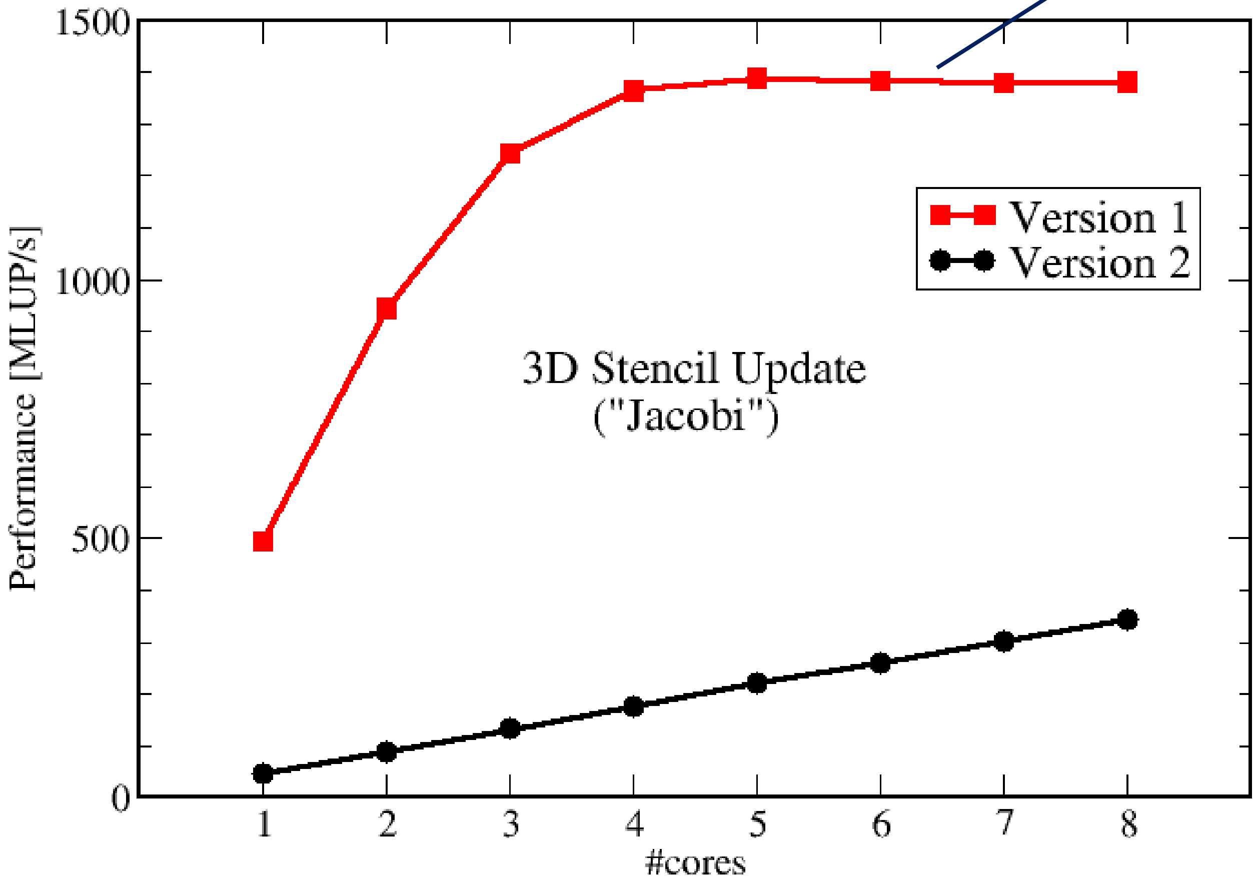
More Science per Joule: Bottleneck Computing

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

PPAM 2013
September 9, 2013
Warsaw, Poland

Motivation (1): Scalability rulez!

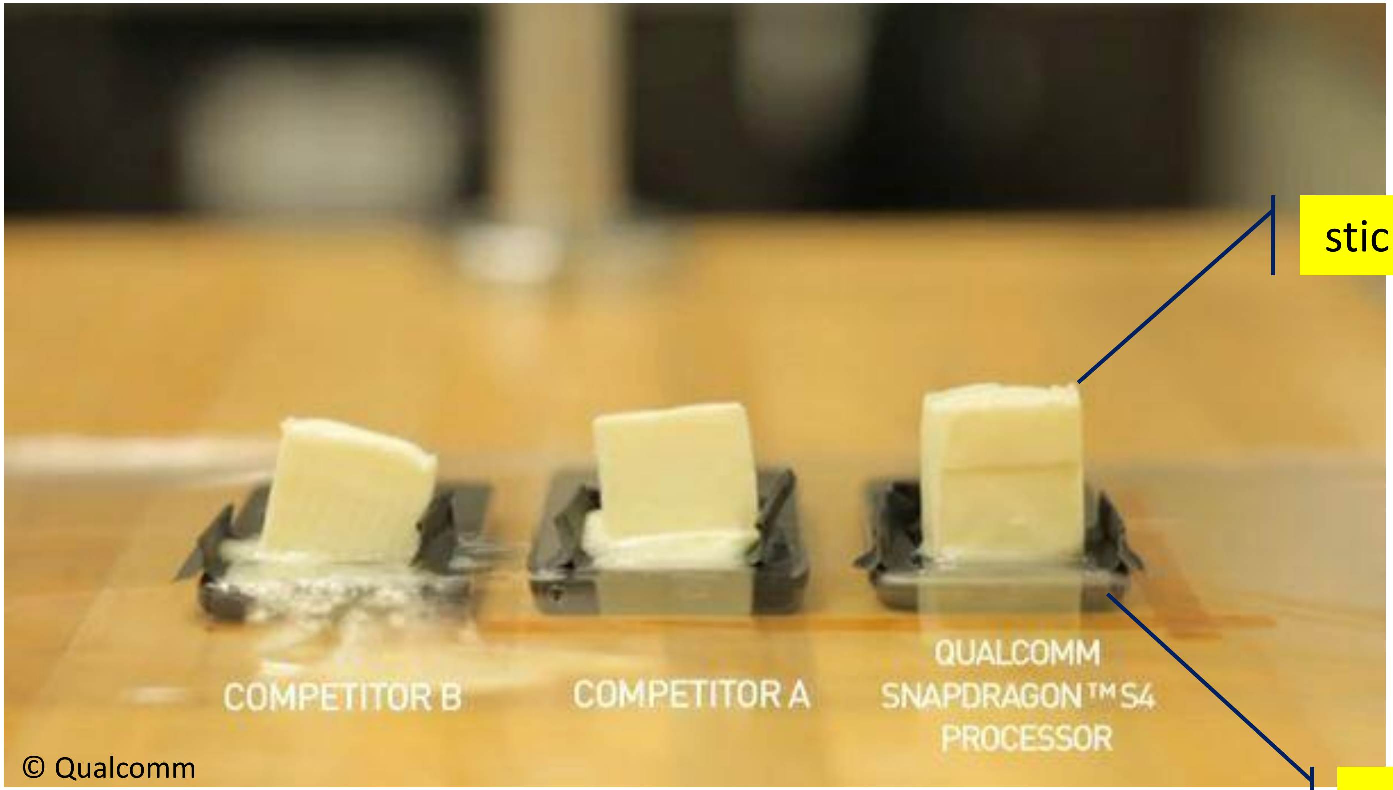
… or does it not?



More "science per day"

Motivation (2): What about power/energy?

… at least it's good for some cool [sic!] propaganda:



© Qualcomm

stick of butter

smartphone

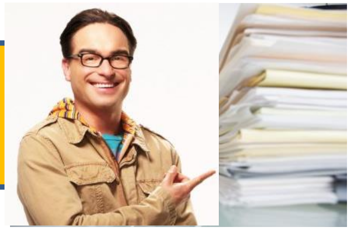
Points of view: Nerds and naggers

Scientist (“nerd”)

CPU time allocation

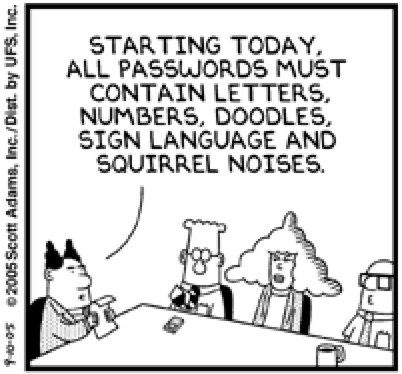
Project runtime

Science



Metric: Papers/CPUh

Computing Center (“naggers”)



STARTING TODAY, ALL PASSWORDS MUST CONTAIN LETTERS, NUMBERS, DOODLES, SIGN LANGUAGE AND SQUIRREL NOISES.

9-10-05 ©2005 Scott Adams, Inc./Dist. by UFS, Inc.

