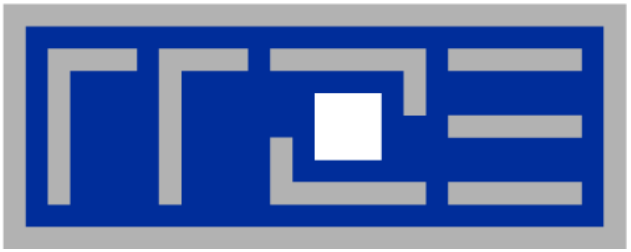
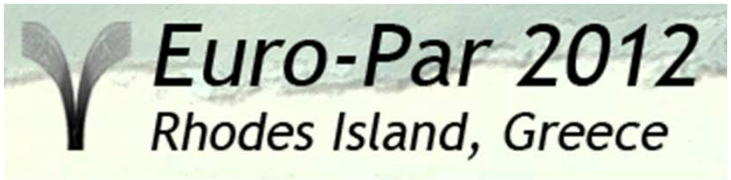


Performance and Power Engineering: From Numbers to Insight

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

PROPER Workshop at Euro-Par 2012
August 28, 2012

Rhodes Island, Greece



Prerequisites

Why care about program performance?

1. **Show off at parties?**
2. **Win the Gordon Bell Award?**
3. **Solve problems faster or solve larger problems in acceptable time!**

How do I know that performance is “good”?

1. **Good scalability across cores?**
2. **High fraction of peak performance?**
3. **Code execution hits the relevant bottleneck!**

… and how do I know what the relevant bottleneck is?

→ Performance Modeling!

What should be modeled?

- **The ability of a programmer, framework, library, compiler to generate efficient code?**

- **The impact of a set of hardware metrics on application performance and scalability?**

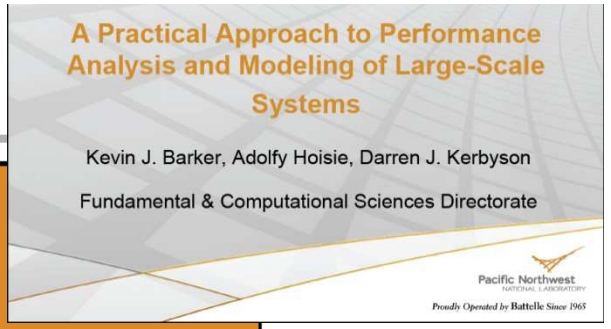
- **The performance of (ideally)**
  - … an implementation of an algorithm
  - … on kernel, solver, application levels
  - … on a compute node, network, full system

How should it be modeled?

- **“Black box” approach?**
  - Determine utilization of processor resources, network, file system at runtime
  - Determine performance of given application for different input sets for a given architecture
  - Determine correlation of certain hardware metrics with performance behavior
  - Automatic “tuning”: Scan all implementation alternatives for best performance

- **“White box” approach!**
  - Set up an (analytical) model for a given algorithm/kernel/solver/application on a given architecture
  - Compare with measurements to validate the model
  - (Probably) identify optimization opportunities and start again

Others have said it better…



Overview of Performance Modeling

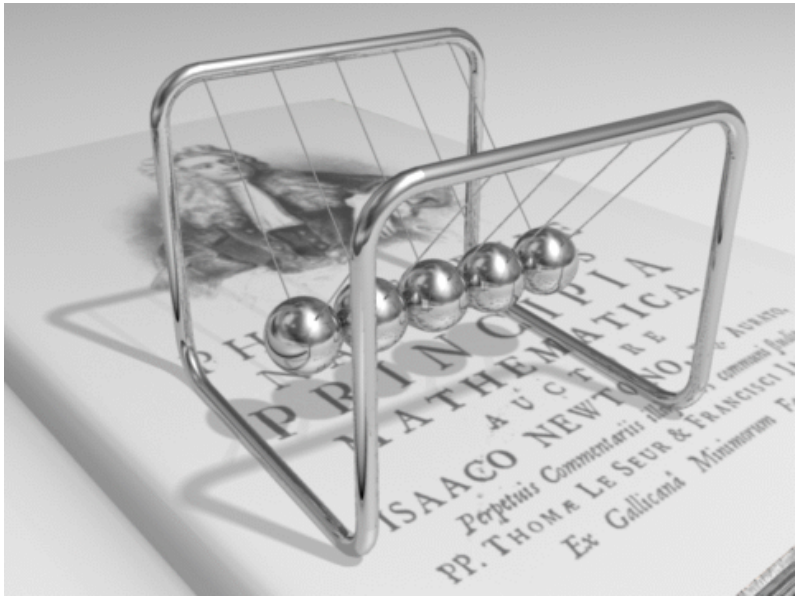
Performance modeling	"Brand X"
Relies on understanding of true application behavior	Relies on pattern matching and curve fitting (extrapolation & interpolation)
White-box approach (application-centric)	Black-box approach (application-oblivious)
Explains performance; detail of explanation correlates with accuracy of prediction	Predicts without providing insight or gauges of accuracy
Disagreements with measurements challenge assumptions and yield new insights	Disagreements with measurements merely showcase limitations of approach

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

An example from physics

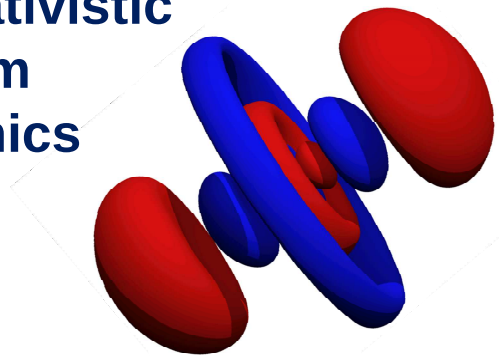
Newtonian mechanics



$$\vec{F} = m\vec{a}$$

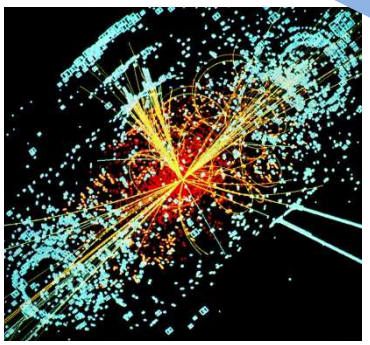
Fails @ small scales!

