ERLANGEN REGIONAL COMPUTING CENTER [RRZE]

Thirteen modern ways to fool the masses with performance results on parallel computers

Georg Hager
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg

GridKa School, 2018-08-29

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Legal disclaimer

The information contained in this talk is for general guidance on matters of interest only. The application and impact of laws can vary widely based on the specific facts involved. Given the changing nature of laws, rules and regulations, and the inherent hazards of electronic communication, there may be delays, omissions or inaccuracies in information contained in this talk. Accordingly, the information in this talk is provided with the understanding that the authors and publishers are not herein engaged in rendering legal, accounting, tax, or other professional advice and services. As such, it should not be used as a substitute for consultation with professional accounting, tax, legal or other competent advisers. Before making any decision or taking any action, you should consult an HPC professional.

While we have made every attempt to ensure that the information contained in this talk has been obtained from reliable sources, we are not responsible for any errors or omissions, or for the results obtained from the use of this information. All information in this talk is provided "as is", with no guarantee of completeness, accuracy, timeliness or of the results obtained from the use of this information, and without warranty of any kind, express or implied, including, but not limited to warranties of performance, merchantability and fitness for a particular purpose. In no event will we, our related partnerships or corporations, or the partners, agents or employees thereof be liable to you or anyone else for any decision made or action taken in reliance on the information in this talk or for any consequential, special or similar damages, even if advised of the possibility of such damages.

Certain links in this talk connect to other websites maintained by third parties over whom we have no control. We make no representations as to the accuracy or any other aspect of information contained in other talks, websites, or papers.

And finally, we take no responsibility whatsoever for the consequences of you showing these slides around and getting **harassed**, **shouted at**, **beaten**, or **spanked** by your boss, your peers, your spouse, your kids, your mother, or anyone who might be offended because they don’t get the inherent irony. So there.

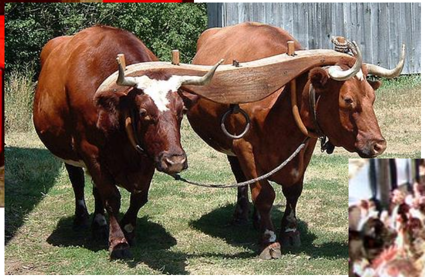
1991

David H. Bailey
Supercomputing Review, August 1991, p. 54-55
"*Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers*"

1. Quote only 32-bit performance results, not 64-bit results.
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
3. Quietly employ assembly code and other low-level language constructs.
4. Scale up the problem size with the number of processors, but omit any mention of this fact.
5. Quote performance results projected to a full system.
6. Compare your results against scalar, unoptimized code on Crays.
7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
9. Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

What supercomputing was like in 1991

If you were plowing a field, which would you rather use?



Two strong oxen
or 1024 chickens?

(Attributed to
Seymour Cray)

What supercomputing was like in 1991

Strong I/O facilities

SIMD/MIMD parallelism

32-bit vs. 64-bit FP arithmetic

Vectorization
(the real one, not the SSE/AVX c**p)

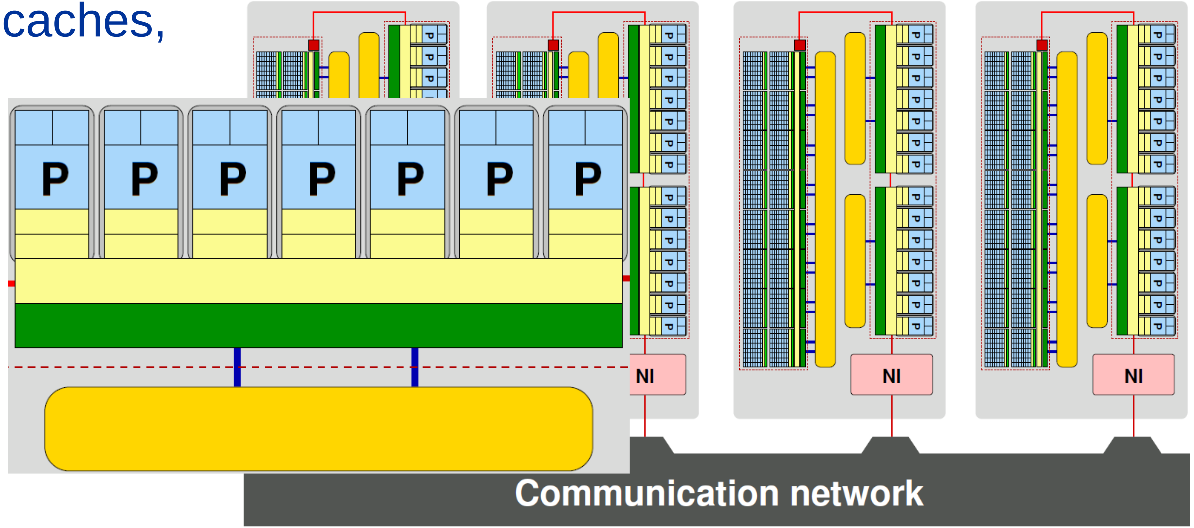
No parallelization standards

System-specific optimizations

Today we have…

Multicore processors

with shared/separate caches,
shared data paths



Hybrid, hierarchical systems

with multi-socket, multi-core, ccNUMA, accelerators, heterogeneous networks

Today we have…

Ants all over the place

- GPUs, ~~Xeon Phi~~, ...

Today we have…

Commodity everywhere

x86-type processors, cost-effective interconnects, GPUs, GNU/Linux

The landscape of High Performance Computing and the way we think about HPC has changed over the last 25 years, and we need an update!

Still, many of Bailey’s points are valid without change

<irony>

Stunt 1

Report scalability, not absolute performance or time to solution.

