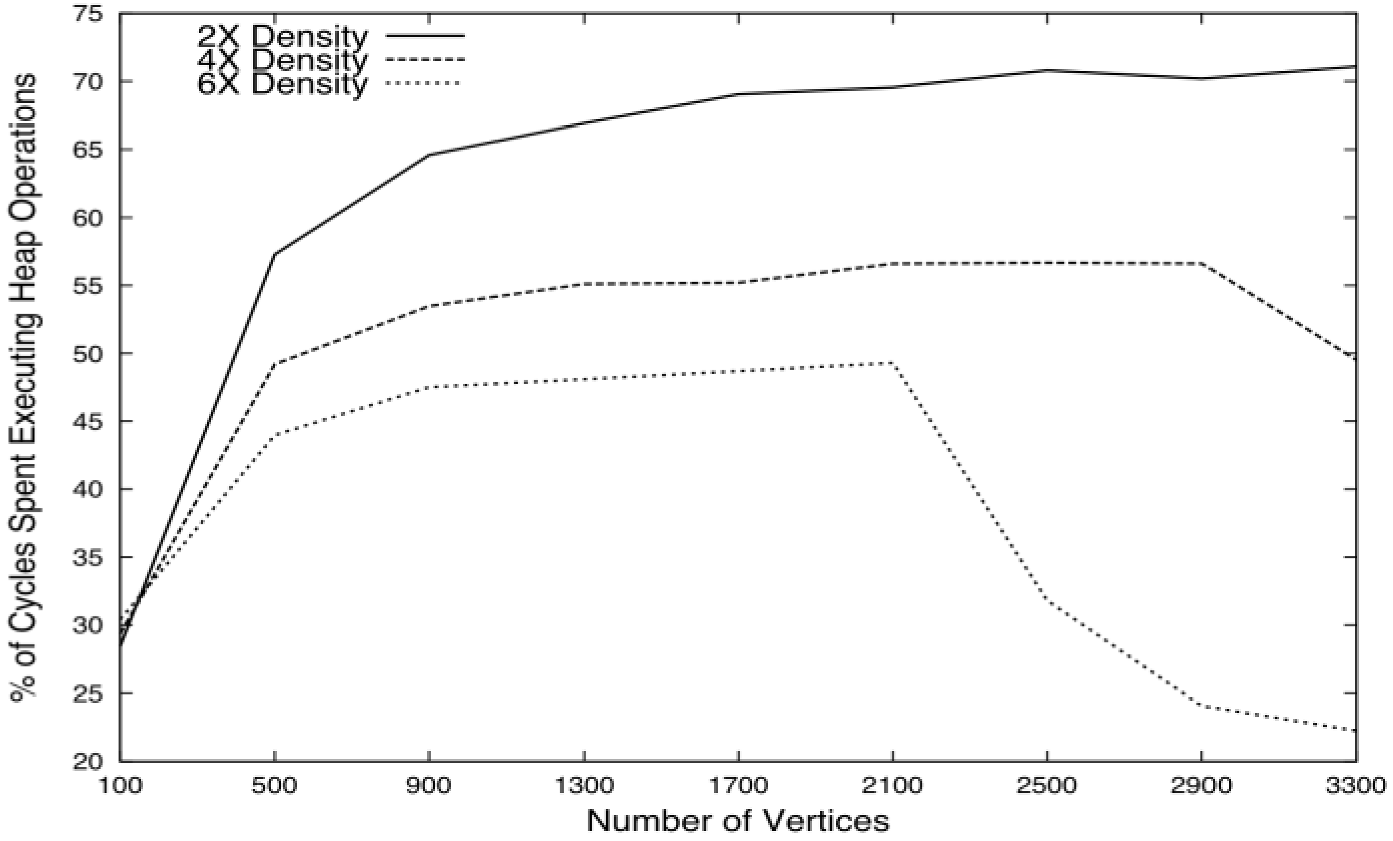
## Evaluation Methodology

- Used M-Sim simulator http://www.cs.binghamton.edu/~msim
- Simulated DSCP using dual-core and dual-threaded processors
- Each core is 4-way wide with 128-entry Reorder Buffer, 32KB I-L1 and D-L1 caches and 512KB L2 cache
- We assumed 300 cycles memory latency

