The Limits of Parallelism for Simulation

or,

Are there “Embarrassingly Serial” Problems?

John L. Gustafson, Ph.D.

Sr. Fellow, Chief Product Architect
Advanced Micro Devices
THESIS

- By 2018, we will have systems with a million servers, each doing a trillion ops/sec via highly parallel GPU accelerators.

- Simulation apps aren’t keeping up. “My algorithm doesn’t have enough parallelism!” So we get capacity instead of capability. But problems are often parallel even when our algorithms are not.

- Other than job parallelism, Single-Instruction Multiple Data (SIMD) parallelism is the oldest and easiest to program form of parallelism (for serial-trained programmers). And it’s energy-efficient.

- A new approach that improves answer quality also provides both ample SIMD parallelism and relief from the “memory wall”.
PROBLEMS SOME CLAIM ARE “EMBARRASSINGLY SERIAL”

- Having a baby: Yup. That’s one.
- Sorting: Not serial.
- Satisfiability: NP-hard? Brute force is parallel.
- Shortest-path: Not serial.
- Markov chains: Large matrix ops are parallel.
- Compiling Verilog: Serial… for synchronous chips.
- Fibonacci series, digits of π: Not serial! Compute \( n^{th} \) # directly.
- Place-and-route optimization: Looks parallelizable…
- Fiduccia-Mattheyses partitioning: Parallel approaches exist.
- Fluid pressure/temp in long pipeline: Serial… by current methods.
- 3-body problem for many time steps, and other small marching problems: Surprise: massively parallel if you want high accuracy!

Note: My thanks to Patrick Madden for several of these!
The simplest model that is still useful is

$$\text{Time}(N,P) = \frac{N}{P} + OH(N,P)$$  (Saeed Iqbal)

Where $N =$ parallel work that takes 1 second on 1 processor
$P =$ Number of processors,
$OH$ is all overheads, in seconds (communication, load imbalance, etc.)

Differentiate with respect to $P$ to find minimum (optimal $P$):

$$\frac{N}{P^2} + \frac{OH(N,P)}{P} = 0$$

$$P_{OP} = \sqrt[ \frac{OH(N,P)}{P} \bigg|_{P_{OP}} ]$$

(It’s usually a nonlinear equation)
$OH(N,P) = k \ln(P)$  (Typical for reduction operations)

$OH(N,P) = k \ln(P)$

$\frac{OH(N,P)}{P} = \frac{k}{P}$

$P_{OP} = \sqrt{\frac{N}{k \sqrt{P_{OP}}}}$

$P_{OP} = \frac{N}{k}$

Exercise for reader: Maximum speedup = $P_{OP} / (1+\ln(P_{OP}))$. $k$ drops out!

For a dot product on a trillion numbers, $k \approx 2000$, $P_{OP} \approx 500$ million. Maximum speedup will be about 20 million. The limits of parallelism for simulation are still a long way off.
SOME APPS THAT NEED EXASCALE LEVELS OF PARALLELISM

• Simulate all chemical activity in a living cell. (Reverse engineer life itself.)

• Model commercial aircraft behavior with uncompromising, dynamic CFD and structural analysis; optimize the design

• Find all activity controlled by a genome, including that of the “dark DNA” (noncoding for proteins)

• Model macroscopic material behavior with quantum-mechanical accuracy for each atom and bond

• Compute light emitted from every surface in a scene, given full physics (angle-dependent reflectance functions) at video rates
20 megawatts / 1 million servers = only 20 watts per server (including RAM). Data center power costs $1 / watt / year now.

2000 racks implies 500 servers/rack
POWER BUDGETS FORCE PARALLELISM

- If servers are 1 GHz, 1000 flops/clock still gets us to exascale (which is why you’ll need graphics coprocessors)
- Need 50 Gflops/watt by 2018… but now (2012), AMD is shipping discrete GPUs that are 16 Gflops/watt (32-bit).
- Heterogeneous parallelism is inherently power-efficient.
A SIMD accelerator approach gives up Control to reduce wattage per Tflops/s. Which can work, for applications that have a high ratio of operations to decision-making.
With great power comes great responsibility
—Uncle Ben

Yes, and also some really big heat sinks.
—John G.
## LOOKING FOR PLACES TO CUT ENERGY... 

<table>
<thead>
<tr>
<th>Operation</th>
<th>Approximate energy consumed (2011 data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit multiply-add</td>
<td>200 pJ</td>
</tr>
<tr>
<td>Read 64 bits from cache</td>
<td>800 pJ</td>
</tr>
<tr>
<td>Move 64 bits across chip</td>
<td>2000 pJ</td>
</tr>
<tr>
<td>Execute an instruction</td>
<td>7500 pJ</td>
</tr>
<tr>
<td>Read 64 bits from DRAM</td>
<td>12000 pJ</td>
</tr>
</tbody>
</table>

Notice that 12000 pJ @ 3 GHz = 36 watts!

So with every DRAM transfer, start asking:

*Is this trip necessary?*

*What is the best use we can make of DRAM data?*
IMPACT ON THE MEMORY MODEL

Automatic caches are very wasteful

• Hardware cache policies are speculative, designed to minimize *miss rates*, at the expense of low cache *utilization* (typically around 20%).