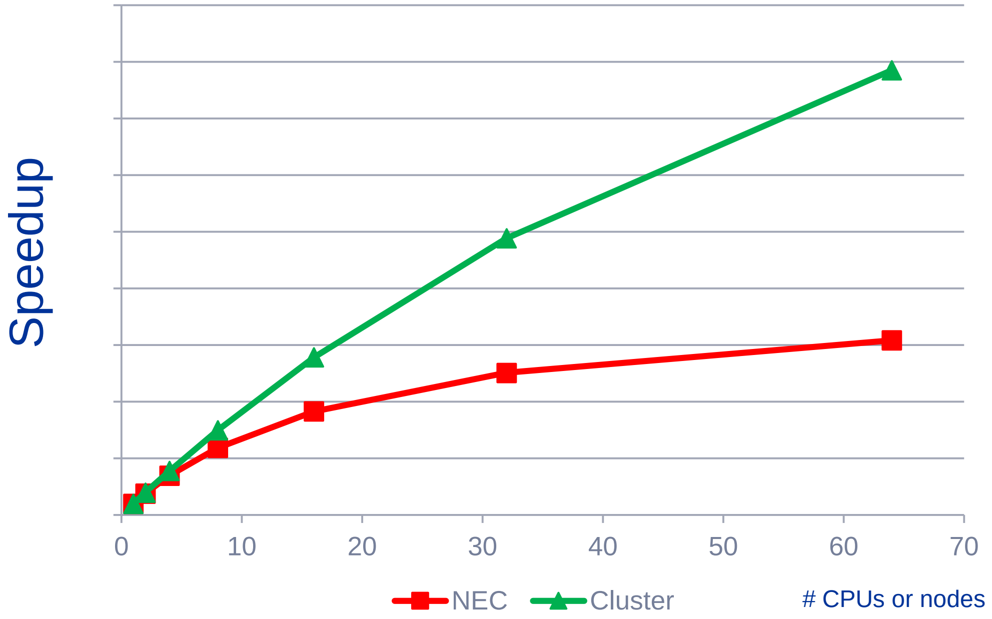
Speedup: $$S(N) = \frac{\frac{\text{work}}{\text{time}} \text{with } N \text{ workers}}{\frac{\text{work}}{\text{time}} \text{with 1 worker}}$$

"Good" scalability $\leftrightarrow S(N) \approx N$

Consequence: Makes your slow but scalable system look better

Stunt 1: Scalability vs. performance

And… instant success!



Stunt 2

Slow down code execution.

Strong scaling, "Non-execution" overhead $c(N)$

$$S(N) = \frac{1}{s + \frac{1-s}{N} + \boxed{c(N)}}$$

Slow down execution by a factor of $\mu > 1$:

$$S(N) = \frac{\mu}{\mu\left(s + \frac{1-s}{N}\right) + c(N)} = \frac{1}{s + \frac{1-s}{N} + c(N)/\mu}$$

I.e., if there is overhead, the slow code/machine scales better

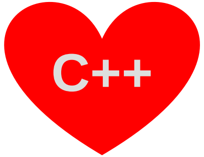
Stunt 2: Slow computing

Corollaries:

~~−O3~~

−O0

1. Do not use aggressive compiler optimization or manual tuning.
   Reason: reproducibility of results!
2. Use C++, Java, Perl, Python, Lua, or MS-Basic for hot spots.
   Reason: maintainability and flexibility!



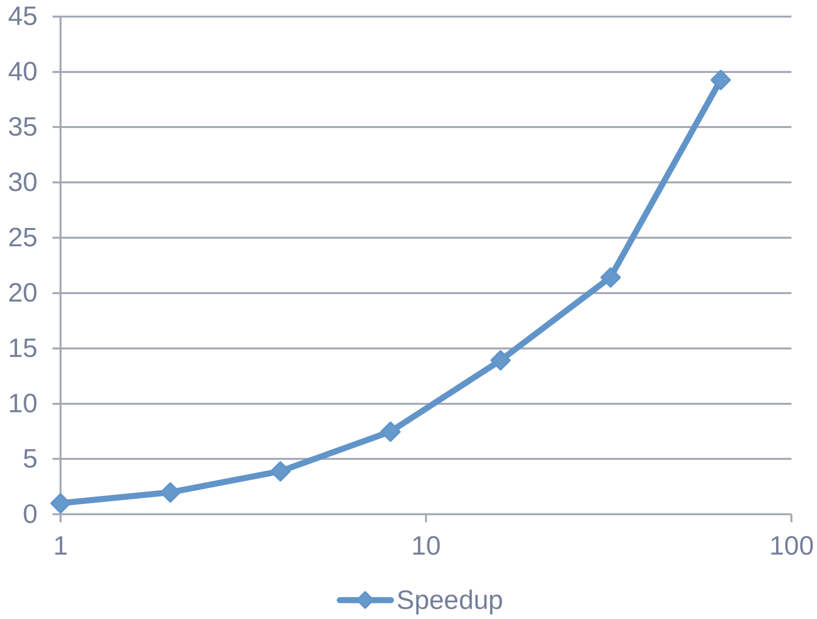
3. Scalability is still bad?
   → Parallelize some short loops with OpenMP.
   Reason: hybrid machines need hybrid code, don't they?

Time to solution? → "My code scales on Exascale systems.
Hold my beer…"

Stunt 3 (The power of obfuscation, part I)

If scalability doesn't look good enough, use a logarithmic scale to drive your point home.

1. Linear plot: bad scaling, strange things at N=32
2. Log-log plot: better scaling, but still the N=32 problem
3. Log-linear plot: N=32 problem gone
4. … and remove the ideal scaling line to make it perfect!



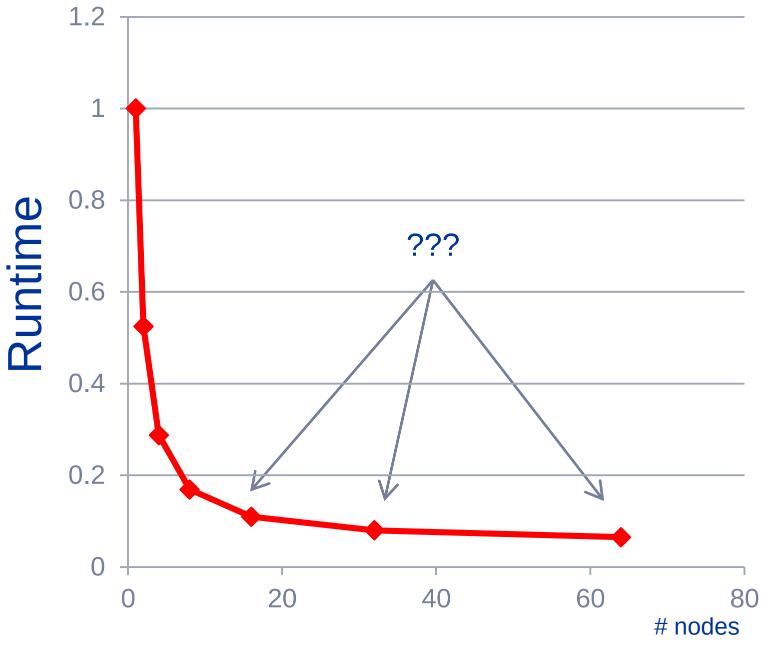
Stunt 4 (The power of obfuscation, part II)