## Simulated Graphs

- **Synthetic graphs**
  - **Sparse**: 100 to 3300 nodes with the number of edges being 2x, 4x and 6x the number of nodes
  - **Dense**: 100 to 2900 nodes with edge coverage of 15%, 45% and 75%
- **Real-life graphs**
  - Road map benchmarks from the 9th DIMACS Implementation Challenge
    - Full USA map, maps of individual states.

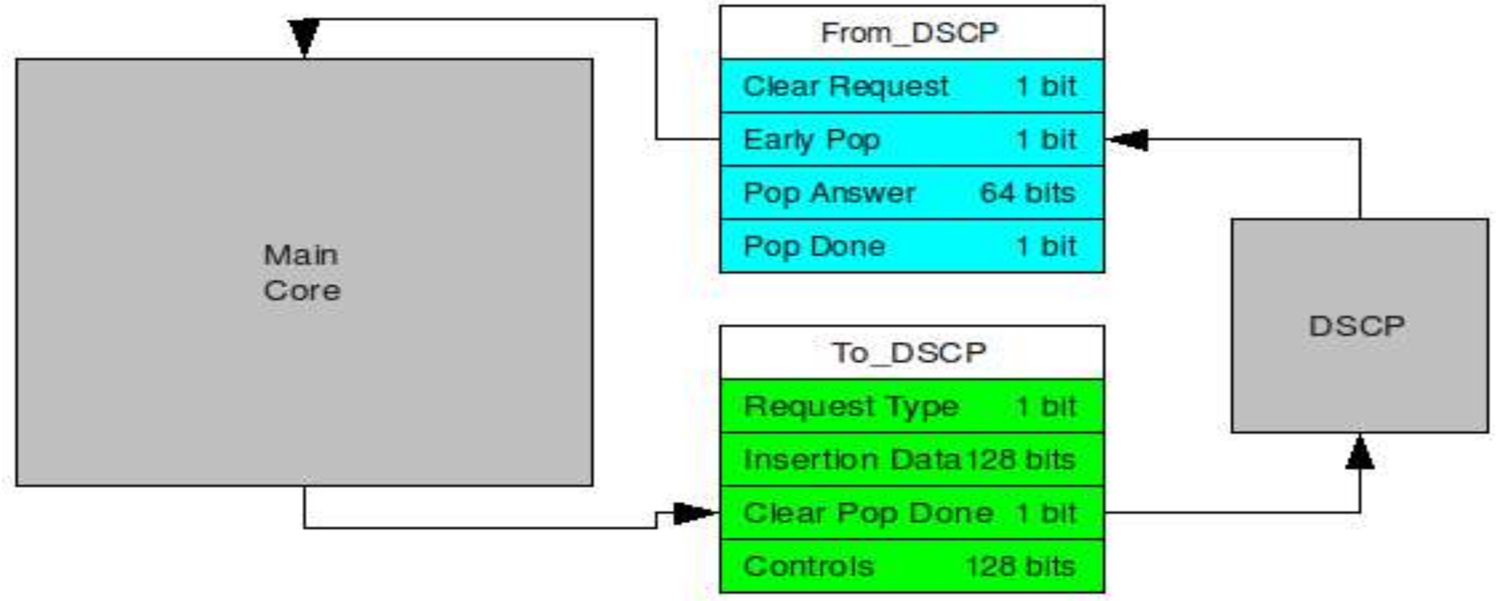
# Dijkstra’s Algorithm: How Much Time is Spent Processing Heaps?