Nonrelativistic quantum mechanics



$$i\hbar\frac{\partial}{\partial t}\psi(\vec{r},t) = H\psi(\vec{r},t)$$

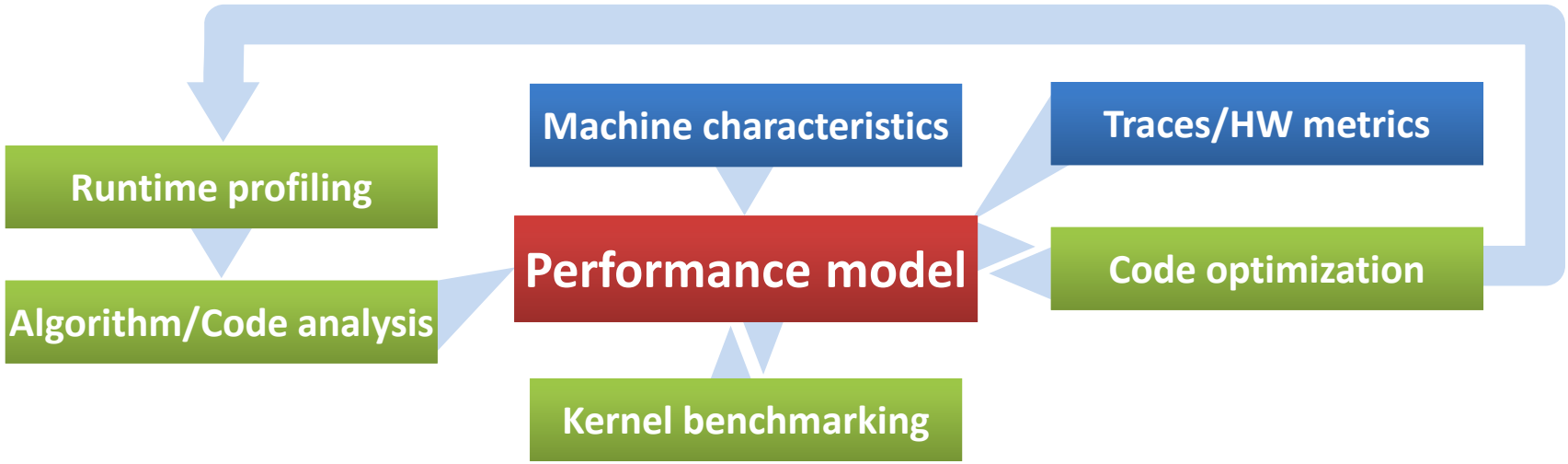
Fails @ even smaller scales!



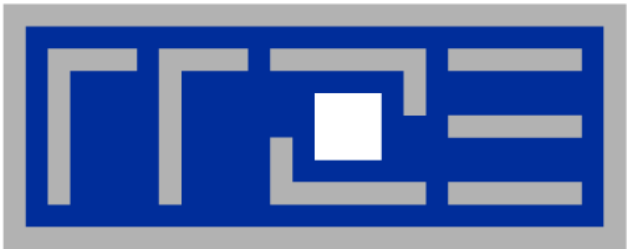
Relativistic quantum field theory

$$U(1)_Y \otimes SU(2)_L \otimes SU(3)_c$$

The Performance Engineering (PE) process:



The performance model is the central component



“White Box” Models on the chip level

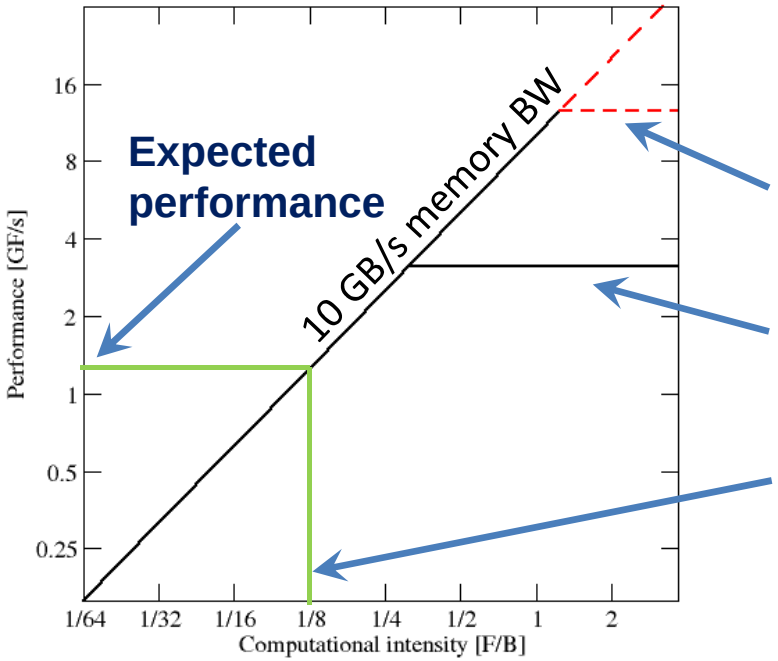
Roofline model

ECM model

Power modeling for multicore

An example: The roofline model

1. Determine the applicable peak performance of a loop, assuming that data comes from L1 cache
2. Determine the data traffic per Flop over the slowest data path utilized
3. Determine the applicable peak bandwidth of the slowest data path utilized