Instead of performance, plot absolute runtime vs. CPU count

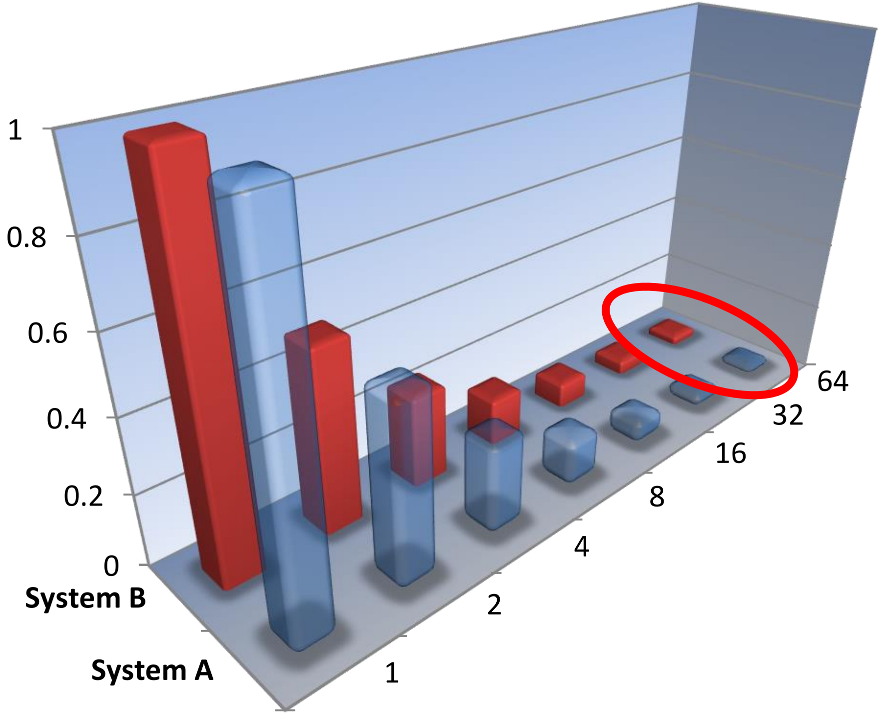
Very, very popular indeed!

Nobody will be able to tell whether your code actually scales

Caveat: Make sure to use linear scales!



Stunt 4 (but general): Eye candy can't hurt



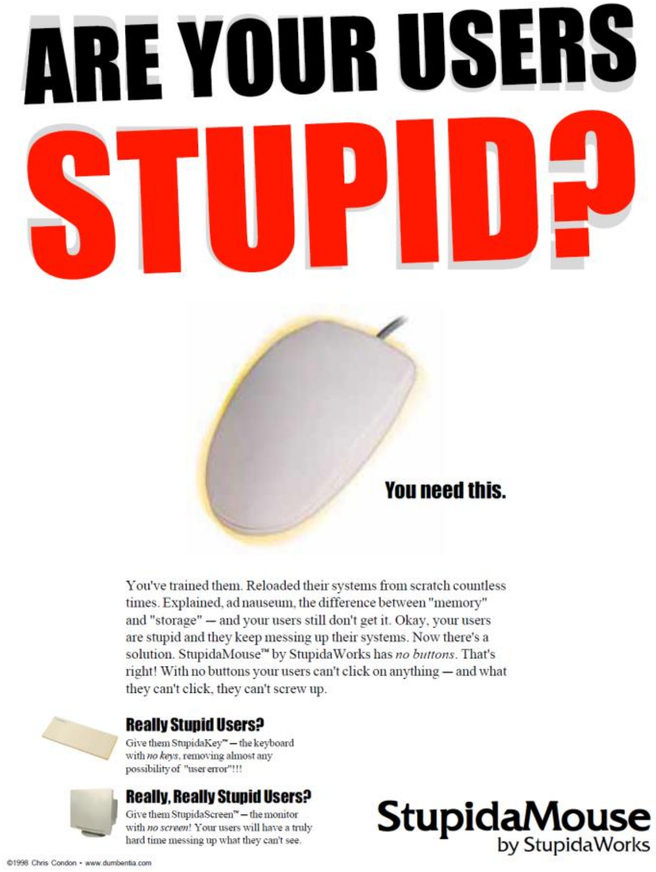
Stunt 5

Play mysterious.

Goal: Boost your citation count without giving away your dirty secrets

Idea: They can’t question what they can’t reproduce

Method: Make it hard. Really hard.



Stunt 5: Play mysterious.

Test case	problem size	# Iterations	Runtime [s]	Performance
car	see page 456	500	2.34443521	1.02x competition
plane	300^3	sufficient	3.14159	doesn't crash
train	Putin's ego	roughly 112	0.11991	0.64 cache misses per pJ
chicken	0.03 bu	whatever	42.0	1 egg/day

Stunt 5: Play mysterious.

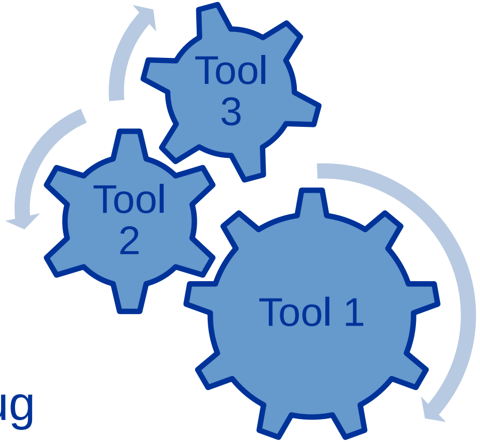
"For benchmarking we used an Intel Xeon under Ubuntu 16.04 LTS running Linux 4.4.0-131-generic #157-Ubuntu."

"Building the software requires gcc 2.95.2, MS Brainfuck 0.45pre, and Glasgow Haskell (< v7.2.1 but > v7.6.2) under CP/M 3.0"

"We are, on average over all test cases, 34% faster than the median of all competing frameworks."

"Our code is available for download at `http://goodstuffxxx.ru/koalemos`"

Stunt 6



Worship the god of automation.

Computers are for automating tasks. Why not automate the process of performance analysis?