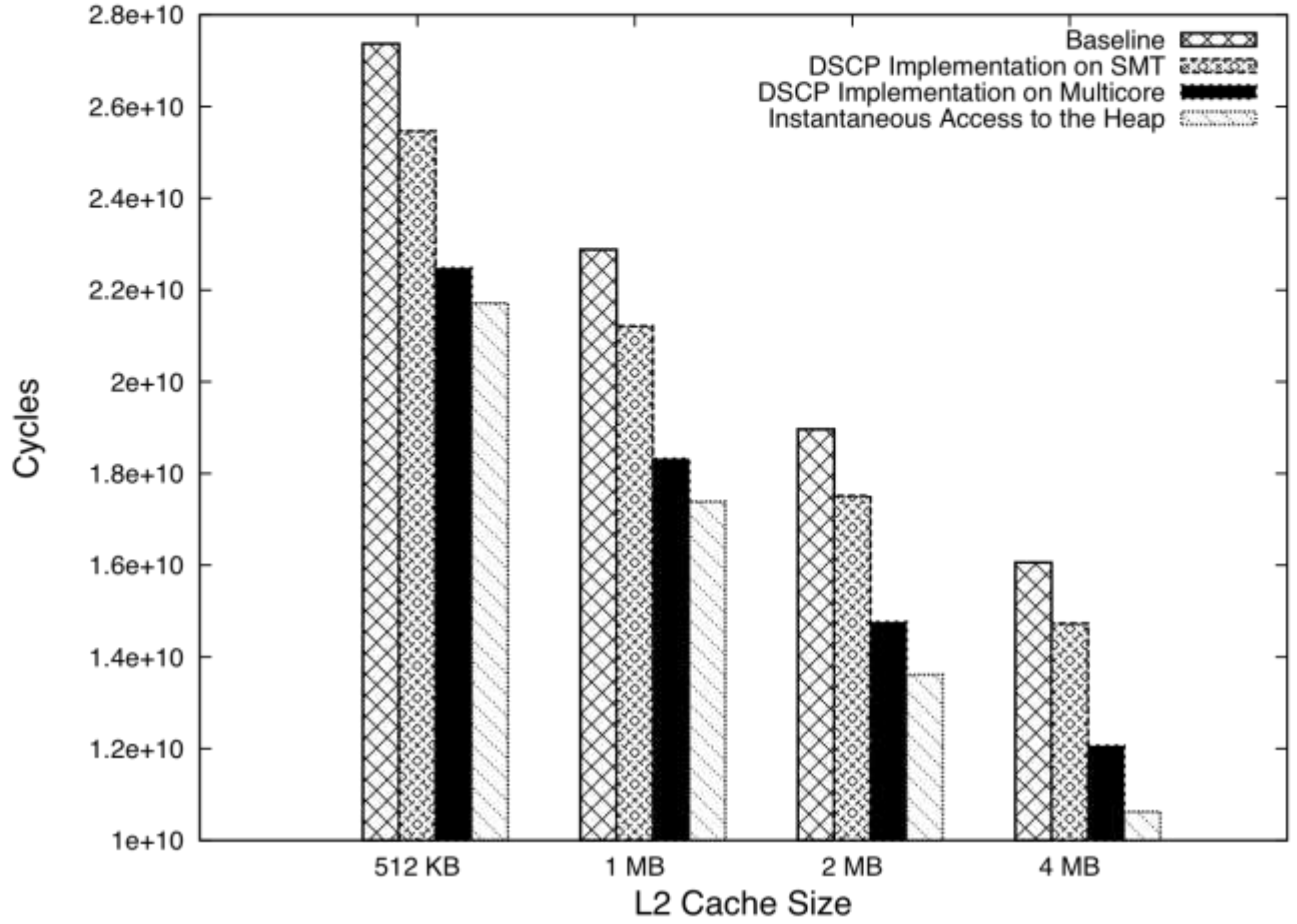
## Multicore Implementation

- DSCP is performed on a separate core
  - **Advantage**: no resource competition
  - **Disadvantage**: long communication latencies, because L1 cache is not shared
  - Hardware support needed to reduce cross-core communication latencies
    - Inter-core registers