Example: `do i=1,N; s=s+a(i); enddo`
in DP on hypothetical CPU, N large

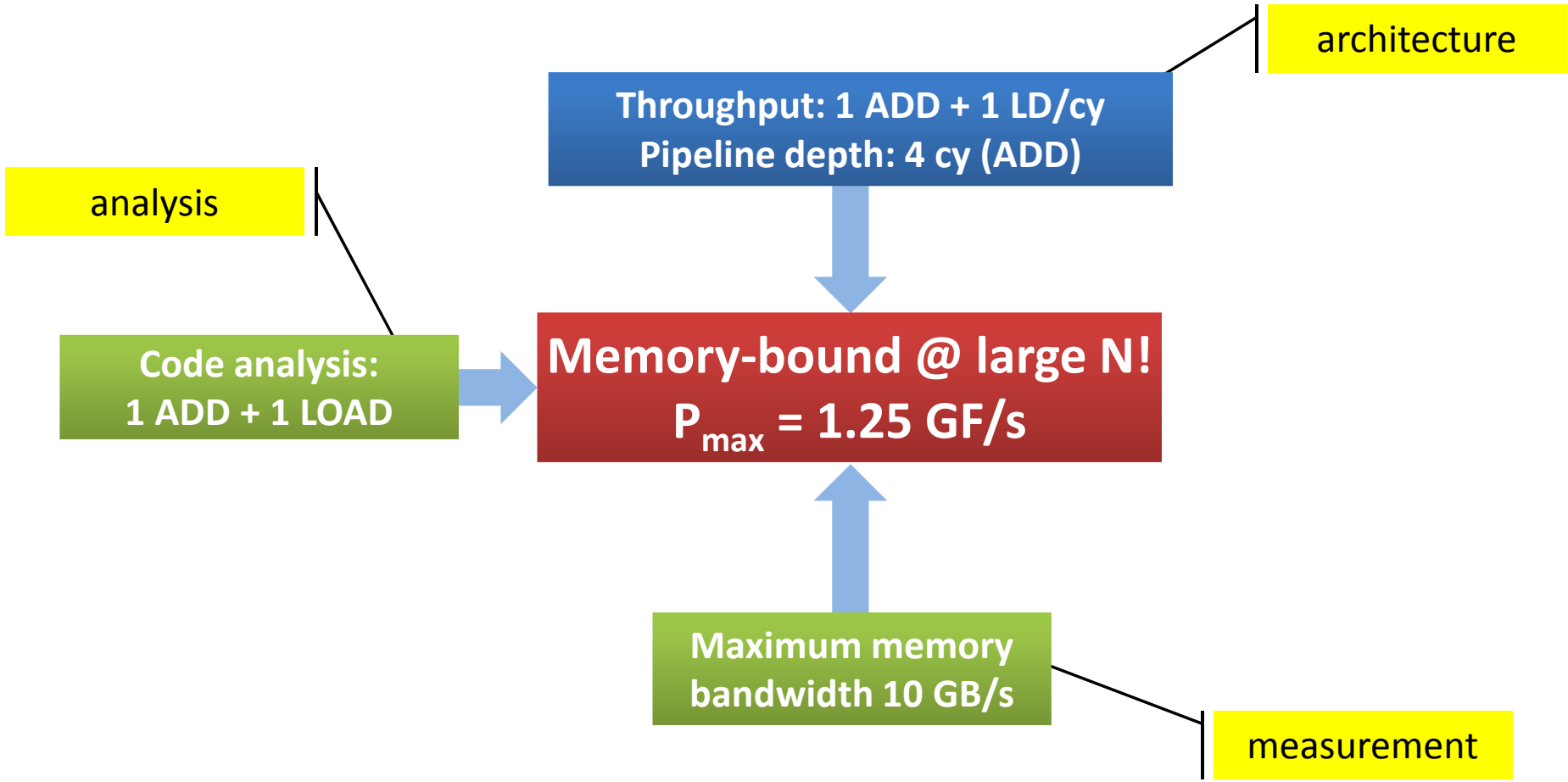
ADD peak (half of full peak)

4-cycle latency per ADD if not unrolled

Computational intensity

Input to the roofline model

… on the example of `do i=1,N; s=s+a(i); enddo`



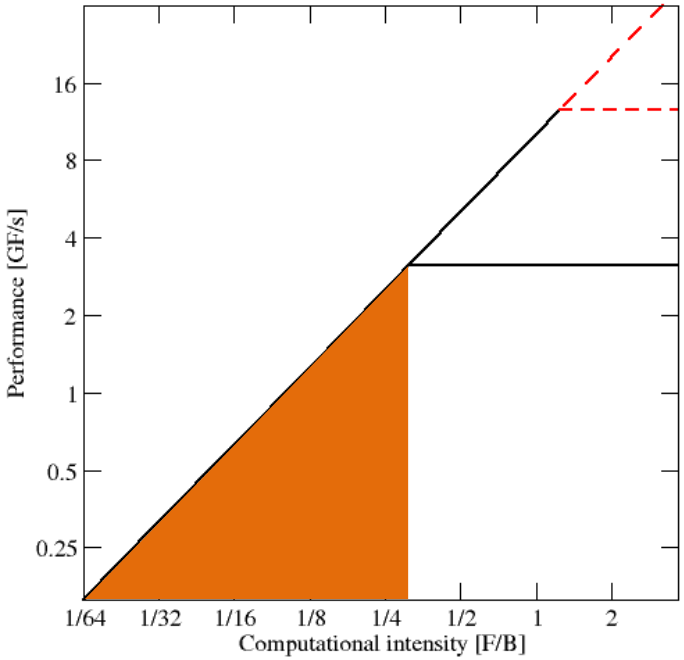
Factors to consider in the roofline model

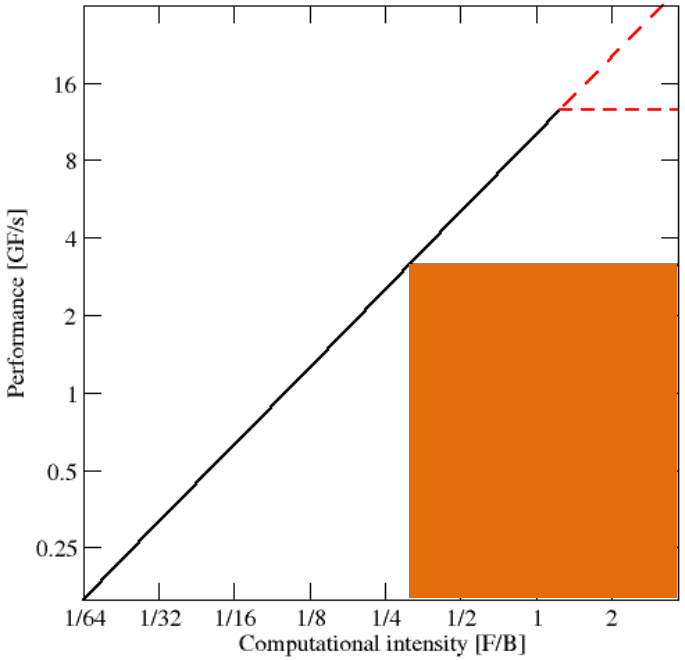
Bandwidth-bound (simple case)

- Accurate traffic calculation (write-allocate, strided access, …)
- Practical ≠ theoretical BW limits
- Erratic access patterns

Core-bound (may be complex)

- Multiple bottlenecks: LD/ST, arithmetic, pipelines, SIMD, execution ports
- Still probably some contributions from data access

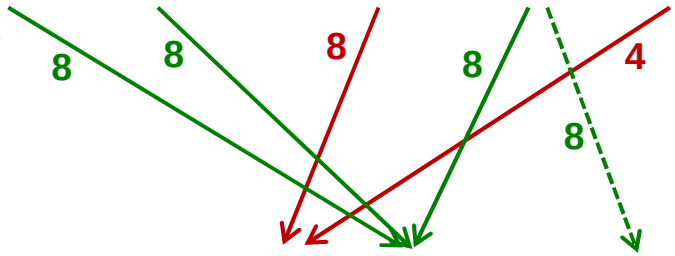




Example: SpMVM node performance model

- **Sparse MVM in double precision w/ CRS:**

```
do i = 1,Nr
  do j = row_ptr(i), row_ptr(i+1) - 1
    C(i) = C(i) + val(j) * B(col_idx(j))
  enddo
enddo
```