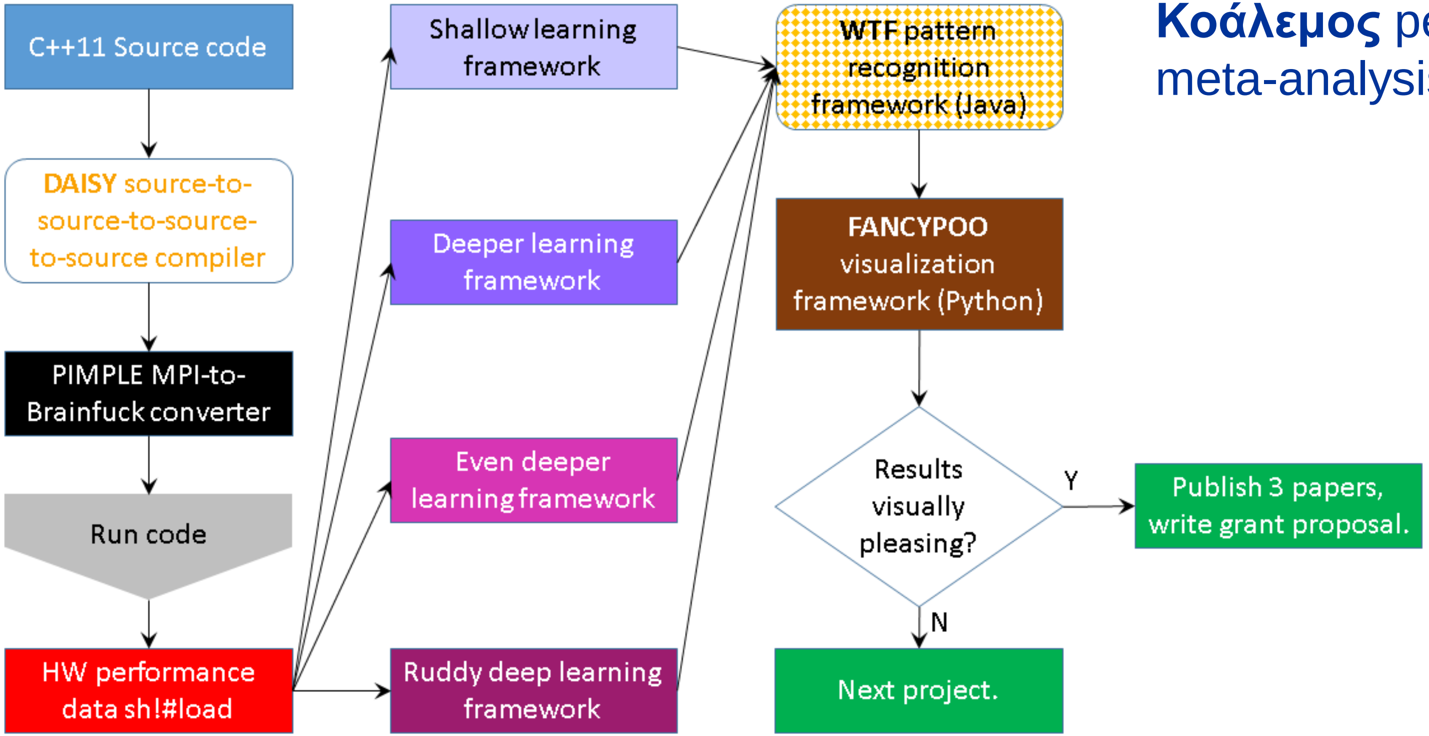
Automate everything. Use as many tools as possible and plug them together.

Use machine learning. Always. Throw in some big data for good measure.

Use at least three different languages (may be automatic).

Give the whole thing a catchy name.

Stunt 6: The power of automation



Κοάλεμος performance meta-analysis framework

Stunt 7

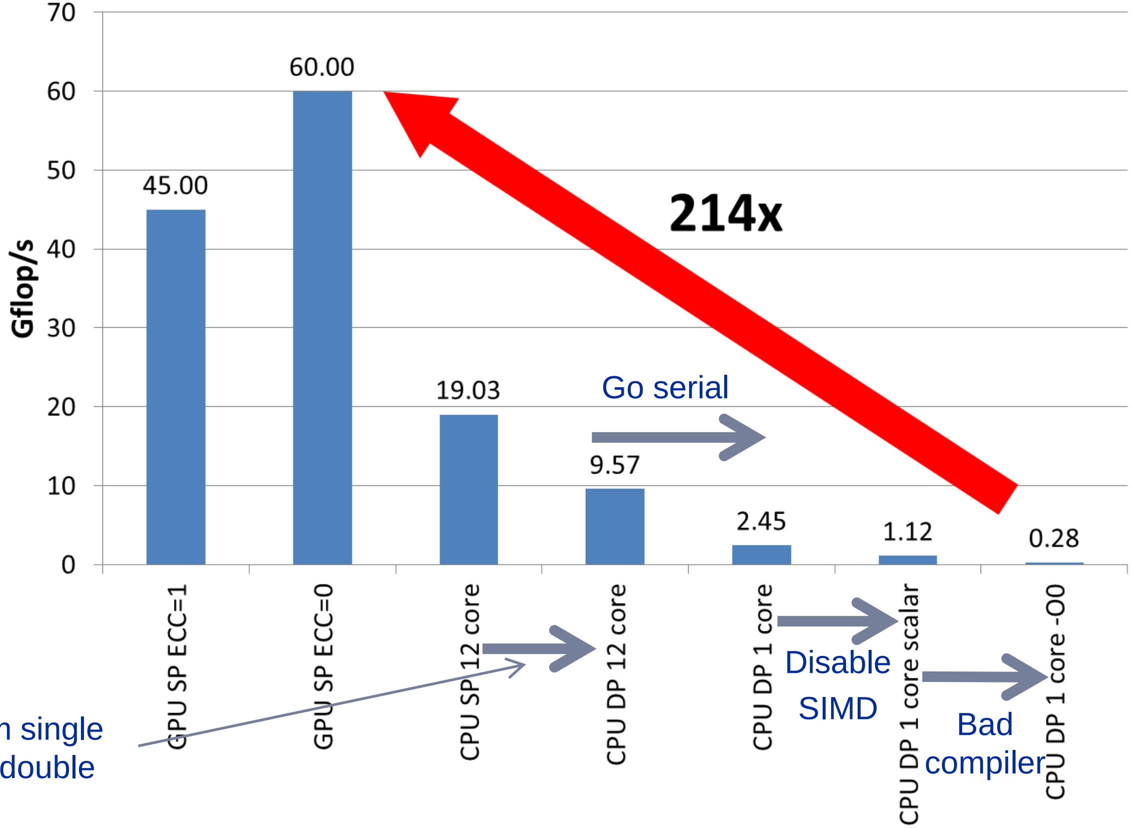
Emphasize the quality of your shiny accelerator code by comparing it with scalar, unoptimized code on a single core of an old standard CPU. *And use GCC 2.7.2*.