## Speeding Up Communication: Inter-core Registers



## Performance with Full USA Benchmark



Warning: parallel computing trick with the base of the graph being 1e+10

## Results Summary

- 26% performance improvement on average for all simulated graphs with CMP
- Dense graphs are within 2% of maximum possible improvement that would be achieved if all data structure operations were eliminated
- For sparse graphs, significant room for further improvements
- 20% to 25% improvement on Full USA map depending on the L2 cache size used

## Wrap-Up

- Serial bottlenecks are real.  Amdahl is (and will always be) right.  Anyone who tells you otherwise is some combination of {stupid, corrupt, deluded}.
- We must redouble efforts to improve serial performance.
- There are opportunities to chip away at hard serial bottlenecks
  - Unfortunately, not as "sexy" as the bogus results that are grabbing attention
  - It's hard for real improvements to compete with a parallel fantasy world….

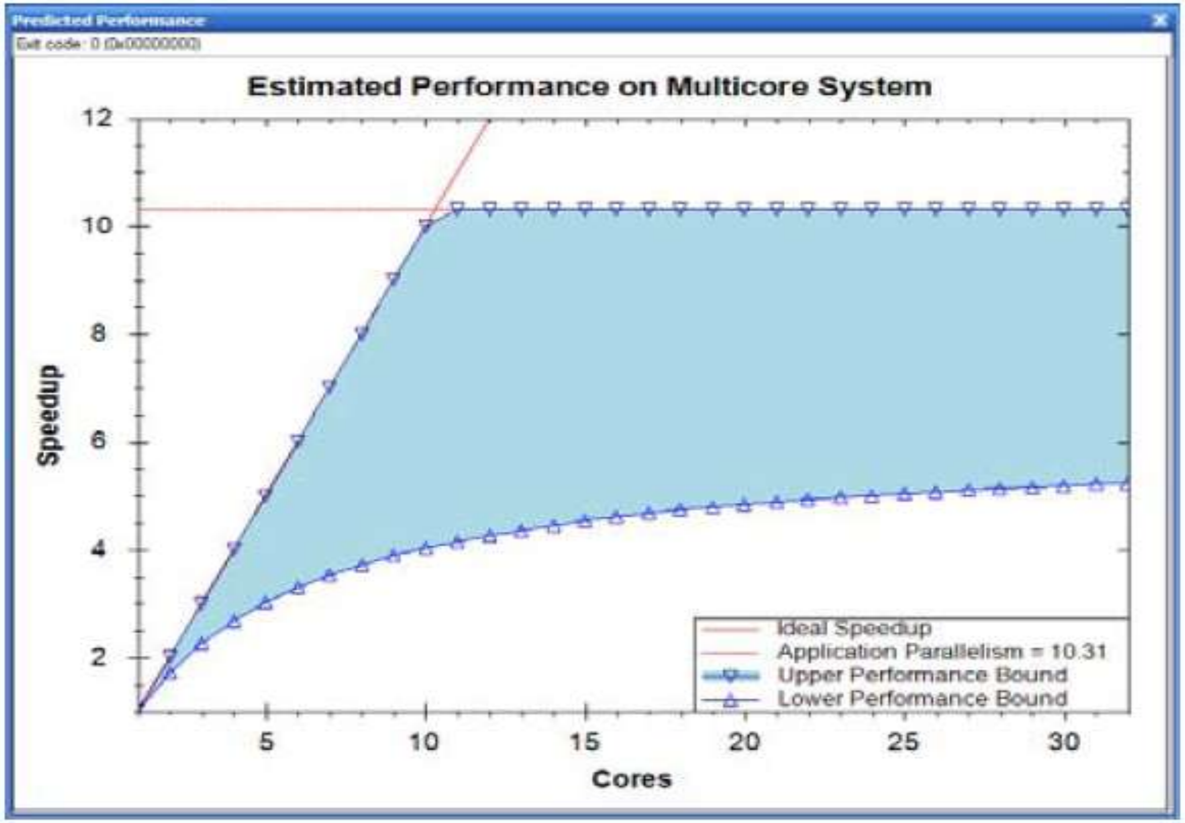
# Fooling the Masses

## Parallel Computing == Perpetual Motion

- David H. Bailey, 1991 Supercomputing Review, “Twelve Ways to Fool the Masses when Giving Results on Parallel Computers”
  - Numbers have been cooked since the earliest days of parallel computing
  - Some of the problems are honest mistakes.
  - Some of it shocking ignorance.
  - Some of it is outright fraud.
  - All of it is wrong, and bad for science.
- Bailey’s DAC 2009 paper was an update (he agreed to attend after I showed him a few EDA papers)