Hardware & maintenance cost

Machine lifetime

Energy cost

Power cap

Science

Next machine

Metric: ?

Line of thought

1. High Performance Computing == Computing at a bottleneck

2. There is code optimization potential in almost every application on every computer in this world

3. Making an application run faster by code optimization will reduce the energy spent on solving a problem (“code race to idle”)

4. Making an application run faster by playing with the clock speed may or may not save energy

5. Leaving part of the machine idle may reduce energy consumption without compromising performance

6. Maximum performance and optimized energy consumption are sometimes contradictory

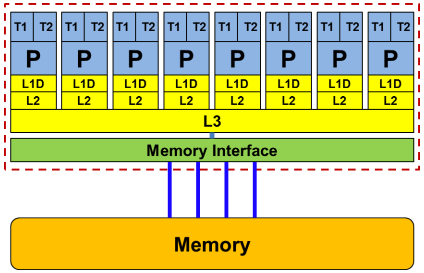
Setting the Stage (I): Performance Bottlenecks

Roofline Model

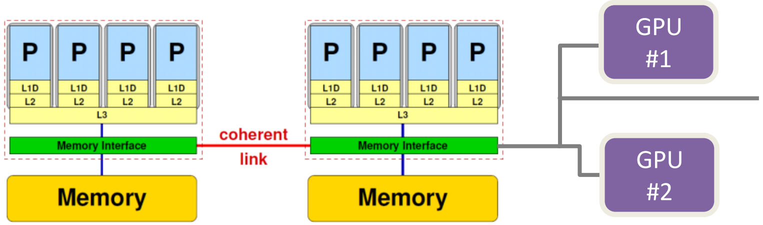
ECM Model

Typical bottlenecks in scientific computing

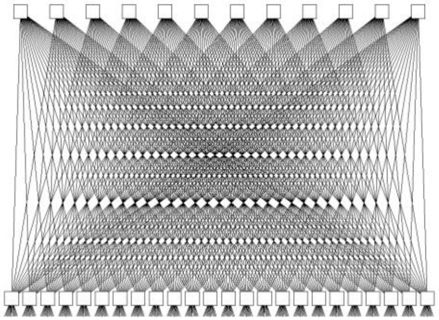
- **Chip level**
  - Execution units, pipelines
  - Cache transfer bandwidths
  - Memory bandwidth



- **Node level**
  - Intra-node communication (NUMA, PCI)
  - Network connection(s)



- **System level**
  - Network topology
  - Power constraints

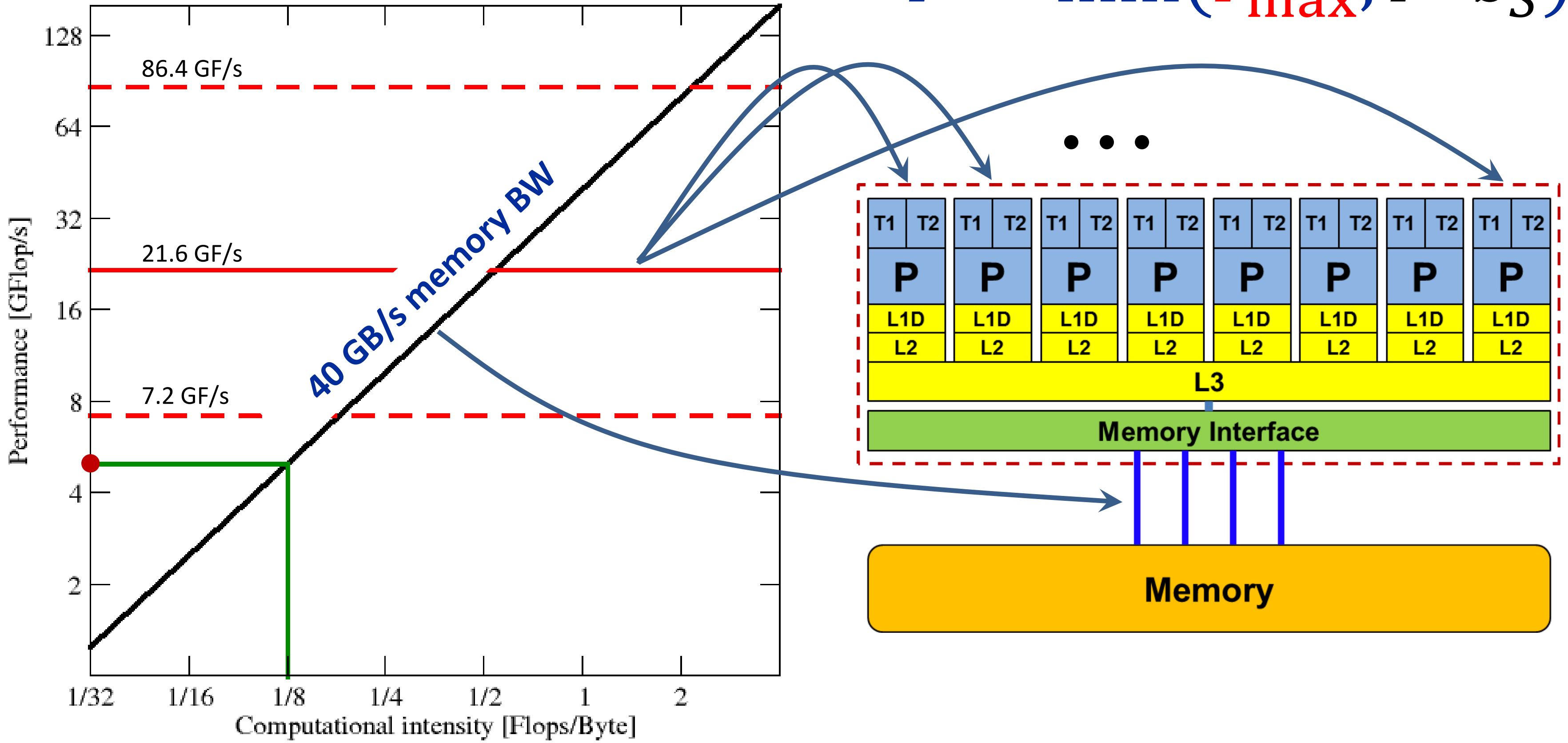


How do you know that you have hit a bottleneck? → **Performance modeling!**

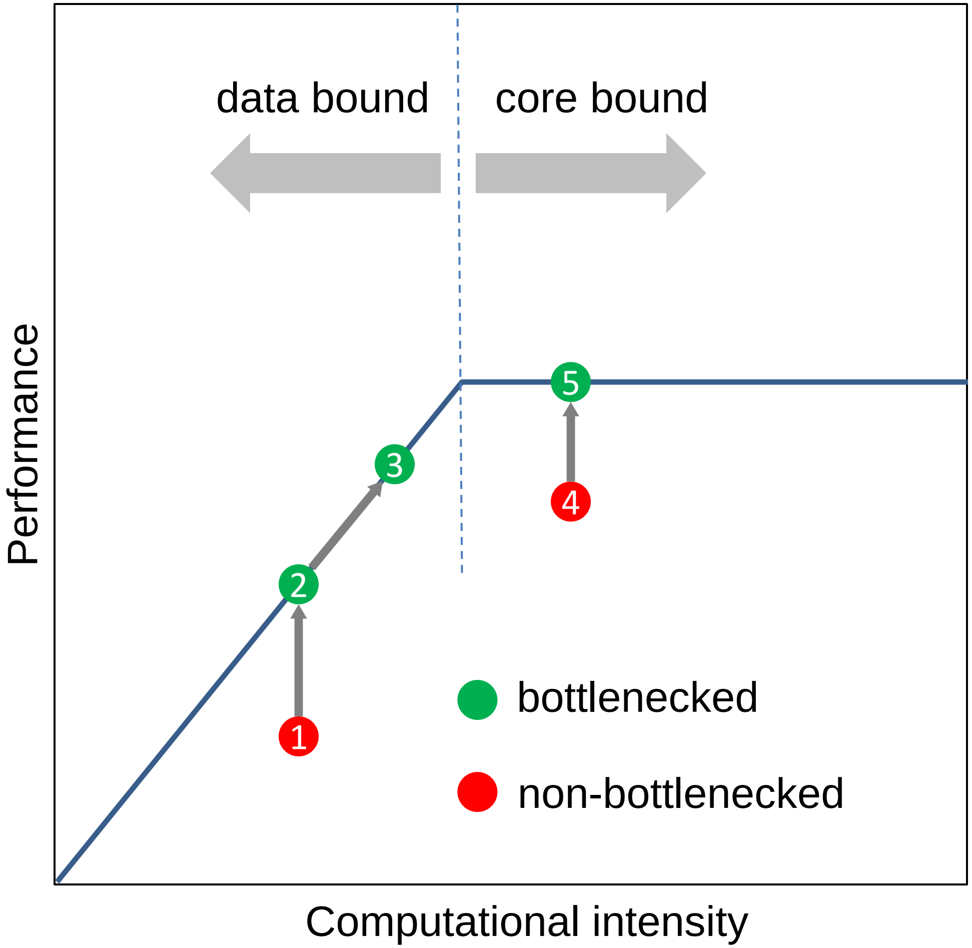
Performance modeling

Simplest chip-level approach: The Roofline Model

$$P = \min(P_{\mathrm{max}}, I \cdot b_S)$$



Roofline: Hitting bottlenecks



1: memory-bound, but inefficient access?