Besides, the compiler will do what's necessary on the CPU!

Corollary: Use single precision on the GPU but double precision on the CPU.

Stunt 7: Fabricating a usefully slow CPU baseline

Dense matrix-vector multiplication (N=4500), Nvidia Tesla C2050 vs. Intel dual Westmere

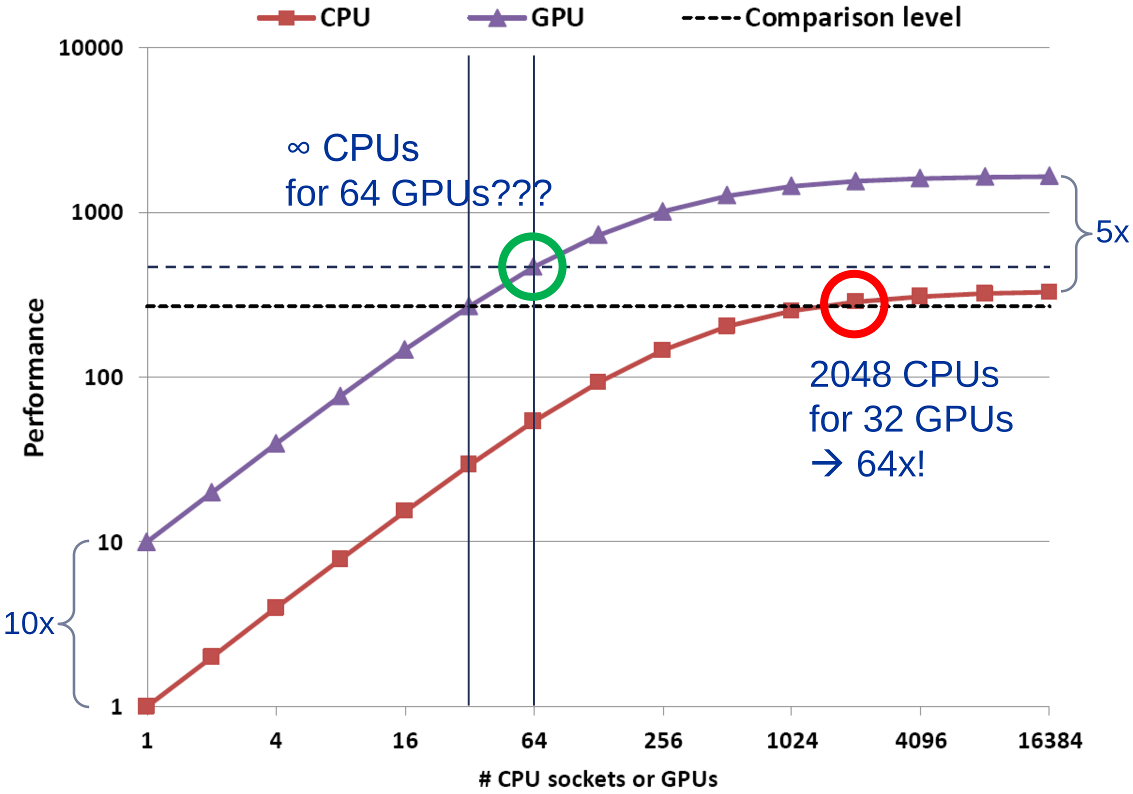


Change from single precision to double precision

Stunt 7a

If you ask the right questions, accelerators at scale give you arbitraty speedup!

Speedup = (How many CPUs do we need to outcompute N GPUs?)/N



Stunt 8

Quote GFlops, MIps, or any other ~~irrelevant~~ interesting metric instead of (inverse) time to solution.

Floptimizaton:

```
for(i=0; i<N; ++i)
  for(j=0; j<N; ++j)
    b[i][j] = 0.25*(a[i-1][j]+a[i+1][j]+a[i][j-1]+a[i][j+1]);
```

```
for(i=0; i<N; ++i)
  for(j=0; j<N; ++j)
    b[i][j] = 0.25*a[i-1][j]+0.25*a[i+1][j]
              +0.25*a[i][j-1]+0.25*a[i][j+1];
```

Stunt 8: Redefine “performance” appropriately

“Full resource utilization! Yay!”

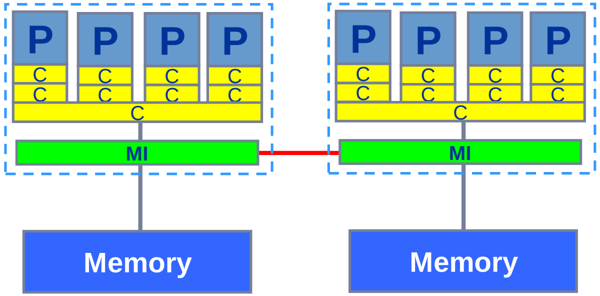
“Our new algorithm shows a factor of 5 fewer cache misses compared to the baseline.”

“Lower Watts/Flop!”

“24% less coding person-hours per MPI call!”

Stunt 9

Ignore affinity and topology issues. Real scientists are not bothered by such details.



SMT

OpenMP overhead

Shared cache shortage/re-use

OS buffer cache

Bandwidth contention

Intra-node MPI

ccNUMA page placement

Stunt 10 (The power of obfuscation, part III)