- **DP CRS code balance**
  - κ quantifies extra traffic for loading RHS more than once
  - Predicted Performance = streamBW/B_{CRS}

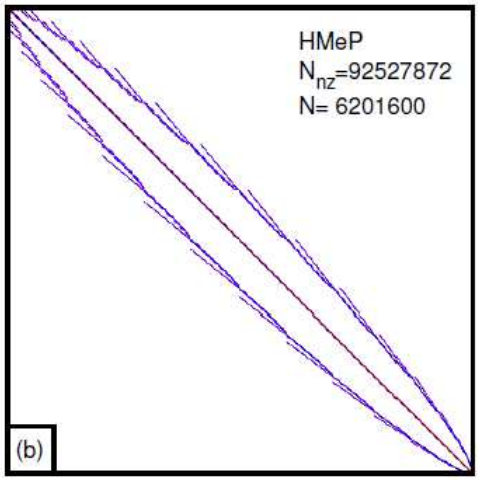
$$B_{\mathrm{CRS}} = \left(\frac{12+24/N_{\mathrm{nzr}}+\kappa}{2}\right)\frac{\mathrm{bytes}}{\mathrm{flop}} = \left(6+\frac{12}{N_{\mathrm{nzr}}}+\frac{\kappa}{2}\right)\frac{\mathrm{bytes}}{\mathrm{flop}}\,.$$

  - Determine κ by measuring performance and actual memory bandwidth

Test matrices: Sparsity patterns

- **Analysis for HMeP matrix on Nehalem EP socket**
  - BW used by spMVM kernel = 18.1 GB/s → should get ≈ 2.66 Gflop/s spMVM performance if $\kappa = 0$
  - Measured spMVM performance = 2.25 Gflop/s
  - Solve 2.25 Gflop/s = BW/B_{CRS} for $\kappa \approx 2.5$

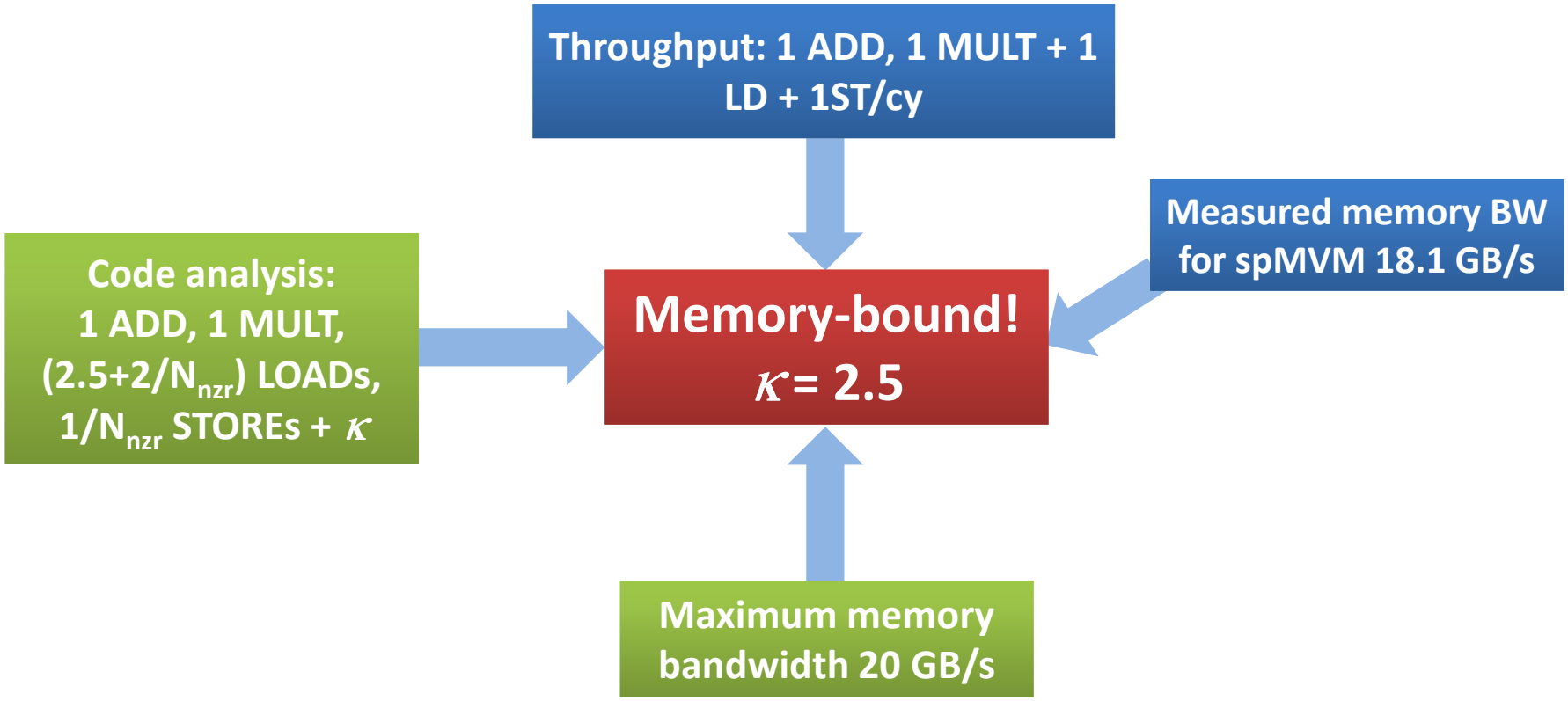
    → 37.5 extra bytes per row
    → RHS is loaded 6 times from memory
    → about 33% of BW goes into RHS



- **Conclusion: Even if the roofline model does not work 100%, we can still learn something from the deviations**