• Memory transfers will soon be half the power consumed by a computer, and computing is already power-constrained.

• We need programming environments that make memory placement visible and *explicit*, at least for programmers of low-level programmers.
HOW DID WE GET INTO THIS MESS?

• Getting 1% more on industry-standard serial benchmarks (like SPEC) at the cost of ~10% more wattage used to make good marketing sense.

• Fear of parallel processing has driven conventional CPUs to speculative architectures with low ops per watt.
ONE WAY WE GOT INTO THIS MESS: WHY COMPUTER LANGUAGES?

Words are not the only way to describe complex systems.

Language is about as serial as anything the human brain does!

For a billion cores, we will want *graphical* programming environments. Maybe 3D.
A BRIEF HISTORY OF SIMD

- 1971: The ILLIAC-IV (source of Amdahl’s law!) 64 processors
- 1979: ICL Distributed Array Processor (DAP), 4096 processors
- 1983: Goodyear MPP, 16384 processors
- 1985: Thinking Machines CM-1, 65536 processors
- 1990: The MasPar MP-1, 16384 processors

…And then SIMD seems to disappear in favor of Shared Program Multiple Data (SPMD) on commodity processors (hypercubes, then Linux clusters). But…
SIMD’S SUBTLE REVIVAL

- 1994: Sun, DEC, IBM, AMD, Intel, all add 2-way and 4-way SIMD instructions to their serial architectures. VIS, MMX, etc.
- Graphics processors emerge with SIMD parallelism in the hundreds of processors. By 1998, every game console uses SIMD.
- 2005: ClearSpeed makes a 96 GFLOPS, 50 watt accelerator board with 192-way SIMD.
- The SIMD instructions in conventional processors keep getting wider. Compilers don’t keep up.
- AMD, Nvidia integrate SIMD GPUs into the CPU chip as well as offering discrete boards
WHY IS SIMD INHERENTLY POWER-EFFICIENT?

- Because instruction issue takes about 30x as much power as a multiply-add! So SIMD can economize on control, if the application has enough data parallelism.
- Data parallelism leads to planned, 100% utilization of DRAM bandwidth
  - In conventional CPUs, only about 20% of a cache line is typically used!
  - And only about 20% of cache write-back data is ever accessed again
- The video game market makes it practical to design special memories for SIMD like GDDR3 that are power-efficient to deliver 100s of GB/s within PCIe form factor.
- SIMD doesn’t need full random access!
A TALE OF TWO SIMDS: ON-CHIP OR DISCRETE?

2012 On-Chip (APU)
- 128 to 384 SIMD cores (256-core APU consumes 17 watts)
- 10s of GFLOPS in double precision
- Easier programming due to tight integration with CPU, but...
- Pin-limited bandwidth to off-chip DRAM

2012 Discrete Graphics
- Up to 2048 SIMD cores, consumes about 200 watts
- 1 TFLOPS in double precision
- Hundreds of GB/s to GB of on-board memory, but...
- PCIe 3.0 limits bandwidth, latency of connection to CPU
DECIDE BASED ON DATA RE-USE, PARALLELISM

2012 On-Chip (APU)

- Beats CPU when data re-use is on the order of 10-100
- Best when data parallelism is on the order of 100s per thread
- Examples: FFTs, sparse linear algebra

2012 Discrete Graphics

- Beats CPU and APU when data re-use is on the order of 1000s
- Best when data parallelism is on the order of 1000s per thread
- Examples: Dense matrix operations, many-body dynamics
Time to move $N$ data to/from another node or an accelerator is $\sim$latency $+ N/B$ seconds.

Because local memory bandwidth is usually higher than $B$, acceleration might be lost in the communication time.

Estimate the breakeven point for the task (note: offloading is different from accelerating, where host continues working).
THE DISCRETE PARALLEL ACCELERATOR IDEA IS AS OLD AS SUPERCOMPUTING ITSELF

General-purpose computer
Runs OS, compilers, disk, printers, user interface

Attached vector processor accelerates certain applications, but not all

Even in 1977, HPC users faced issues of when it makes sense to move data to use floating-point intensive hardware
AN OLD WAY TO INCREASE SIMD PARALLELISM...

- Monte Carlo methods. Not the Metropolis kind with simulated annealing, just the sampling kind that reduce the dimensionality of problems with many degrees of freedom.
- Black-Scholes is a well-known example.
- High-quality random-number generator libraries exist for SIMD, so that part’s easy
- Even quantum chemistry can use Monte Carlo methods, but community is resistant (why?)
- Accumulations should be done in high precision to guard against rounding errors, but summands and total can be much lower precision than double, saving bandwidth!
OPTIMIZATION APPROACHES THAT USE MORE PARALLELISM

- In seeking an optimum, use cores to do *breadth*-first search
- Or, use parallel cores to find where optimum *cannot* be, to chip away at the search space (interval methods)
- Monte Carlo methods have huge ratio of parallel to serial work, and conquer problems with many degrees of freedom
NEW TOPIC: BETTER ARITHMETIC

- We need to change the rules of the game. Classic algorithms are long in the tooth.
- Worrying more about arithmetic *quality* has a big (positive) effect on parallel computing, energy use, and architectural issues.
- Analogy: Printers have gotten much better, but not with respect to speed. Technology goes to *quality*. 
We use the new technology for better prints, not to do low-quality prints in milliseconds.
USING 64-BIT USUALLY IS SPECULATION

Is it enough? Is it too much? We’re guessing.