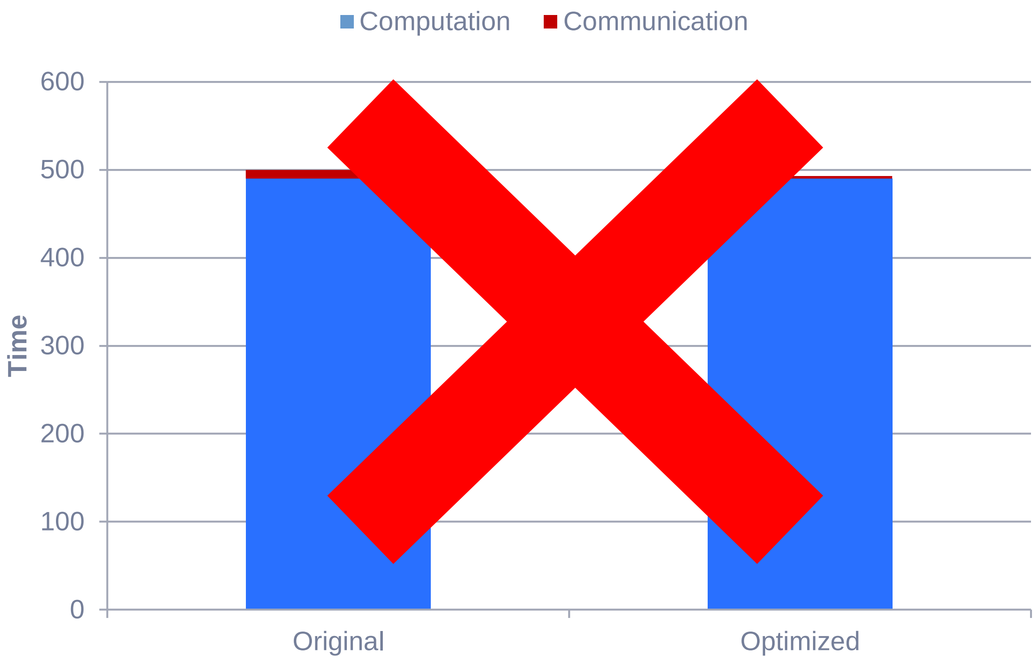
Always emphasize the “interesting” part of your work.

Ever thought about having
a diet coke with your bucket
of chicken wings?

+

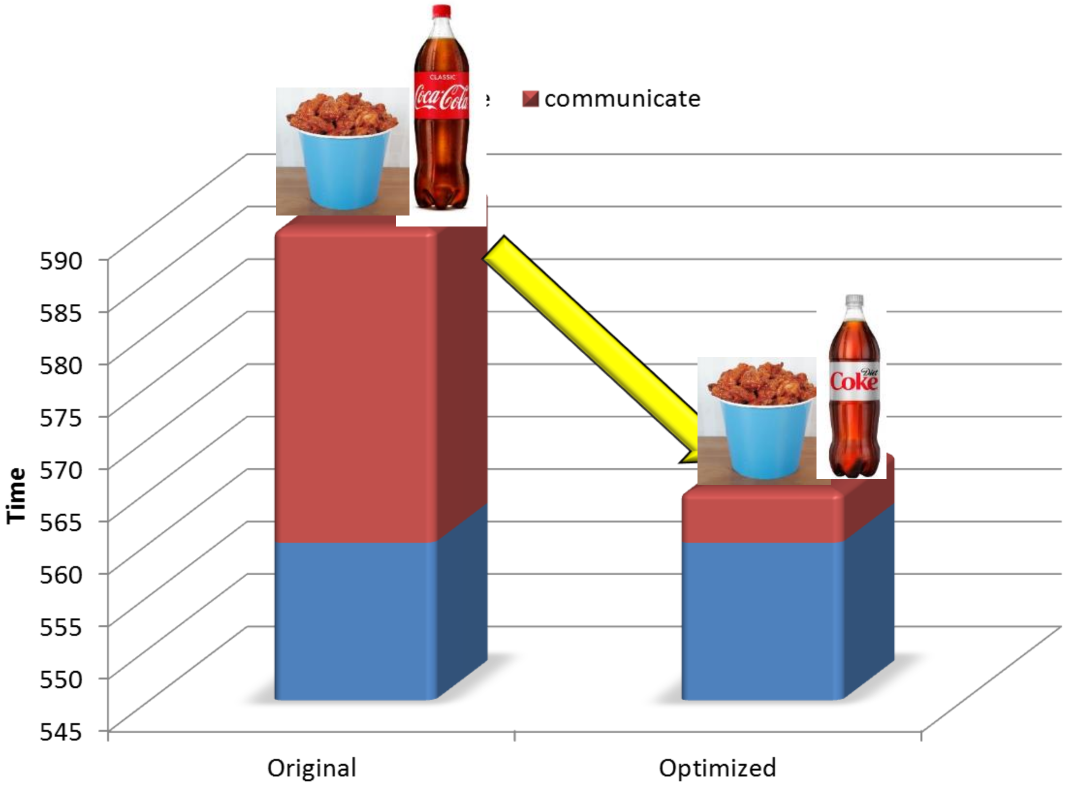
Stunt 10: Diet Coke

"Fig. 3 demonstrates the benefit of our new communication scheme, which reduces overall communication volume by 70% (cf. Tab. 2)"