1→2: Optimization fixes access problems to hit bottleneck

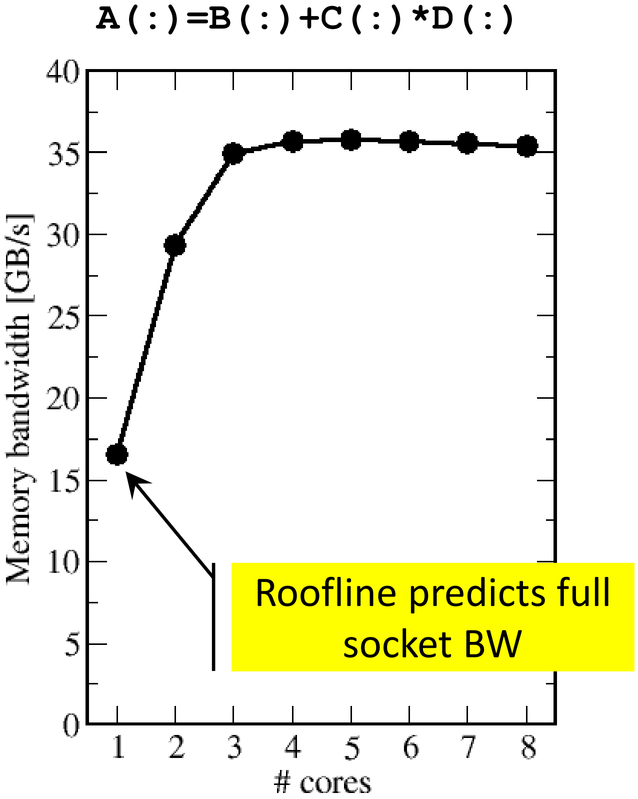
2→3: Optimization increases comp. intensity while staying at bottleneck

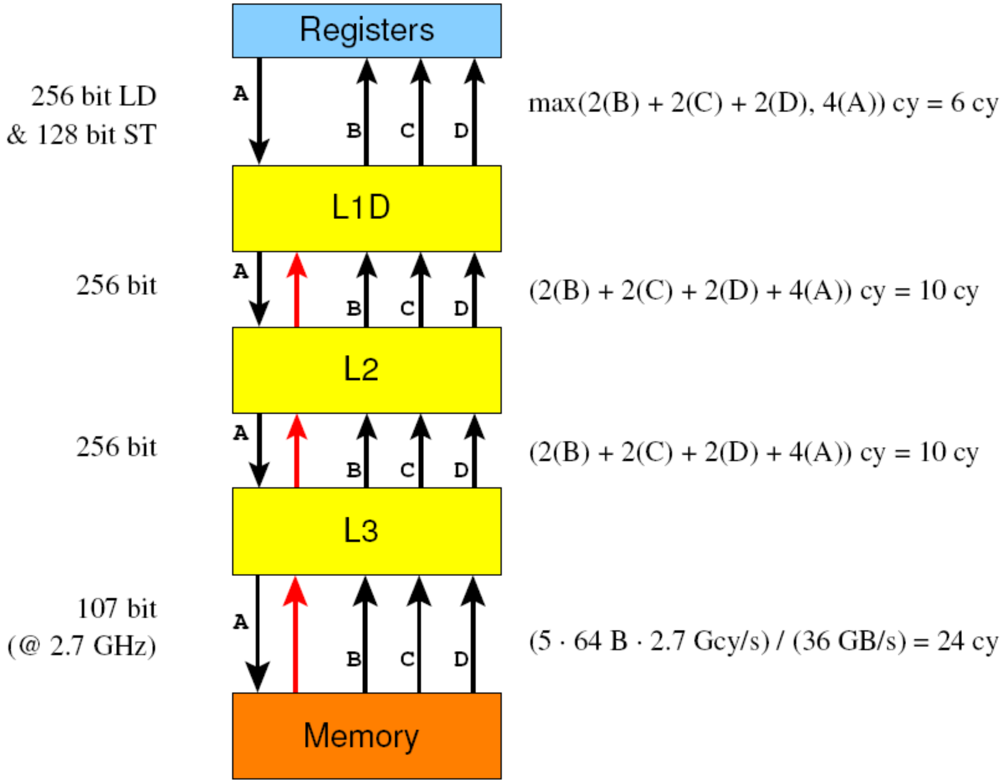
4: compute-bound, but inefficient execution – no SIMD?

4→5: Optimization fixes execution to hit bottleneck

Roofline → ECM (Execution-Cache-Memory)

Problem: Roofline does not explain intra-chip saturation

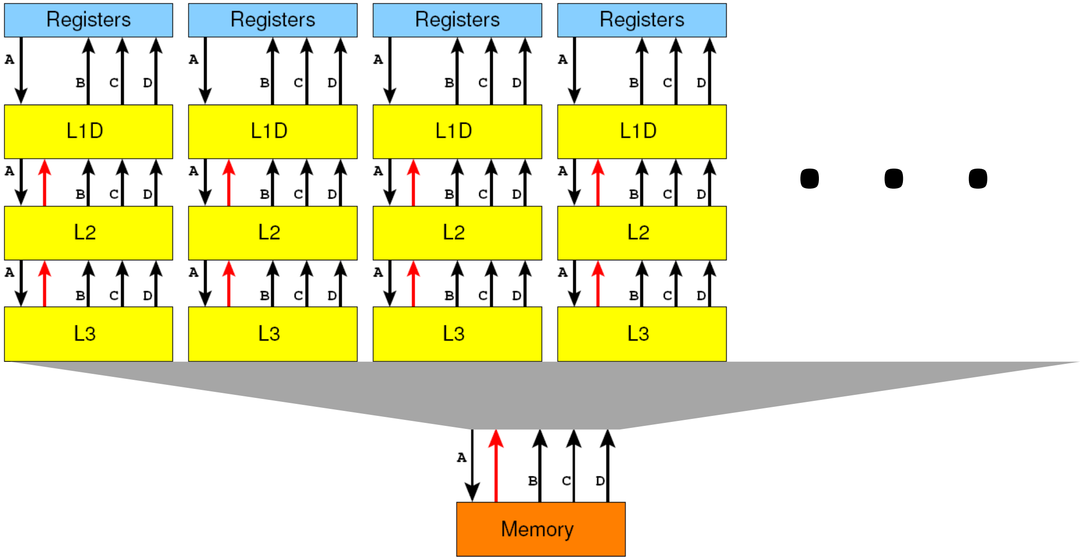



Registers
L1D
L2
L3
Memory
A
B
C
D
256 bit LD & 128 bit ST
256 bit
256 bit
107 bit (@ 2.7 GHz)
max(2(B) + 2(C) + 2(D), 4(A)) cy = 6 cy
(2(B) + 2(C) + 2(D) + 4(A)) cy = 10 cy
(2(B) + 2(C) + 2(D) + 4(A)) cy = 10 cy
(5 · 64 B · 2.7 Gcy/s) / (36 GB/s) = 24 cy

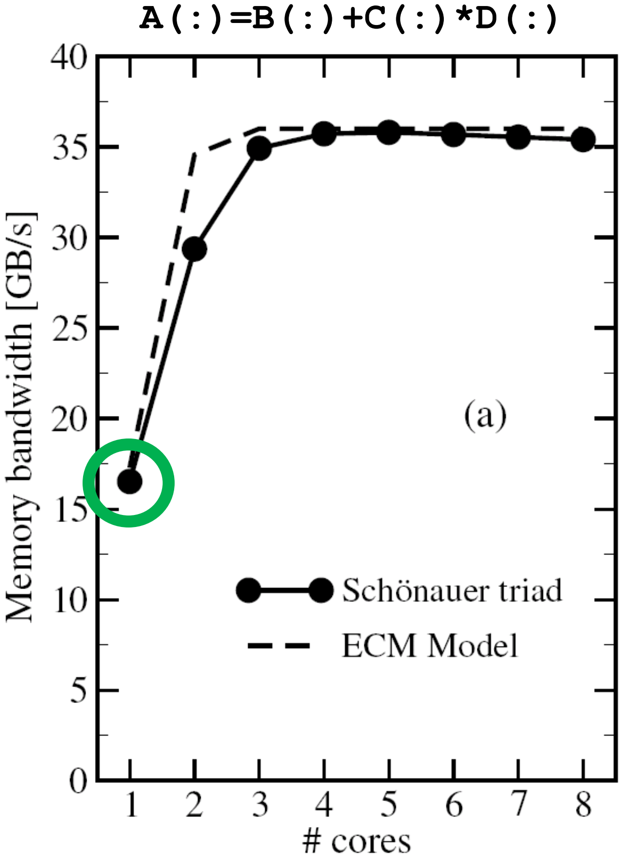
ECM Model accounts for lost cycles by considering data transfers through cache hierarchy …