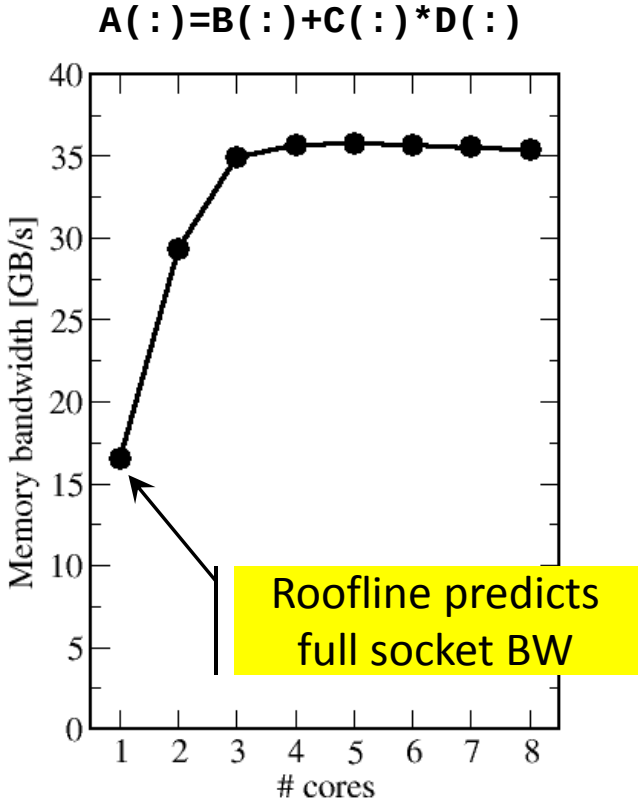
Input to the roofline model

… on the example of spMVM with HMeP matrix



Assumptions and shortcomings of the roofline model

- **Assumes one of two bottlenecks**
  1. In-core execution
  2. Bandwidth of a single hierarchy level
- **Latency effects are not modeled → pure data streaming assumed**
- **In-core execution is sometimes hard to model**
- **Saturation effects in multicore chips are not explained**