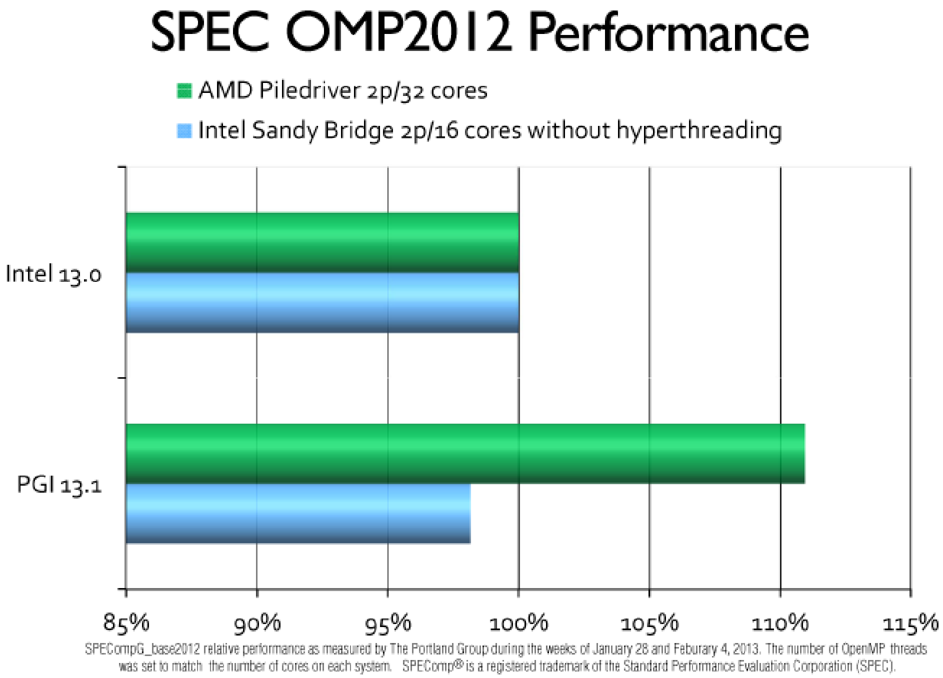


Stunt 10: Diet Coke

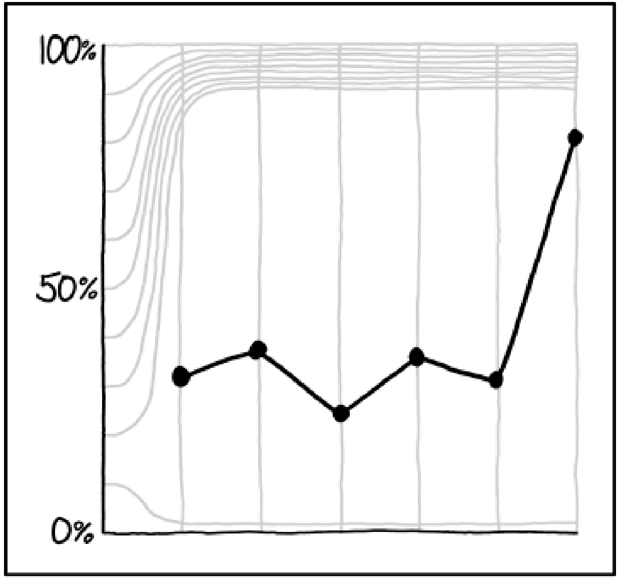
If they can’t see it,
zoom in a little!



Stunt 10: Diet Coke



http://www.pgroup.com/images/charts/spec_omp2012_chart_big.png



PEOPLE HAVE WISED UP TO THE "CAREFULLY CHOSEN Y-AXIS RANGE" TRICK, SO WE MISLEADING GRAPH MAKERS HAVE HAD TO GET CREATIVE.

https://xkcd.com/2023/

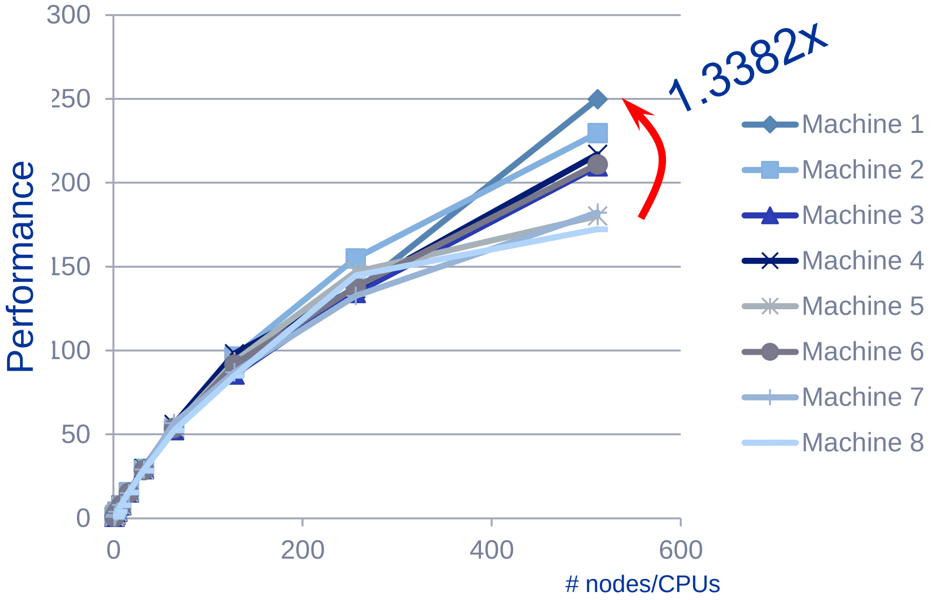
Stunt 11 (The power of obfuscation, part IV)

Show data. Plenty. And then some.

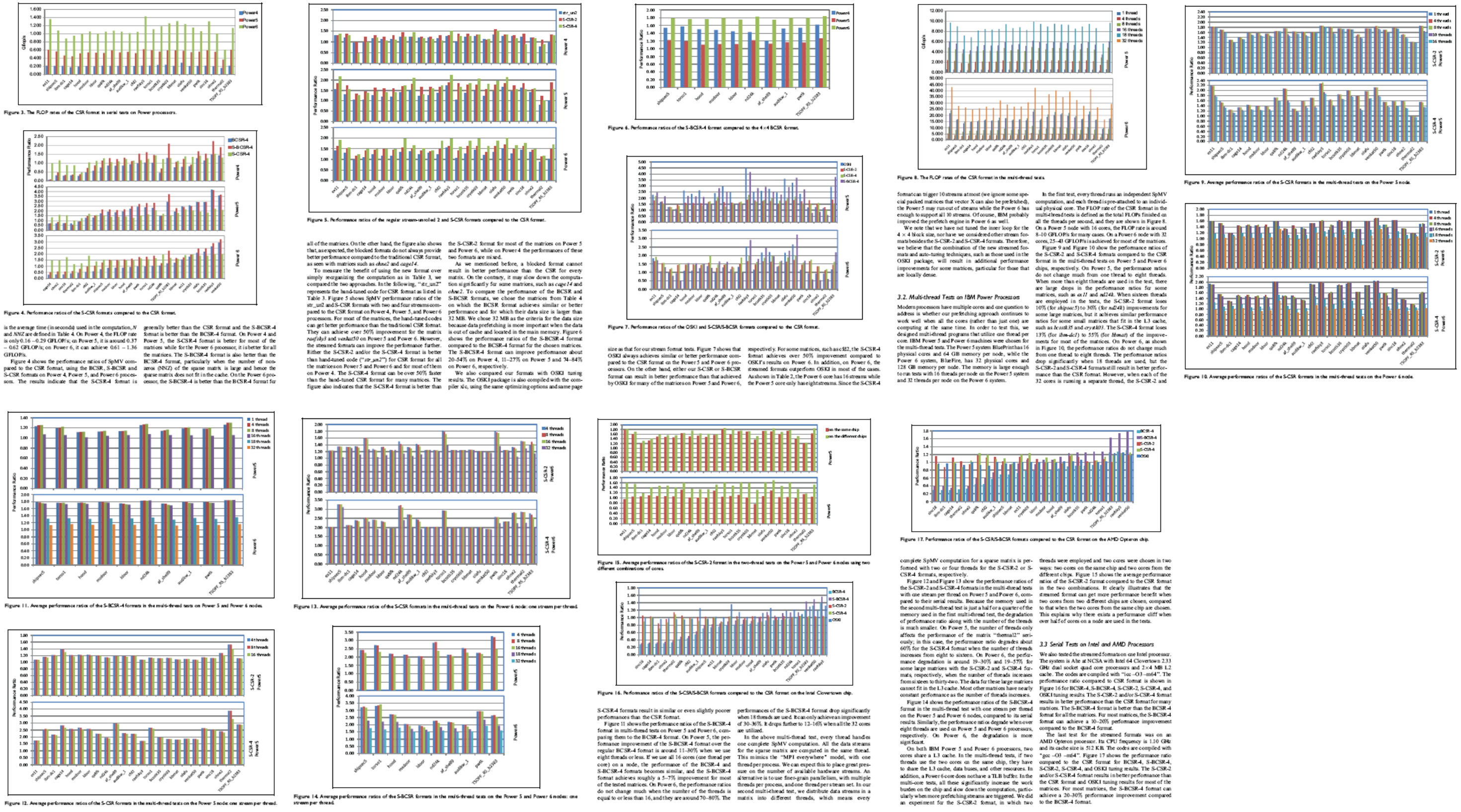
Make people see the breathtaking complexity of what you did. Show at least 8 graphs per plot, all in bright pastel colors, with different symbols.