At 1 exaflop/s ($10^{18}$), 15 decimals don’t last long.
IT’S UNLIKELY A CODE USES THE BEST PRECISION

- Too few bits gives unacceptable errors
- Too many bits wastes memory, bandwidth, **joules**
- This goes for integers as well as floating point
**ENDING FLOATING POINT HAZARDS**

- Parallelism changes answers. Better hardware support for 128-bit *accumulations* might allow lower precision (32-bit) to be used safely.
- Maybe it’s time to restore Numerical Analysis to the standard curriculum of computer scientists
- We can create tools to automate much of the work of a Numerical Analyst
- Using higher precision near the processor might even allow a return to 32-bit in DRAM… *or even less*?
WHAT ABOUT HARDWARE FOR...

REAL*2

- No, I’m not kidding. 16-bit floating point has 3 decimals of accuracy, $10^{10}$ dynamic range.
- We need more precision near the processor, and less near the main memory!
- Replace 8-byte floats (guesses) with two 2-byte floats used as interval bounds (rigorous).
EXAMPLE: LAPLACE’S EQUATION

- Magenta line specifies boundary condition.
- (Classic problem for relaxation methods, but multigrid has lowest arithmetic complexity.)
- Inside the unit square,

\[ \nabla^2 F = 0 \]
LAPLACE’S SOLVERS: WHICH IS BETTER?

64-bit floating point method *seems* to have converged. 15 decimals, some of them probably correct.

16-bit *interval* arithmetic provably bounds accuracy to 3 decimals, uses half the storage and bandwidth and energy.
A NEW APPROACH: UBOXES

- Compute with uboxes, sets of $n$-dimensional floats that represent interval errors that are one Unit in the Last Place (ULP) in each dimension.

- Combines the advantages of interval arithmetic’s rigor with point arithmetic’s slow growth in error bounds.

- I have built a small library of ubox methods for linear algebra, n-body methods, etc. that give massive amounts of data parallelism even for very few degrees of freedom.

- The more data parallelism used, the higher the quality of the result, where $\text{quality} = 1/(\text{total uncertainty in the answer})$.
EXAMPLE FOR 2 EQUATIONS, 2 UNKNOWNS

- Works like a “paint bucket” tool; start with one ubox that touches solution set, and use continuity to find neighbors until they fail the solution test.

- Answer “shapes” are usually complicated, but sets of uboxes track the shape easily.

- The higher the accuracy, the more SIMD things to do at once.

- Ultra-low precision works amazingly well. This example uses 8-bit floating point!
LINEAR SOLVERS BECOME CHALLENGING AGAIN...IF DONE RIGOROUSLY

- Even 2 equations in 2 unknowns involves computational solid geometry... intersecting 8 half-planes with exceptions for zero crossings
- Ultimate solution is the minimum “containment set.”
- Box around that solution leads to “wrapping problem”
- Gaussian elimination with interval values leads to VERY sloppy (usually useless) bounds!
N-BODY DYNAMICS WITH UBOXES
THE 3-BODY SIMULATION PROBLEM

- “Embarrassingly Serial”… only 18 variables, yet modeling behavior involves huge number of steps
- But each step produces an *irregular* containment set. Use all available cores to track. Suddenly it’s a Big Data problem!
- Far more ops per data point. Relieves “memory wall”
- Billions of cores usefully employed. Exploits parallel hardware.
- Provable bounds on the answer. Something most floating-point simulations haven’t ever had.
8-BIT RIGOROUS BOUNDS FOR 3-BODY CASE
“INTERVAL PHYSICS” APPROACHES FOR

- Radiation transfer (graphics, heat)
- Pin-connected truss structures (general structural analysis in the limit of fine structures)
- N-body dynamics (useful for provable CFD at very low pressure?)
- PDEs like Laplace where bounding the forcing function leads to bounds on the answer

This could be a “Golden Age” for algorithm research! We need all new methods.
SUMMARY

- The limits of parallelism for simulation are mainly algorithmic, not in the problems themselves.

- There is no shortage of apps with huge parallelism. Don’t sweat “efficiency”. Use fixed-time models when studying scaling, though, or you’ll get ridiculous predictions.

- If we really cherish every bit moved to and from main RAM, then we can get better arithmetic answers (provable bounds) and as a side effect, help the Memory Wall dilemma and… always have a use for massive parallelism. ■
DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2012 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, FirePro, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Microsoft, DirectX, Windows and Windows Server are trademarks of Microsoft Corporation. OpenCL is a trademark of Apple Inc. used with permission by Khronos. Other names are for informational purposes only and may be trademarks of their respective owners.