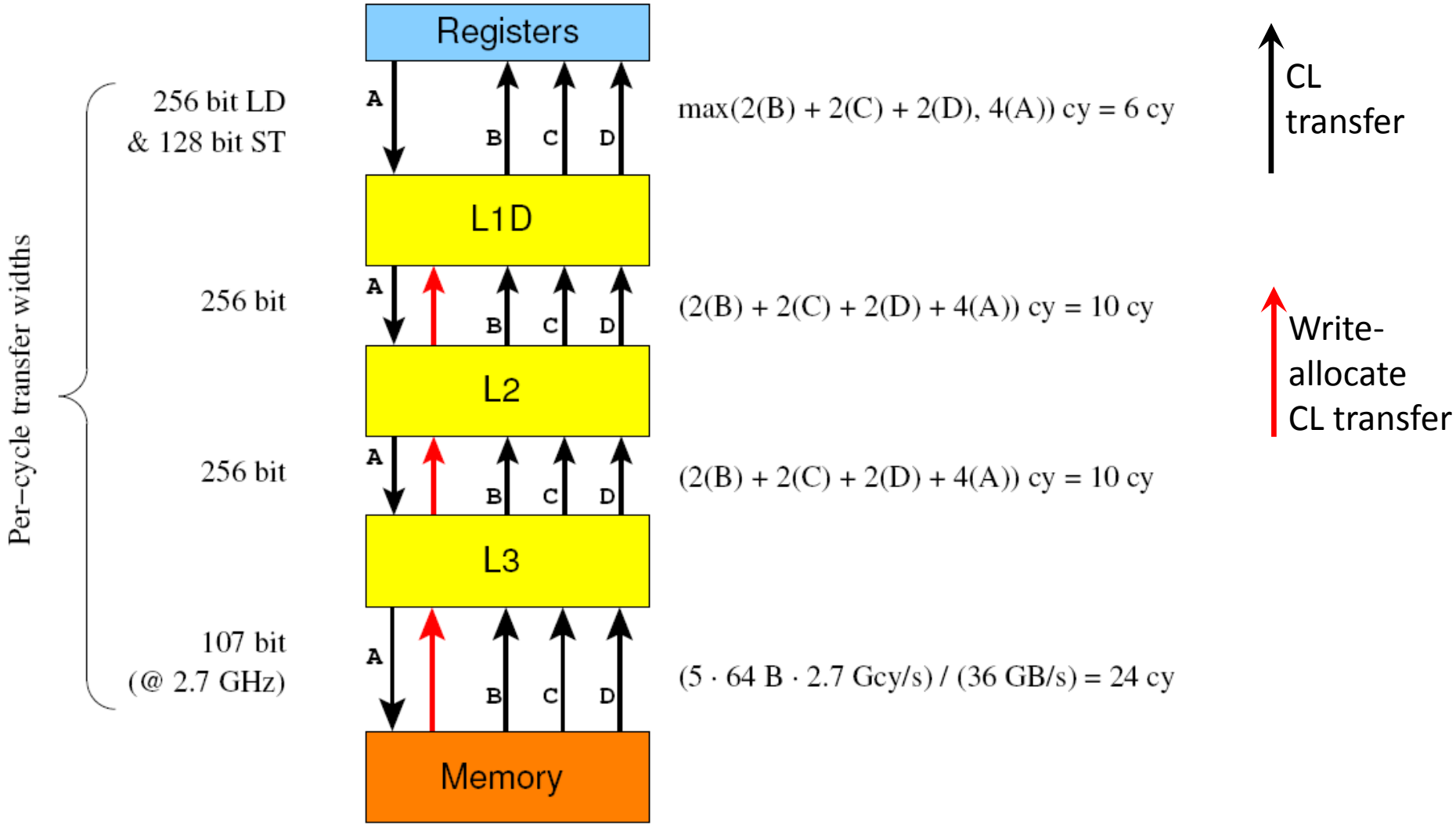


The multicore saturation mystery

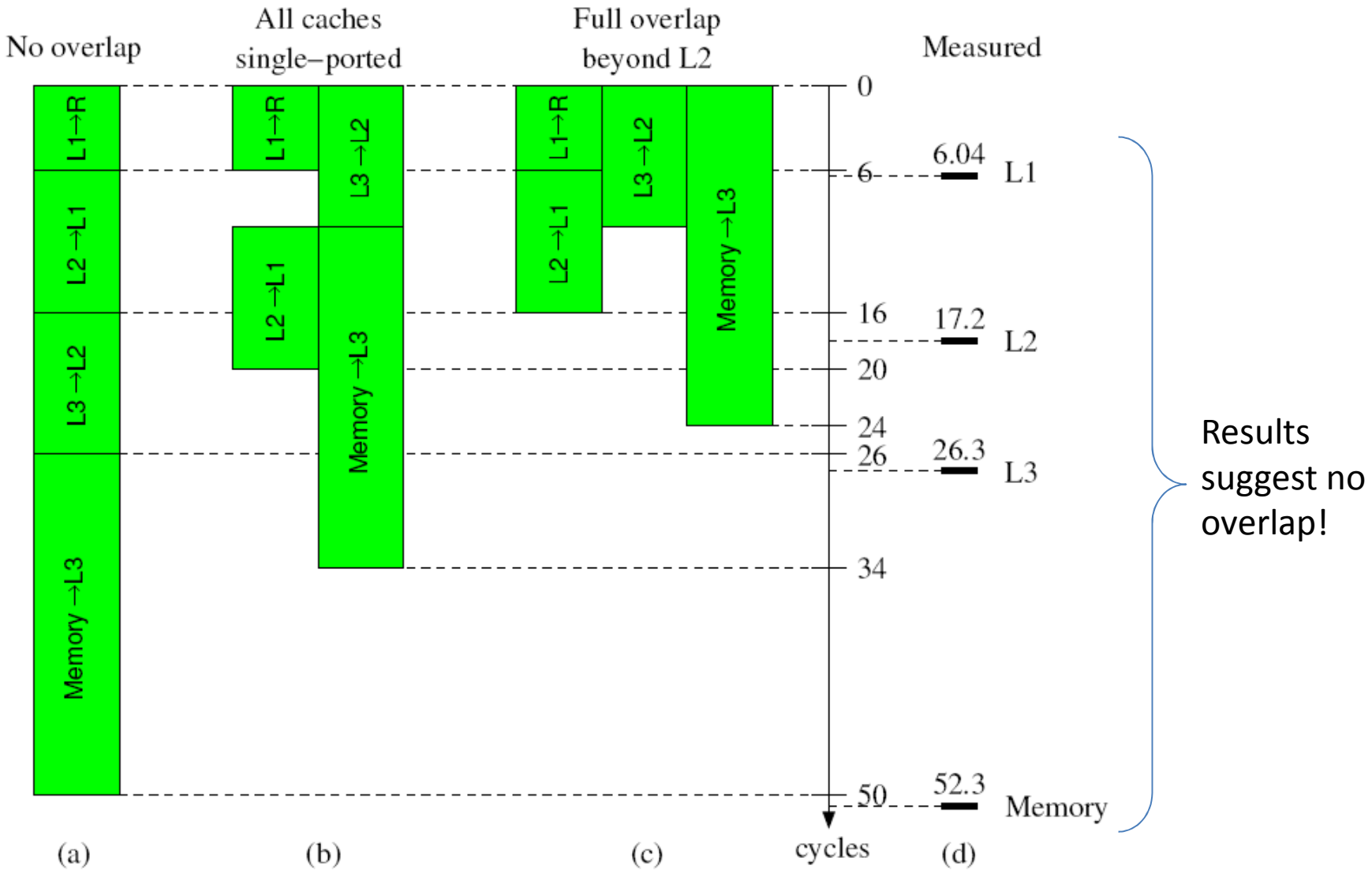
- **Why can a single core often not saturate the memory bus?**
  - Non-overlapping contributions from data transfers and in-cache execution to overall runtime
- **What determines the saturation point?**
  - Important question for energy efficiency
  - Saturation == Bandwidth pressure on relevant bottleneck exhausts the maximum BW cacpacity
- **Requirements for an appropriate multicore performance model**
  - Should predict single-core performance
  - Should predict saturation point

→ ECM (Execution – Cache – Memory) model

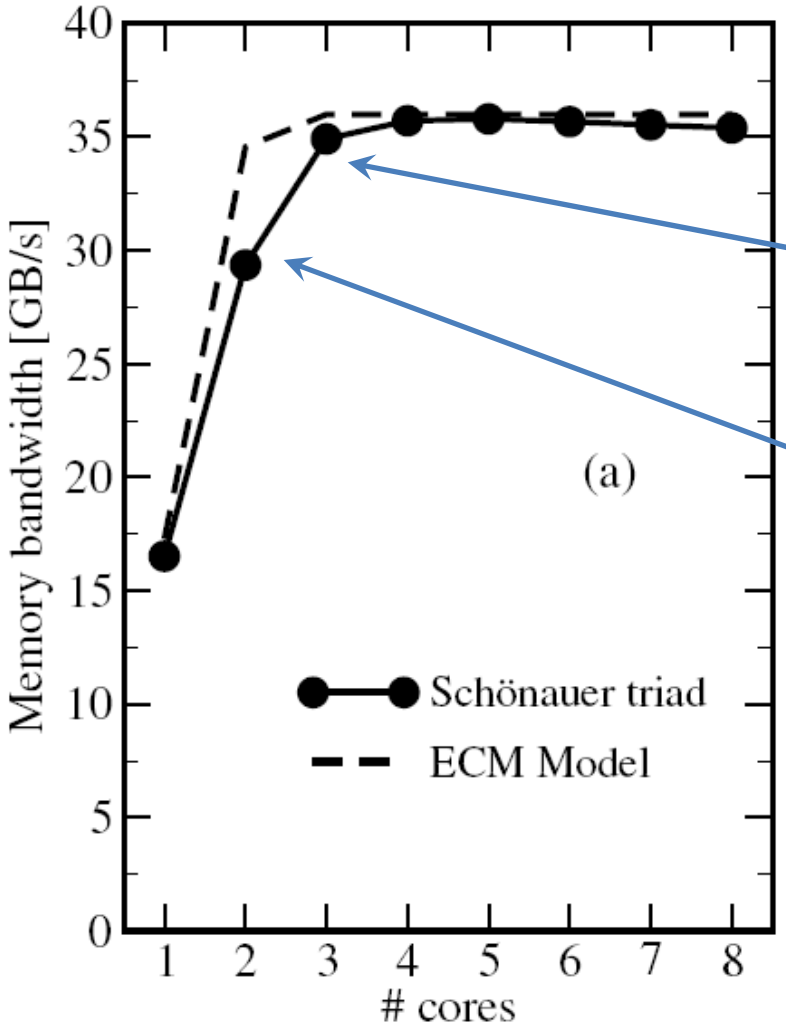
Example: ECM model for Schönauer Vector Triad A(:)=B(:)+C(:)*D(:) on a Sandy Bridge Core with AVX



Full vs. partial vs. no overlap



ECM prediction vs. measurements for `A(:)=B(:)+C(:)*D(:)` on a Sandy Bridge socket (no-overlap assumption)