# But can't parallel speed things up?

## Yes, a little bit.



Charles Leiserson, DAC 2009. For quicksort, an upper bound of about 10x speedup (for a very large input file)

In practice, probably much less.

He agreed to come to DAC after I showed him some EDA papers.

# Rethinking Parallel

- We are faced with a performance problem; clock rates have ground to a halt.
- We have an ocean of transistors available to us, but can’t use them to increase serial ILP

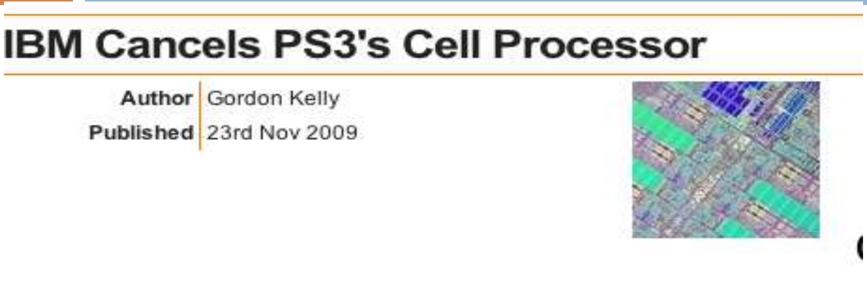
- The hypothesis: massive parallelism will be a useful way to put those transistors to work.
  - Complexity theory: the algorithm is the most important choice.
  - Work and Span laws: many of the best algorithms do not scale, and we have serial sections of applications.
  - Amdahl’s Law: the serial part dominates

## Are We Scientists?

- There are scam artists who try to trick people in investing in perpetual motion machines.

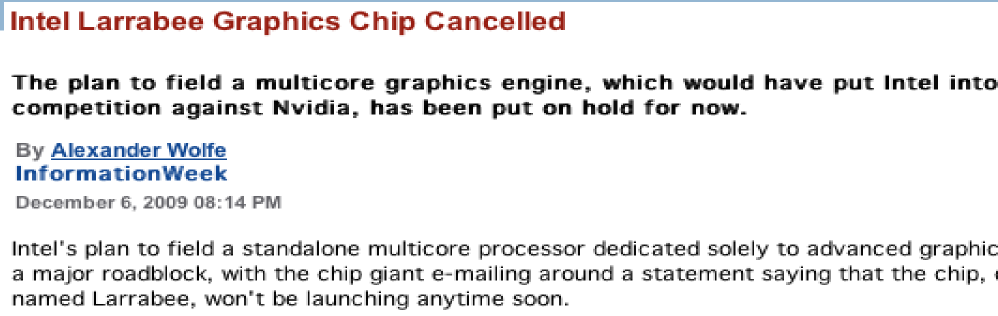
- Are we really any better in our community?
  - Many of the “experts” publishing in top ranked conferences, journals, books, are basing their work on things that are factually incorrect.
  - Shouldn’t we, as scientists, reject this?  And speak out against it?  Or is the funding available causing us to compromise our principles?
  - The lack of scientific rigor, and the tolerance of incorrect facts, reflects poorly on all of us.