ECM bandwidth saturation

… then assumes perfect scaling until the bottleneck is hit

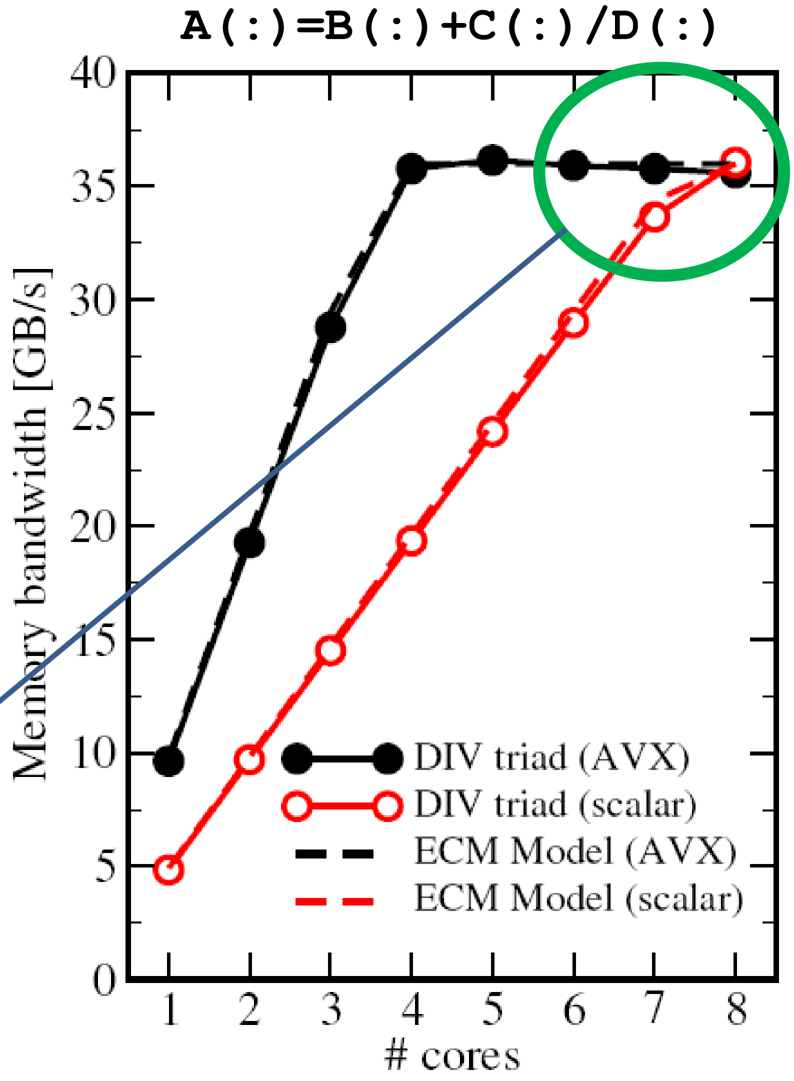


J. Treibig and G. Hager: Introducing a Performance Model for Bandwidth-Limited Loop Kernels. PPAM 2009, DOI: 10.1007/978-3-642-14390-8_64. arXiv:0905.0792
G. Hager, J. Treibig, J. Habich and G. Wellein: Exploring performance and power properties of modern multicore chips via simple machine models. Submitted.
Preprint: arXiv:1208.2908



Healing slow serial code

So why the fuss if we have enough cores to saturate anyway?



Parallelism "heals" bad single-core performance … if you are lucky!

Setting the Stage (II): Energy Consumption

A simple power model for multicore chips

Assumptions:

1. **Power is a quadratic polynomial in the clock frequency *f***
2. **Dynamic power is linear in the number of active cores *t***
3. **Performance is linear in the number of cores until it hits a bottleneck (← ECM model)**
4. **Performance is linear in the clock frequency unless it hits a bottleneck**
5. **Energy to solution is power dissipation divided by performance**

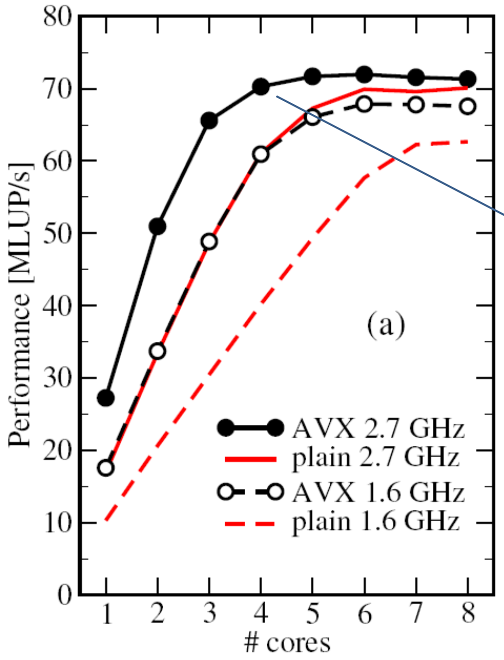
Model:

$$\boxed{E = \frac{\text{Power}}{\text{Performance}} = \frac{W_0 + W_2 f^2 t}{\min(tP_0\, f/f_0\,, P_{max})}}$$

Model predictions

$$E = \frac{W_0 + W_2 f^2 t}{\min(t P_0\, f/f_0\,, P_{max})}$$

1. **Making code execute faster on the core saves energy since**
   - The time to solution is smaller if the code scales ("Code race to idle")
   - We can use fewer cores to reach saturation if there is a bottleneck

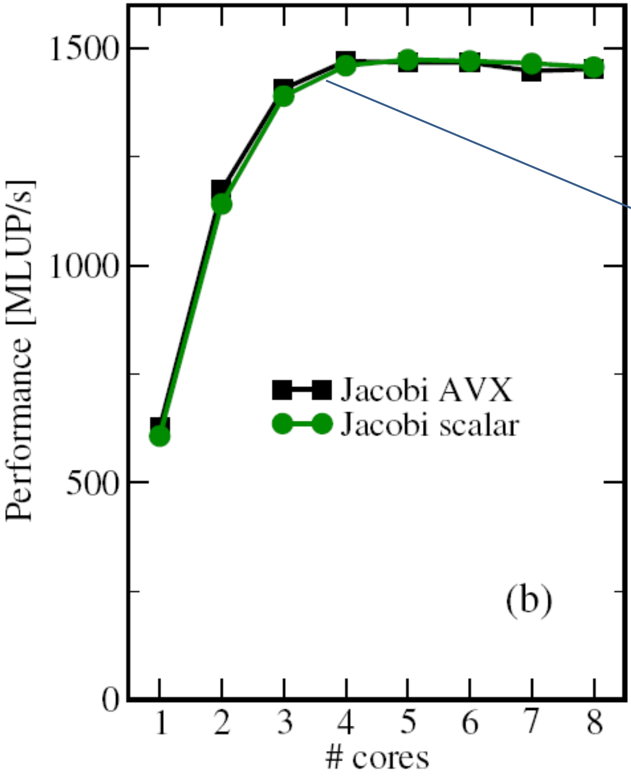


Better code
→ earlier saturation
→ smaller E @ saturation

Model predictions

$$E = \frac{W_0 + W_2 f^2 t}{\min(tP_0\, f/f_0\,, P_{max})}$$

2. **If there is saturation, *E* is minimal near the saturation point**



Minimum E here

$$t_s = \frac{P_{max}}{P_0 f/f_0}$$

Model predictions

$$E = \frac{W_0 + W_2 f^2 t}{\min(tP_0\, f/f_0\,, P_{max})}$$

3. There is an optimal frequency ***f*$_{\text{opt}}$** at which ***E*** is minimal in the **non-saturated** case, with

$$f_{\text{opt}} = \sqrt{\frac{W_0}{W_2 t}}$$ (depends on the baseline power)

→ "**Clock race to idle**" if baseline power is large (accommodates whole system)!