Model: Scales until saturation sets in

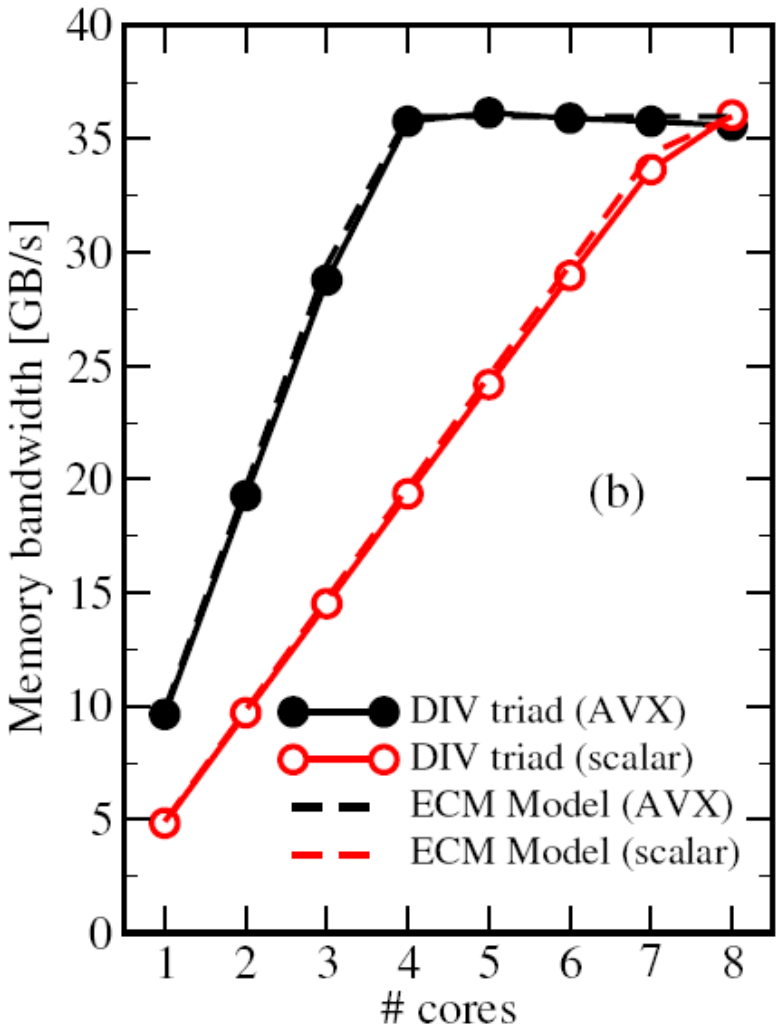
Saturation point (# cores) well predicted

Measurement: scaling not perfect

Caveat: This is specific for this architecture and this benchmark!

Check: Use “overlappable” kernel code

ECM prediction vs. measurements for `A(:)=B(:)+C(:)/D(:)` on a Sandy Bridge socket (full overlap assumption)

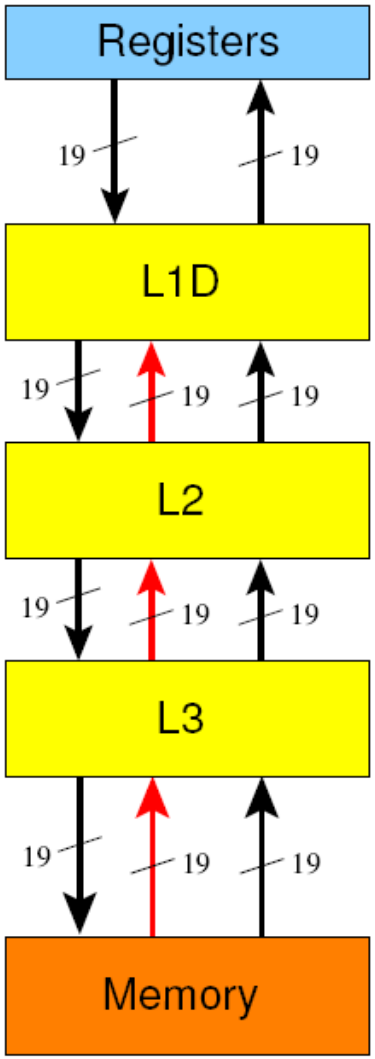


In-core execution is dominated by divide operation

(44 cycles with AVX, 22 scalar)

→ **Almost perfect agreement with ECM model**

Example: Lattice-Boltzmann flow solver



432 cy (IACA throughput)

$3 \cdot 19 \cdot 2$ cy = 114 cy

$3 \cdot 19 \cdot 2$ cy = 114 cy

$(3 \cdot 19 \cdot 64 \cdot 2.7 / 32.3)$ cy = 305 cy (@ 2.7 GHz)
or
$(3 \cdot 19 \cdot 64 \cdot 1.6 / 30.6)$ cy = 191 cy (@ 1.6 GHz)

- **D3Q19 model**
- **Empty channel, 228^3 fluid lattice sites (3.7 GB of memory)**
- **AVX implementation with compiler intrinsics**
- **ECM model input**
  - Core execution from Intel IACA tool
  - Max. memory bandwidth from multi-stream measurements

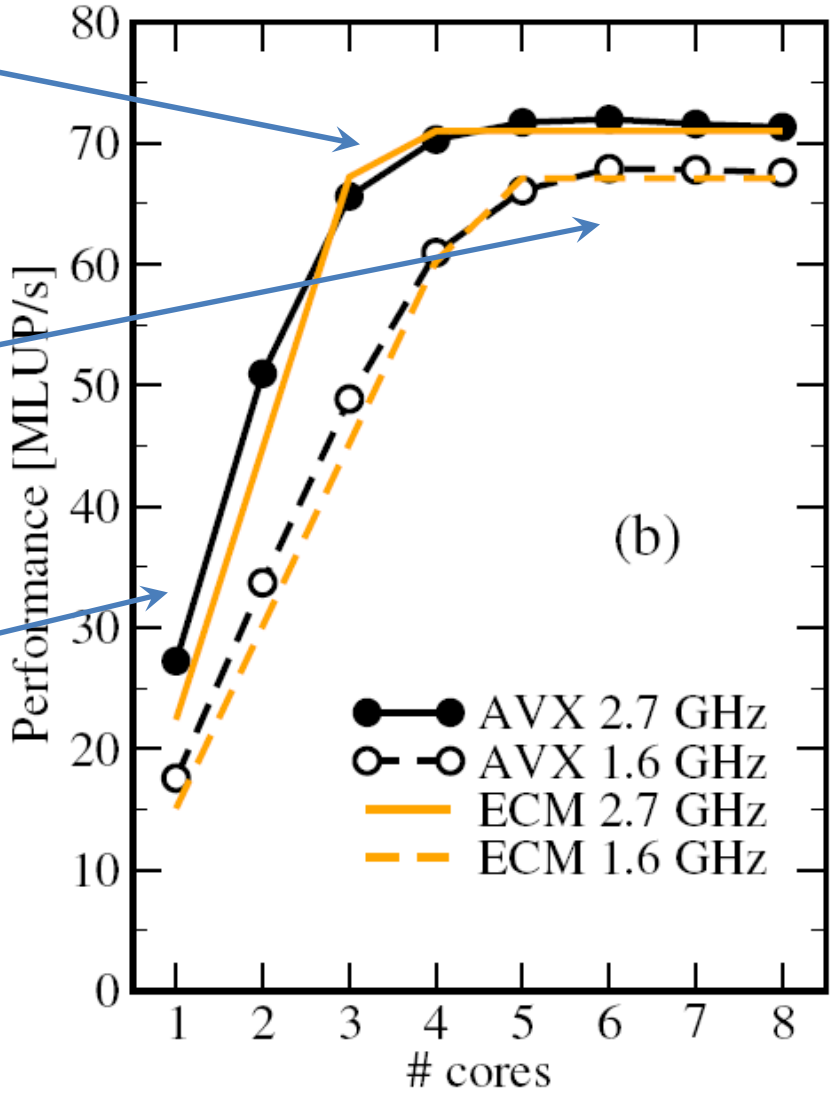
Lattice-Boltzmann solver: ECM (no-overlap) vs. measurements