# 5-Year Prediction



IBM Cancels PS3's Cell Processor

| | |
|---|---|
| Author | Gordon Kelly |
| Published | 23rd Nov 2009 |



Intel Larrabee Graphics Chip Cancelled

**The plan to field a multicore graphics engine, which would have put Intel into competition against Nvidia, has been put on hold for now.**

By Alexander Wolfe
InformationWeek
December 6, 2009 08:14 PM

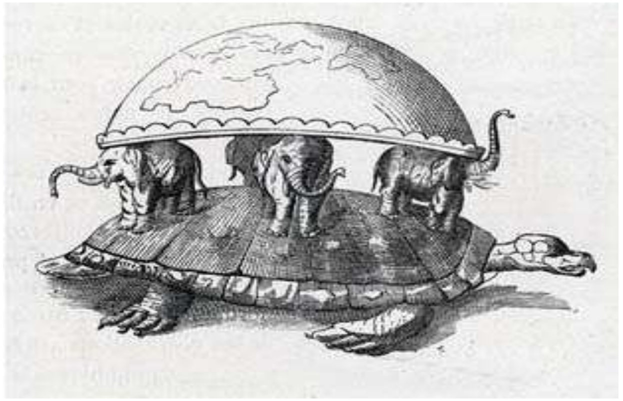
Intel's plan to field a standalone multicore processor dedicated solely to advanced graphic a major roadblock, with the chip giant e-mailing around a statement saying that the chip, named Larrabee, won't be launching anytime soon.

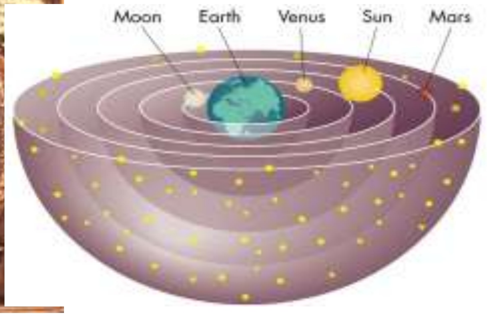
- Larrabee, Cell follow-ons will also be cancelled.
- Research money and best paper awards continue to go to those who know the least about algorithms.  It's advantageous for an idea to look good but not be practical: it means endless funding for an academic.
- Continued willful blindness to the basics of computer science
- More commercial failures, and stagnation of the industry. There's only so long that we can go without making any meaningful performance gain

## A Call to Action

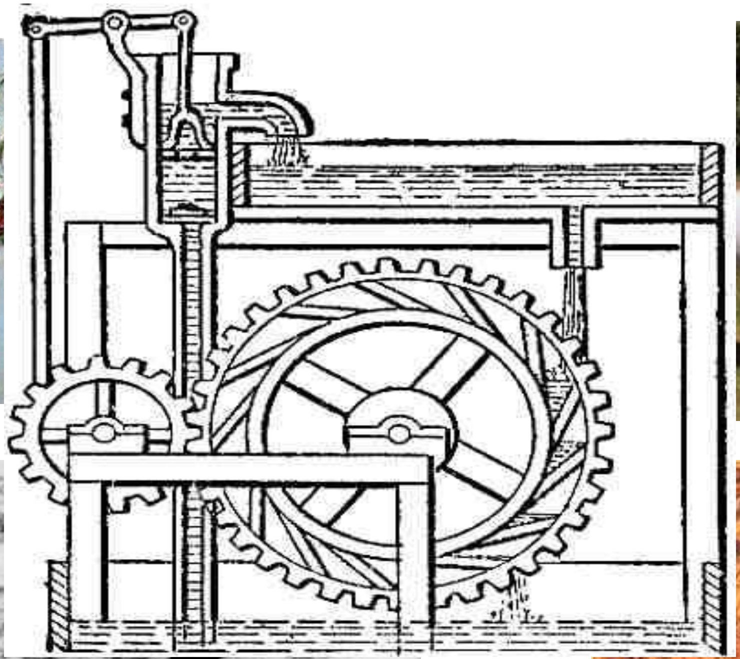
- We are faced with an ocean of transistors, and no way to run processing faster.
- Parallelism will take us only a very short distance.
- We, as a community, must cut through the hype, and pursue ideas other than those that have failed for 50+ years.