Putting it all together:
Chip-Level Energy vs. Performance

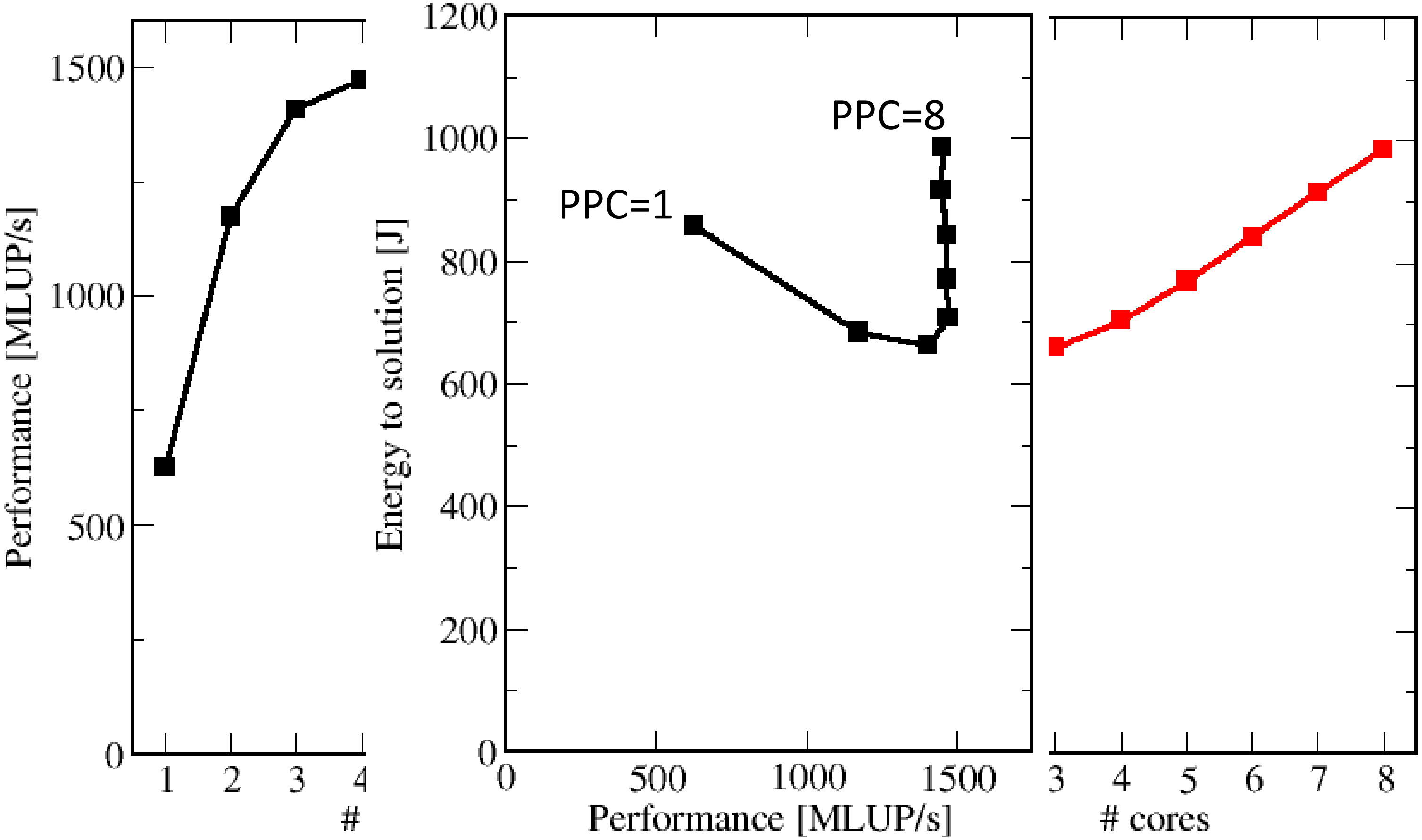
Memory-bound codes

Scalable codes

Case 1: Memory bound (saturating)

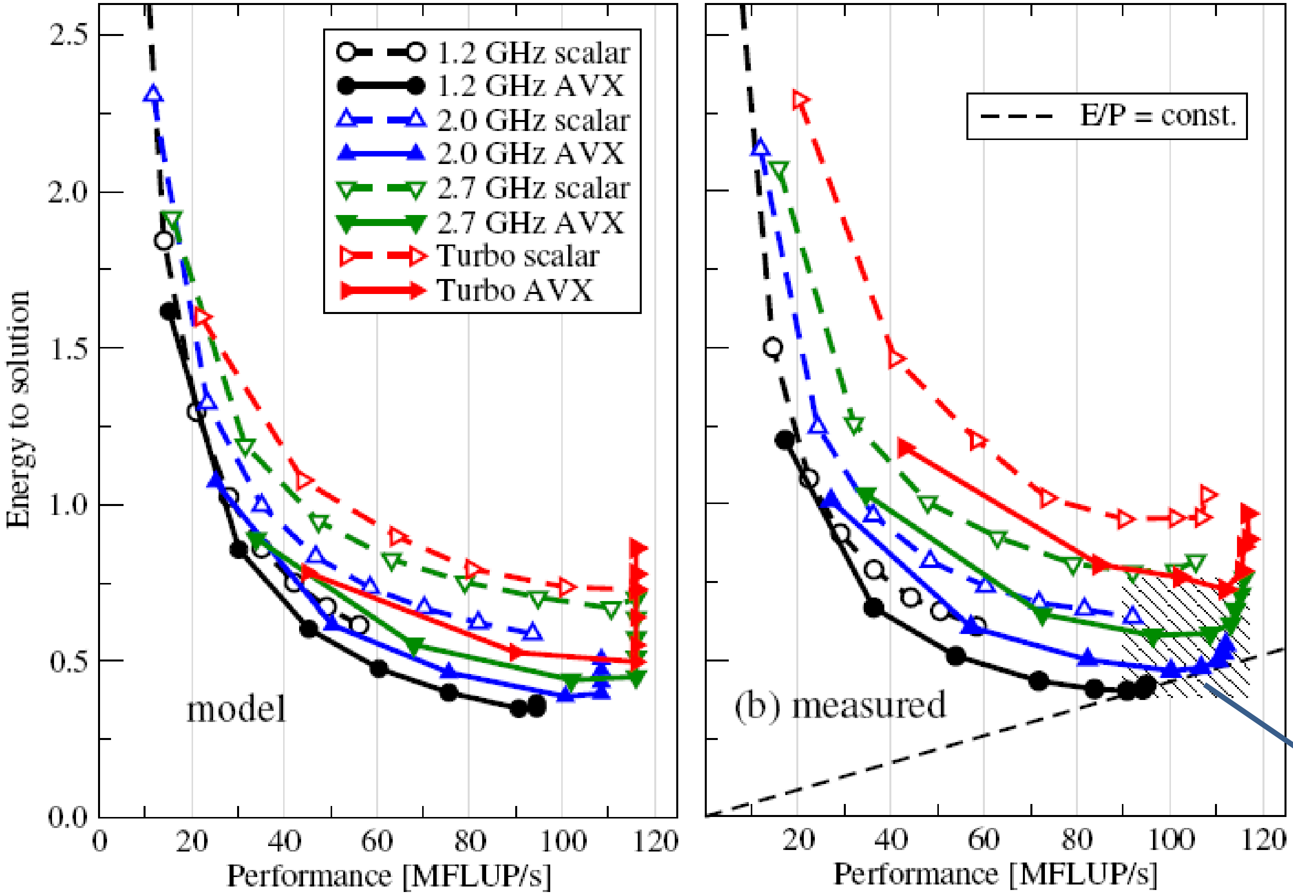
A simple example: Jacobi smoother Z-plot

Peformance & energy to solution (chip-level base power $W_0 = 23W$) @ 2.7 GHz on Sandy Bridge EP



A lattice-Boltzmann flow solver on the Sandy Bridge chip

ECM + Power model vs. measurements (chip level)