Insight?

“It’s complicated!”



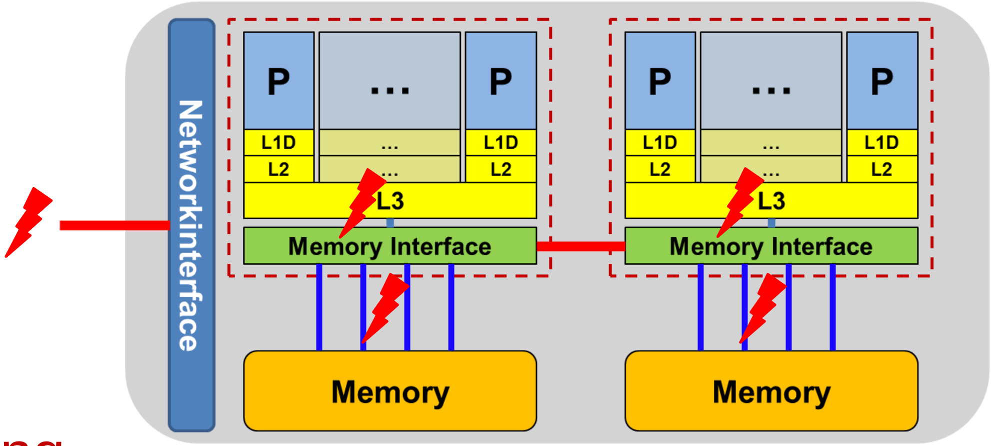
Stunt 11: Show data. Plenty. And then some.



Stunt 12

If they get you cornered, blame it all on "contention".

They will understand and nod knowingly.



Corollary: Depending on the audience, bad prefetching efficiency may work just as fine.

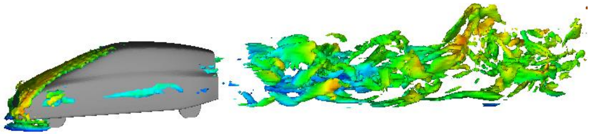
Stunt 12: Blame some *very* technical issue

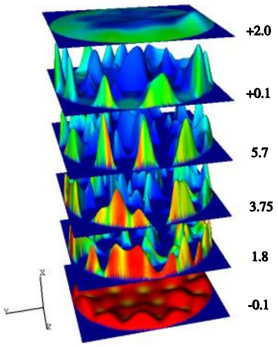
"Technical-detail-not-under-my-control":

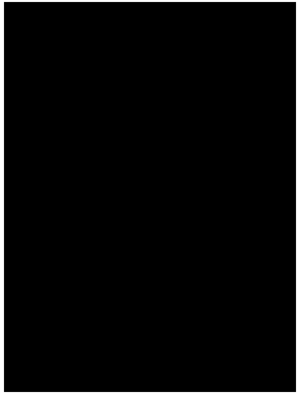
- Stupid compilers: "*Our version of the code shows slightly worse single-thread performance, which is presumably due to the limited optimization capabilities of the compiler.*"
- Out-of-order execution (or lack thereof): "*Processor A shows better performance than processor B possibly because of A's superior out-of-order processing capabilities.*"
- L1 instruction cache misses: "*As shown in Table 1, our optimized code version B is faster because it has 20% fewer L1 instruction cache misses than version A.*"
- TLB misses: *"Performance shows a gradual breakdown with growing problem size. This may be caused by excessive penalties due to TLB misses."*
- Bad prefetching: "*Performance does not scale beyond four cores on a socket. We attribute this to problems with the prefetching hardware.*"
- Bank conflicts: "*Processor X has only* [sic!] *eight cache banks, which may explain the large fluctuations in performance vs. problem size.*"
- Transient network errors: "*In contrast to other high-performance networks such as Cray's Gemini, InfiniBand does not have link-level error detection, which impacts the scalability of our highly parallel code.*"
- OS jitter: "*Beyond eight nodes our implementation essentially stops scaling. Since the cluster runs vanilla* [insert your dearly hated distro here] *Linux OS images, operating system noise ("OS jitter") is the likely cause for this failure.*"

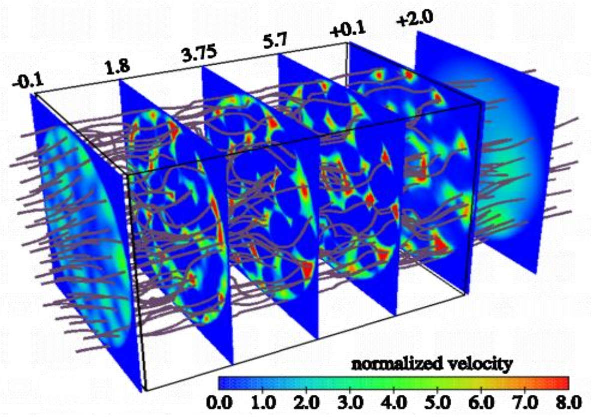
Stunt 13

If all else fails, show pretty pictures and animated videos, and don’t talk about performance.









</irony>

THANK YOU!

Read more at:

http://tiny.cc/foolingthemasses

(Stunt numbers may be different)

Ideas? Observations? → mailto:georg.hager@fau.de