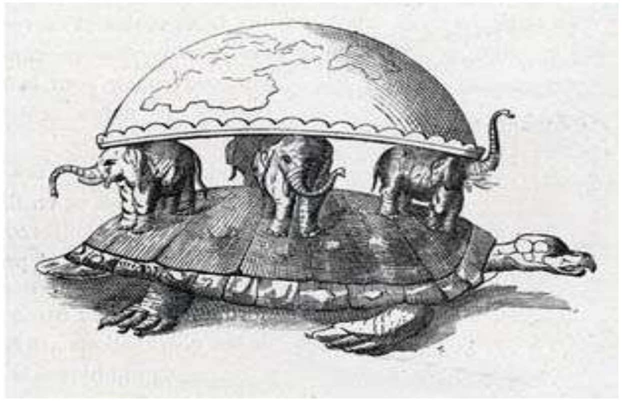
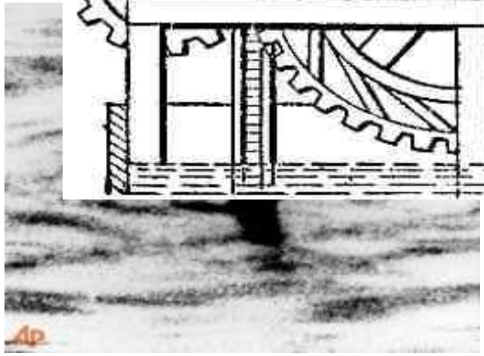
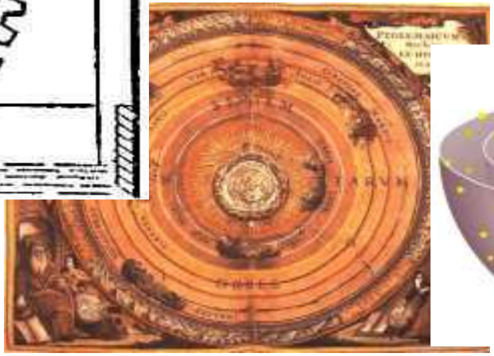
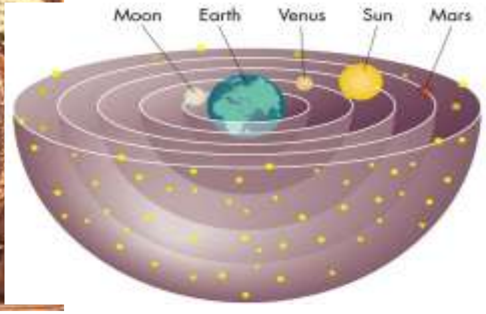
- Hypothesis: massive parallel computing will be a general-purpose way to increase performance, and to fill next-generation dies.