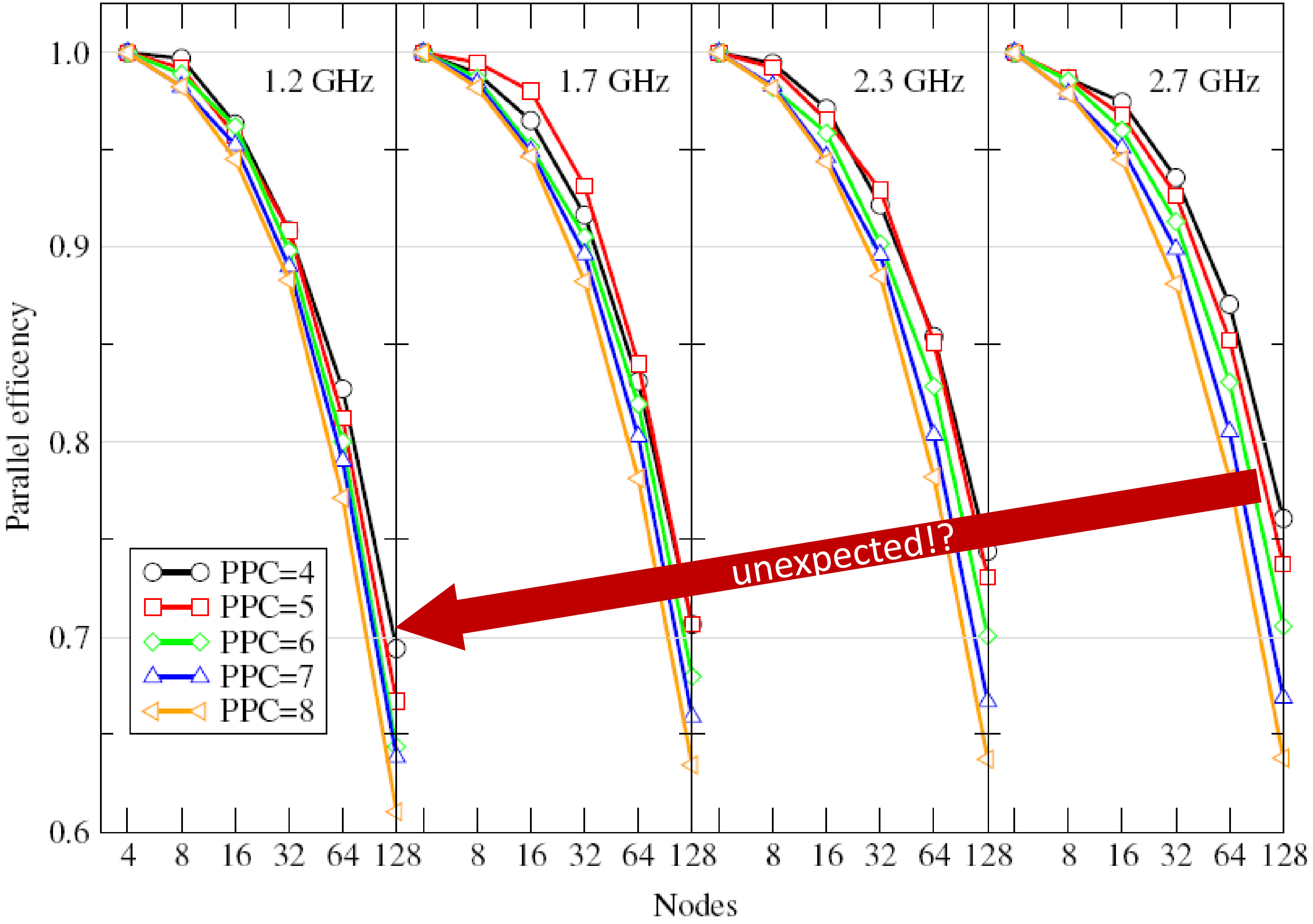


Lowest energy for

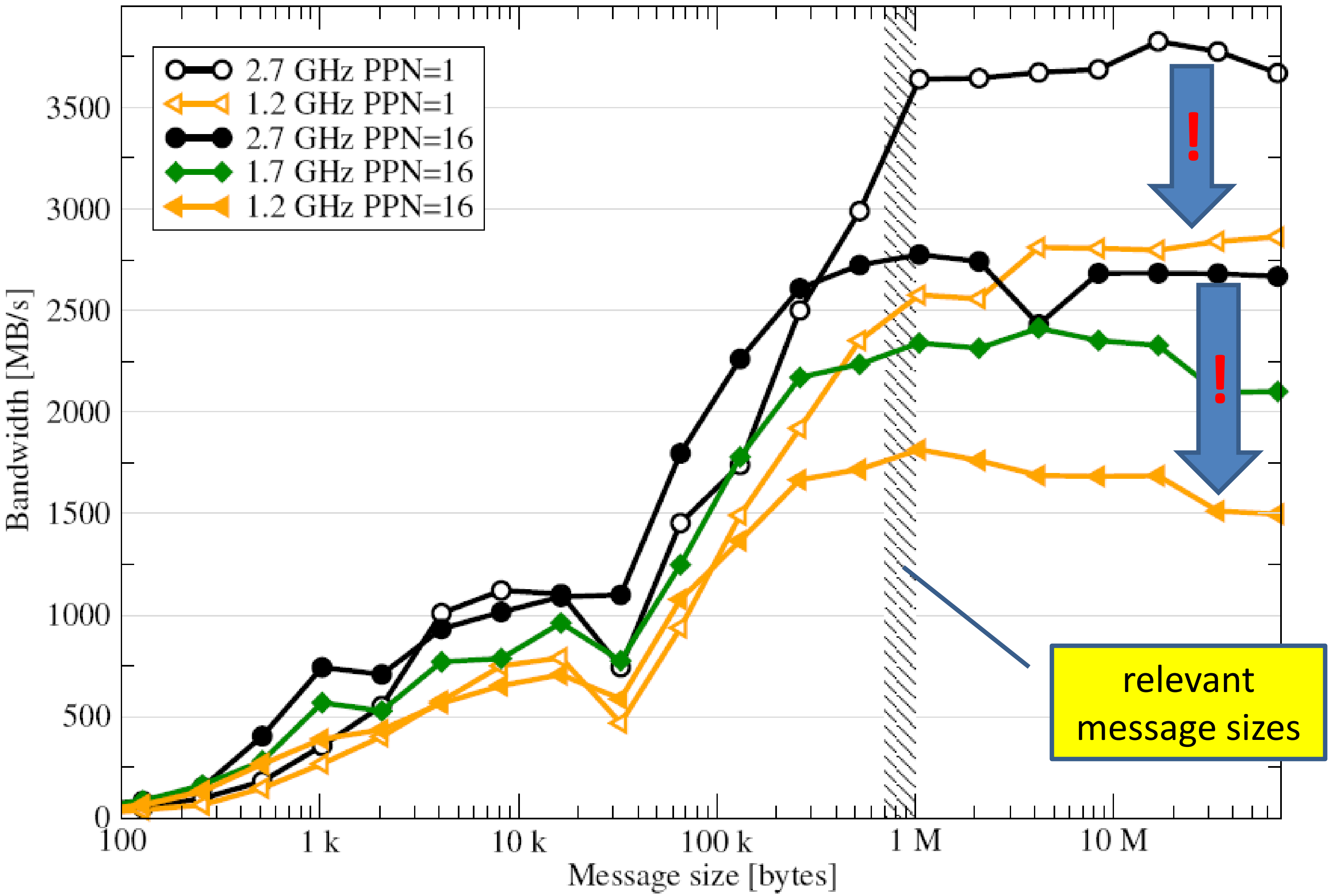
- best code (AVX)
- low-ish clock speed
- optimal number of cores (at bottleneck)

optimization space

Going highly parallel at different frequencies on SuperMUC

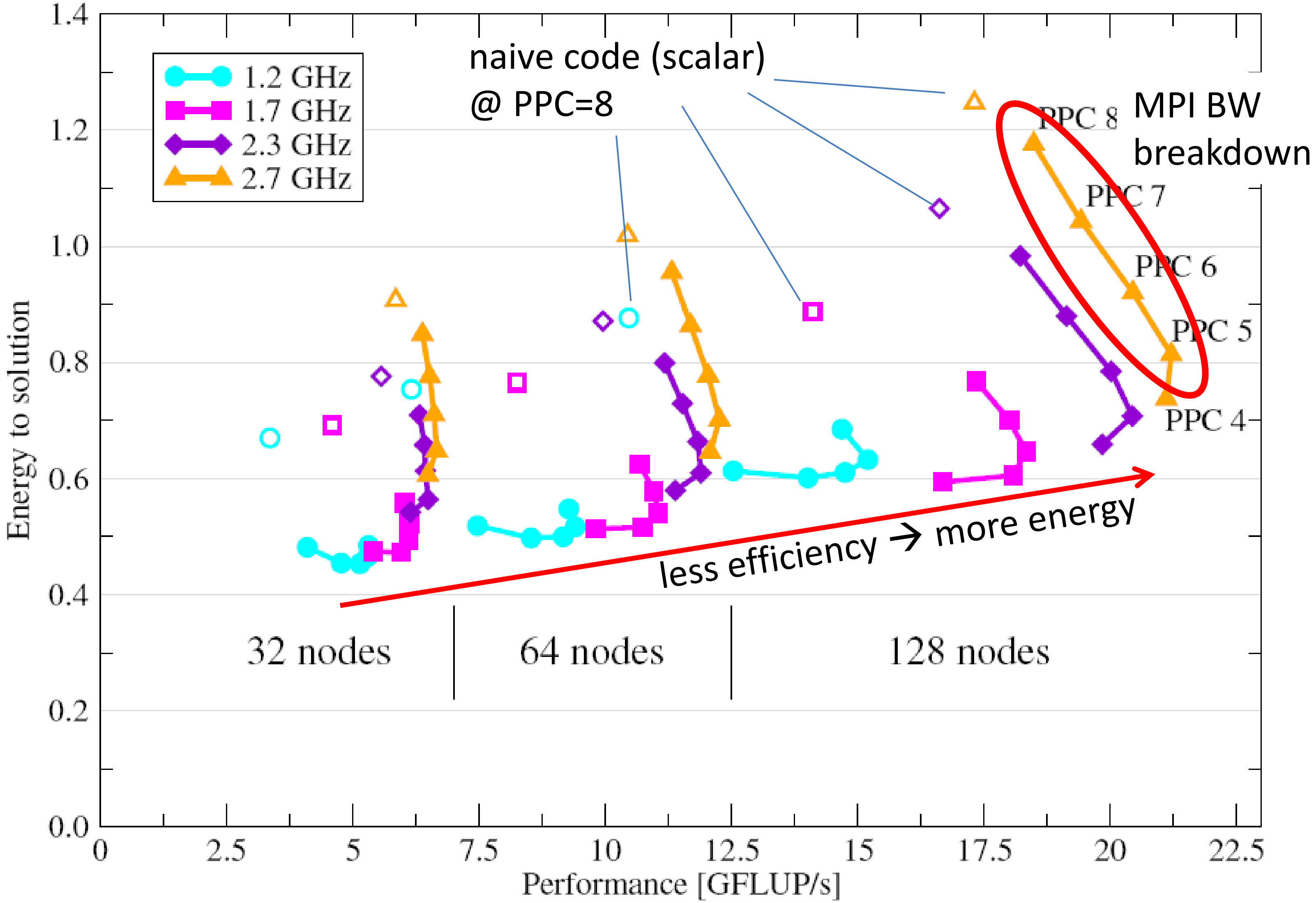


MPI Sendrecv test (mimics halo exchange) on SuperMUC

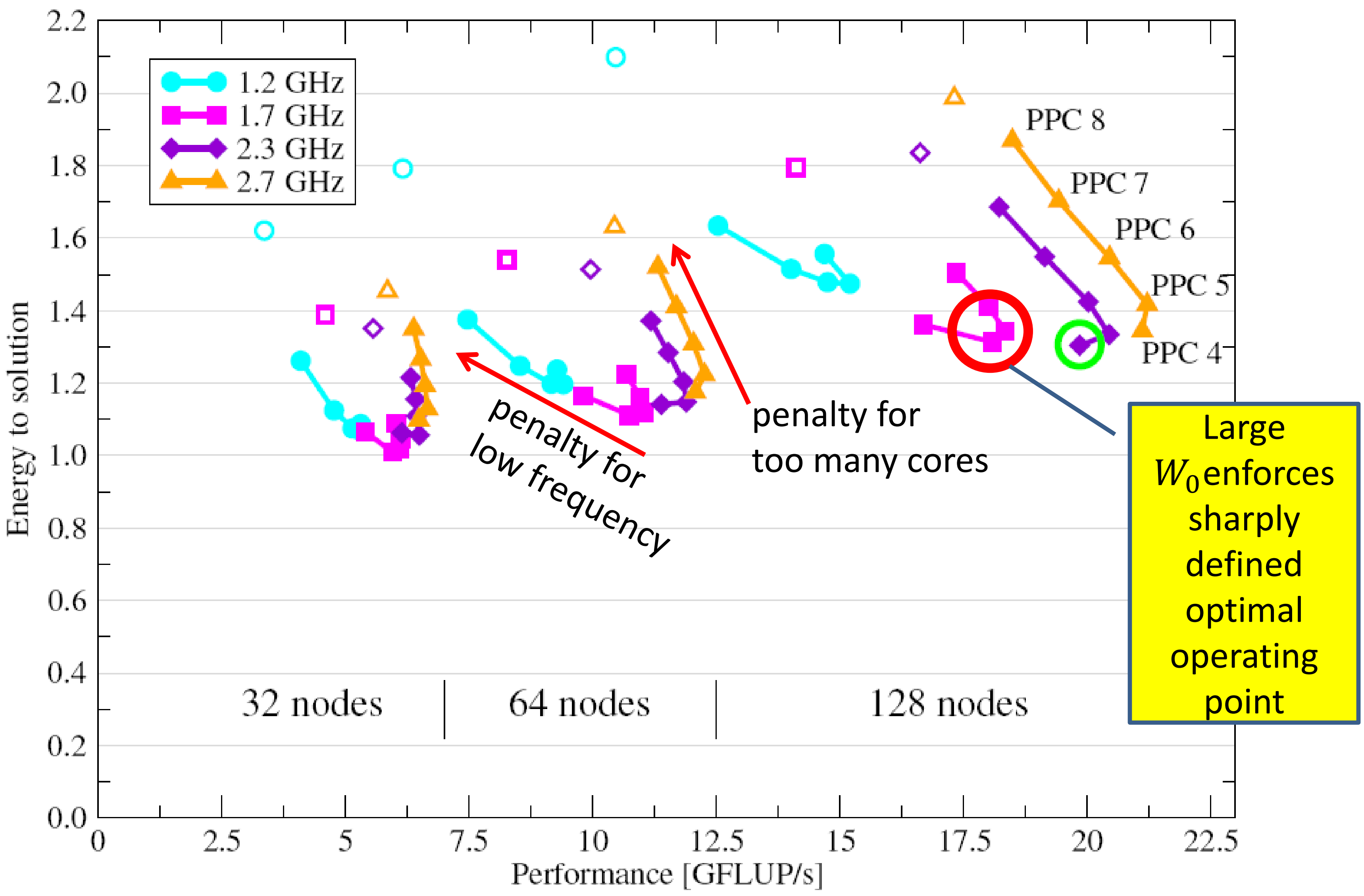


relevant message sizes

What now about the optimal operating point? (chip W_0)



… and taking a realistic $W_0 = 73\text{W}$?

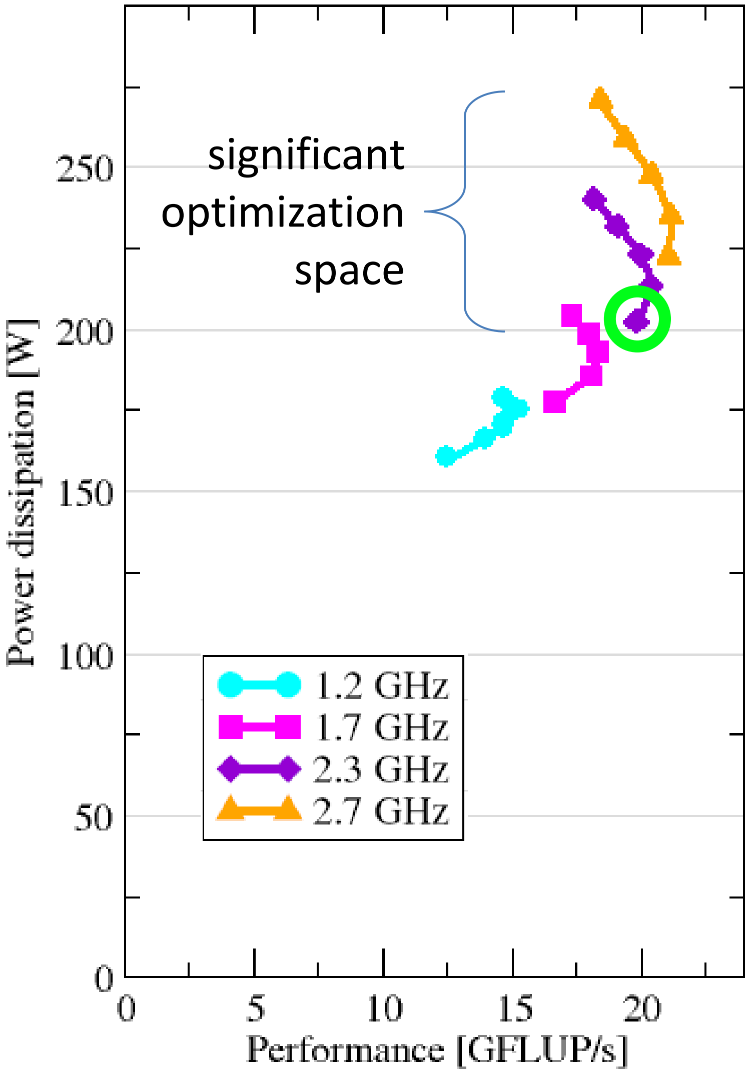


penalty for low frequency

penalty for too many cores

Large W_0 enforces sharply defined optimal operating point

Power capping (realistic W_0)



significant optimization space

Case 2: Cache bound (scalable)