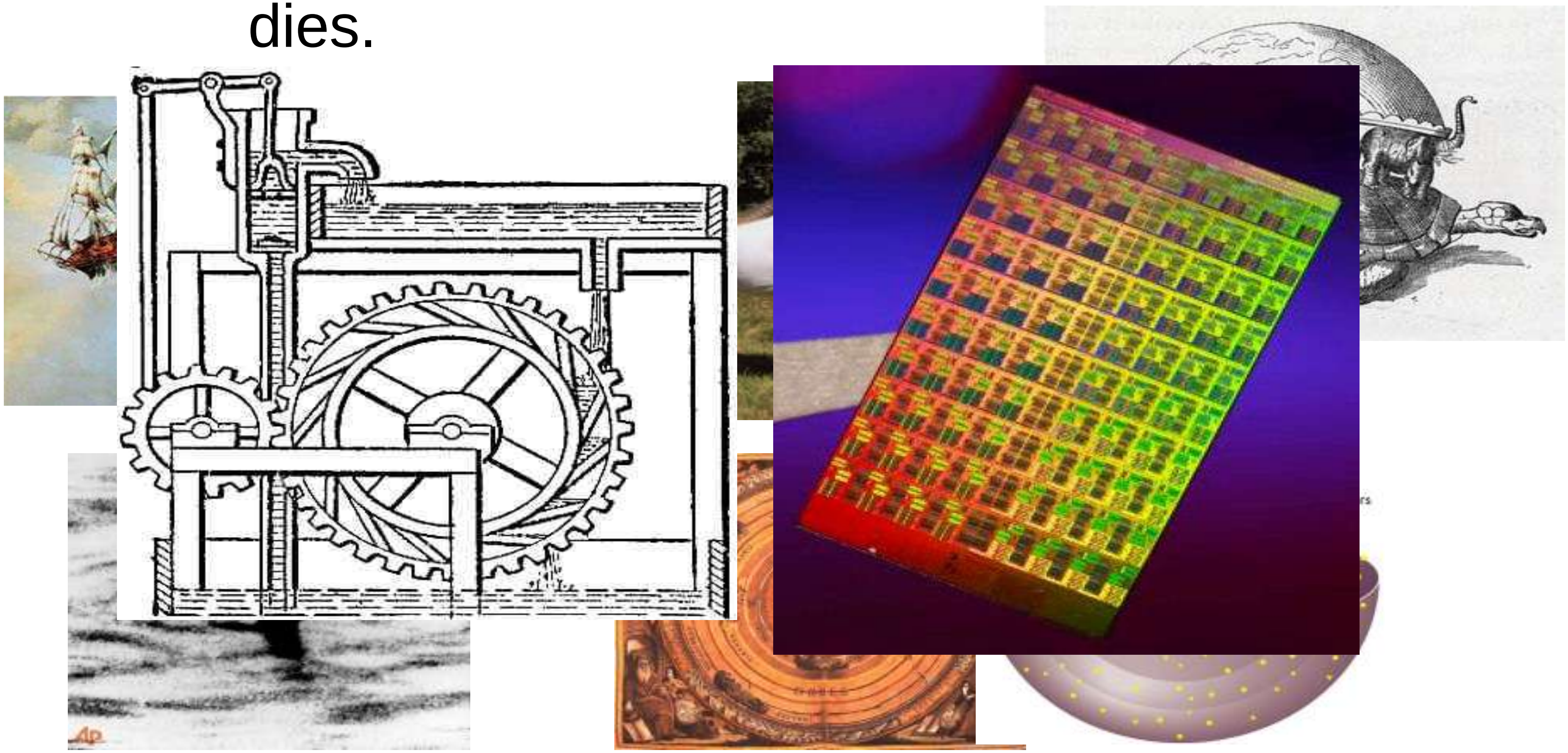


- Hypothesis: massive parallel computing will be a general-purpose way to increase performance, and to fill next-generation dies.



- Hypothesis: massive parallel computing will be a general-purpose way to increase performance, and to fill next-generation dies.



## Computing Task

- Compute the sum of the numbers from 1 to n

```
sum = 0;
for (i = 1; i <= n; ++i)
    sum = sum + i;
```

Great news! We can make this scale to thousands of processors; we break the computation into regions (map), and then combine (reduce)

# Gotcha

- The sum from 1 to n can be computed in a closed form: (n*(n+1))/2
- This has a serial bottleneck: the addition must occur before the multiply

$$\begin{tikzcd}
n/2 \arrow[dr] & \\
 & \text{multiply} \\
n+1 \arrow[ur] &
\end{tikzcd}$$

# Complexity Theory

- **An efficient algorithm is the MOST POWERFUL technology available for computing. Everything else pales in comparison.**
  - A must-read paper: Hartmanis & Stearns, “On the computational complexity of algorithms,” Trans. AMS, 1965

- Parallel computers have no magic ability; they do not defy mathematics. They do not enable time travel or perpetual motion.
- At best, parallel can provide a constant factor improvement in run time; for many problems, the most computationally efficient algorithm offers only modest parallel speedup (if anything)
- Big-O complexity ignores constants for a reason: the algorithm

# Big-O complexity

As the problem size increases, the work increases for the O(n) algorithm. No matter how much speedup we can get with parallel, the lower complexity algorithm wins for large values of n

Lower complexity wins on both time and total power. It also requires no specialized hardware or compiler support.