A DGEMM test

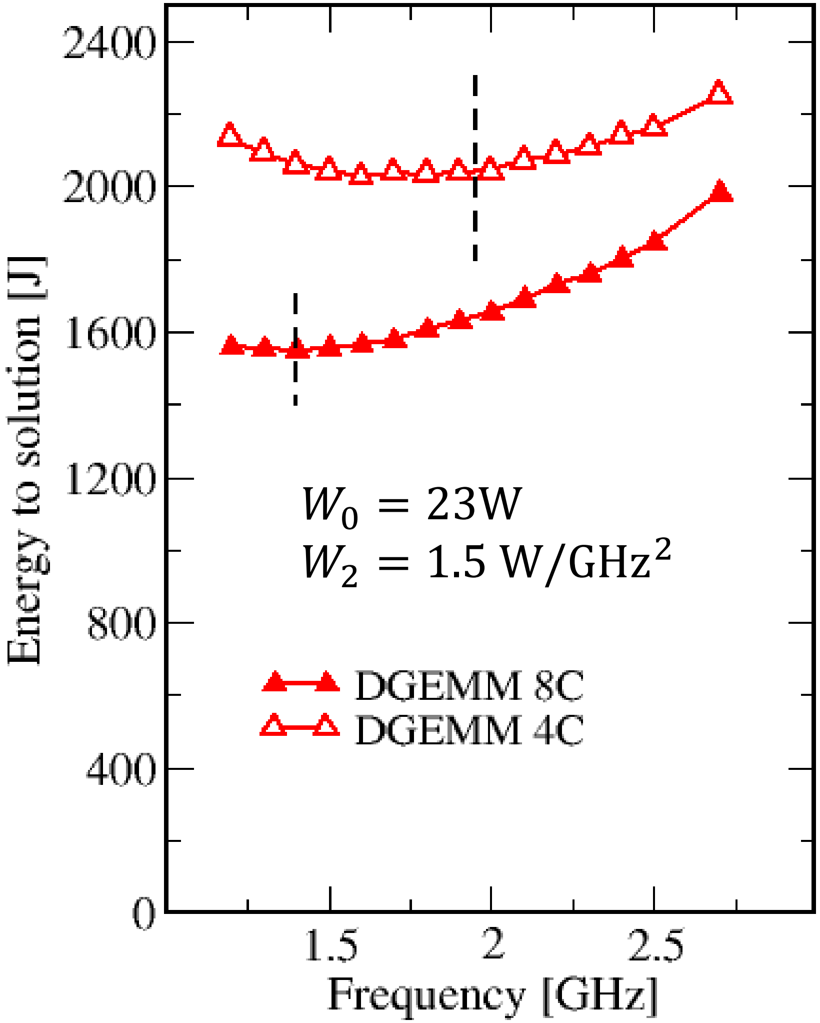
- Optimal frequency for energy to solution on scalable code:

$$f_{opt} = \sqrt{\frac{W_0}{W_2 t}}$$

- Power ratio of optimized vs. base clock speed:

$$\frac{W(f_{opt})}{W(f_0)} = \frac{2W_0}{W_0 + W_2 f_0^2 t}$$

- But clocking down gives me less science per CPU hour!?



Adjusting the size of the machine for scalable load

- **Invest the saved energy into a larger machine to get the same science over its lifetime:**

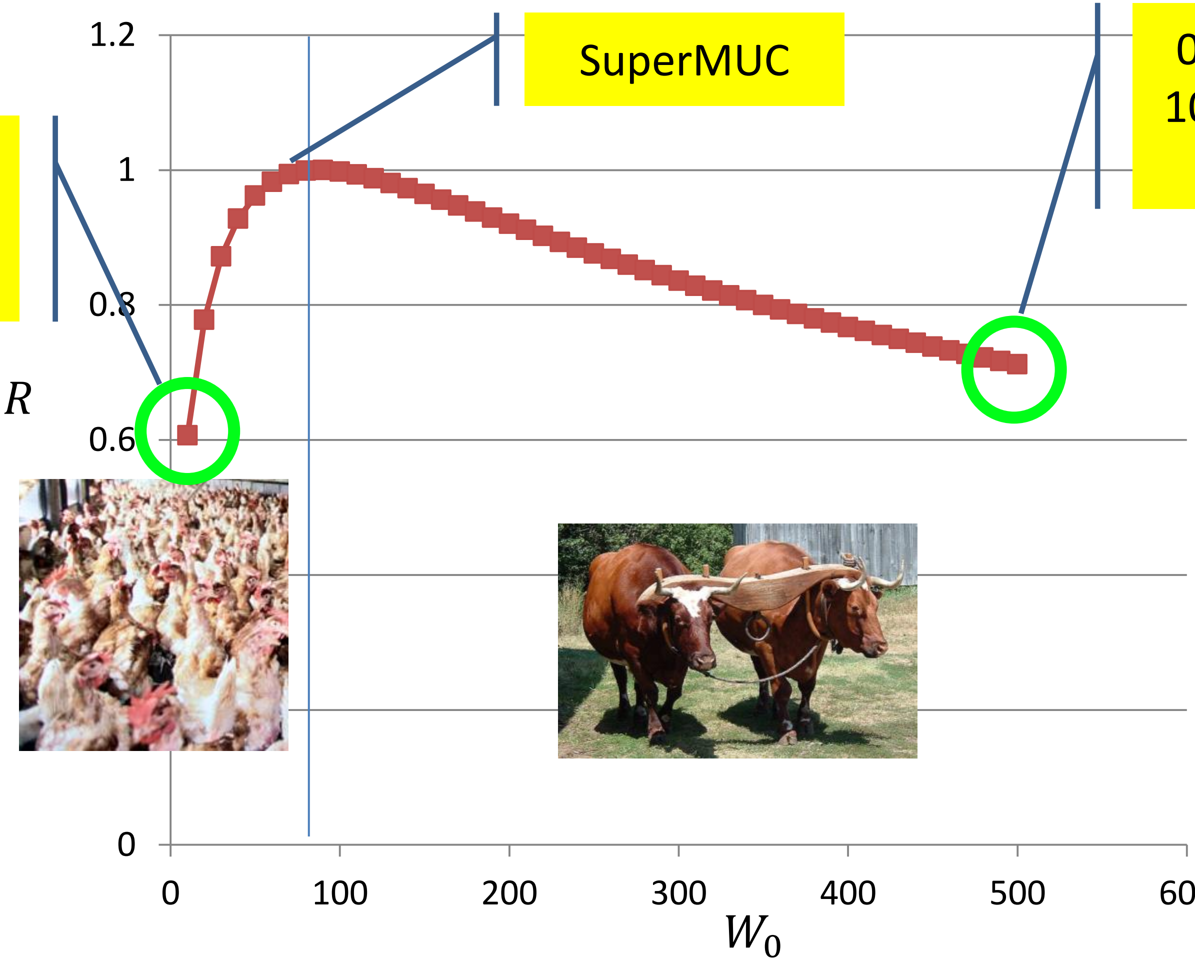
$$R = \frac{W(f_{opt})}{W(f_0)} \cdot \frac{f_0}{f_{opt}} = \frac{2f_0\sqrt{W_0 W_2 t}}{W_0 + W_2 f_0^2 t}$$

- ***R* quantifies the potential for saving energy over the lifetime of the machine (which is constant)**

Energy saving potential for different W_0

It’s chickens vs. oxen time again!

(8 cores, $W_2 = 1.5\ \mathrm{W/GHz^2}$, $f_0 = 2.7\ \mathrm{GHz}$)



SuperMUC

3x size cluster, 20W 900MHz chip

0.4x size cluster, 1000W 6500MHz chip

Conclusions

- **Performance and power models help us understand optimal operating points for saturating codes on the chip level**
  - Including code quality and clock speed dependence

- **This knowledge is even more important in the highly parallel case!**
  - Operating point for saturated codes more sharply defined if communication plays a significant role
  - Exploration of design space for energy-efficient large-scale systems
- **Blindly setting a slow clock speed for bandwidth-bound code may be dangerous**
  - … and the benefit is limited
- **Take-home messages**
  - Write fast single-core code
  - Know about saturation and dump dispensable cores
  - This is also crucial for power capping
  - Adjust clock speed – but do it intelligently!

Moritz Kreutzer
Markus Wittmann
Thomas Zeiser
Michael Meier
Jan Treibig

KONWIHR III
OMI4papps

Bundesministerium
für Bildung
und Forschung
hpcADD
FEPA
SKALB

THANK YOU.