Saturation point again predicted accurately

Saturation performance matches multi-stream benchmarks

No-overlap assumption seems a little pessimistic

Not all execution is LD and ST



Why the fuss about the saturation point?

Energy consumption!

Assumptions:

1. Power is a quadratic polynomial in the clock frequency
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

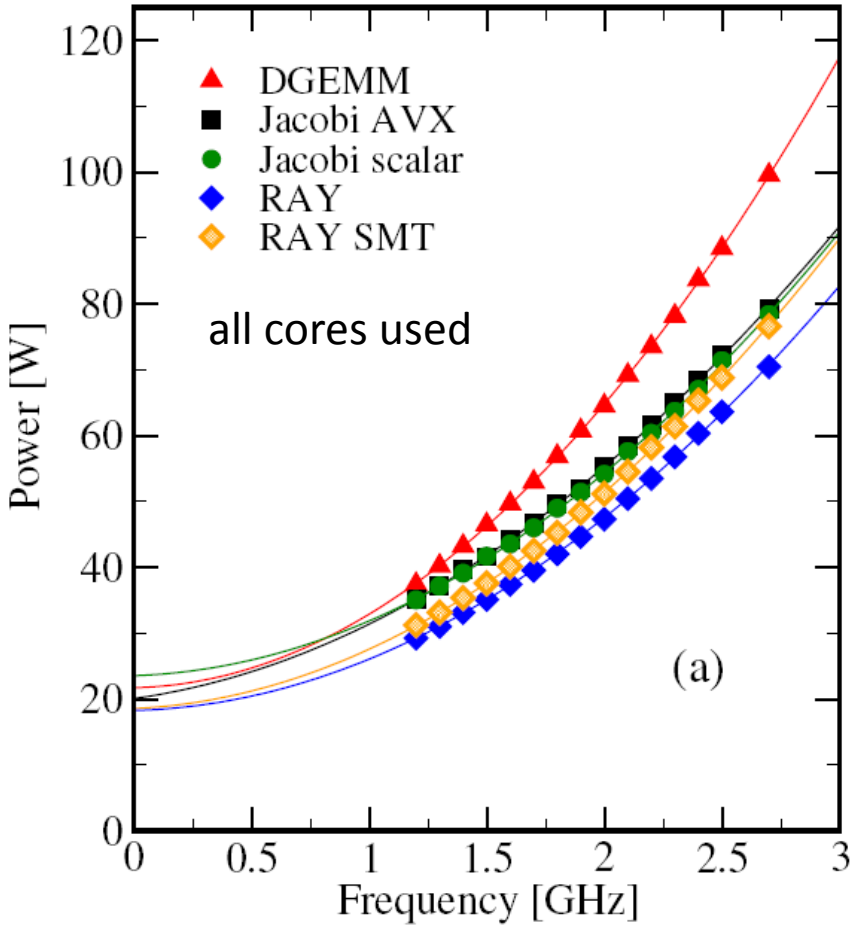
Model:

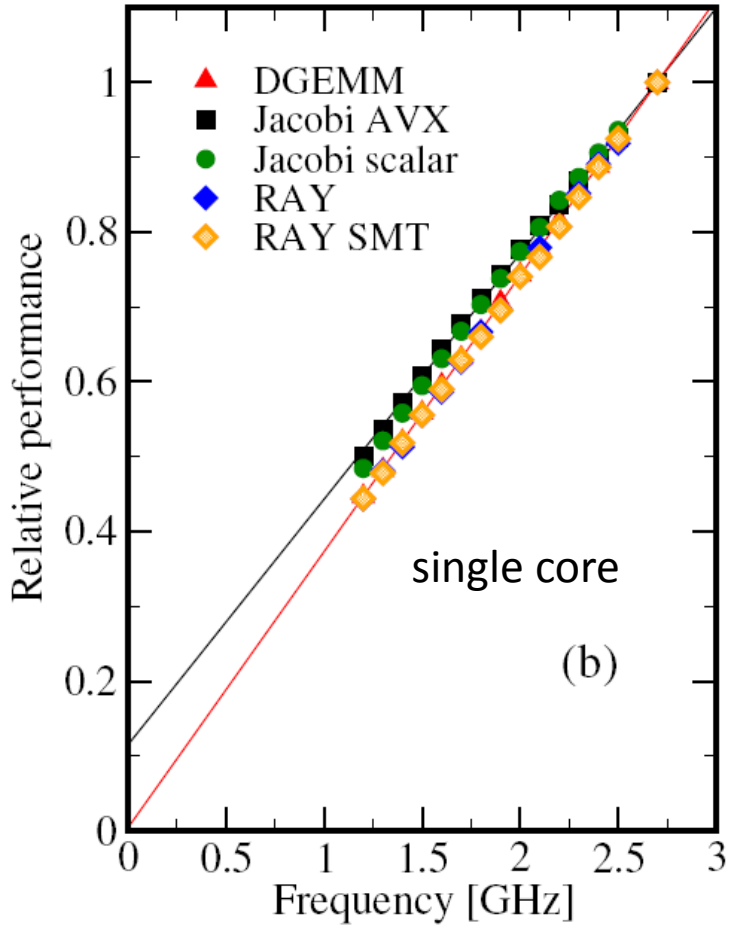
$$E = \frac{W_0 + (W_1 f + W_2 f^2)t}{\min\left((1+\Delta\nu)tP_0, P_{\max}\right)}$$

where $f = (1+\Delta\nu) f_0$

How do we arrive at those assumptions?

Performance and power vs. clock for different applications (SNB):

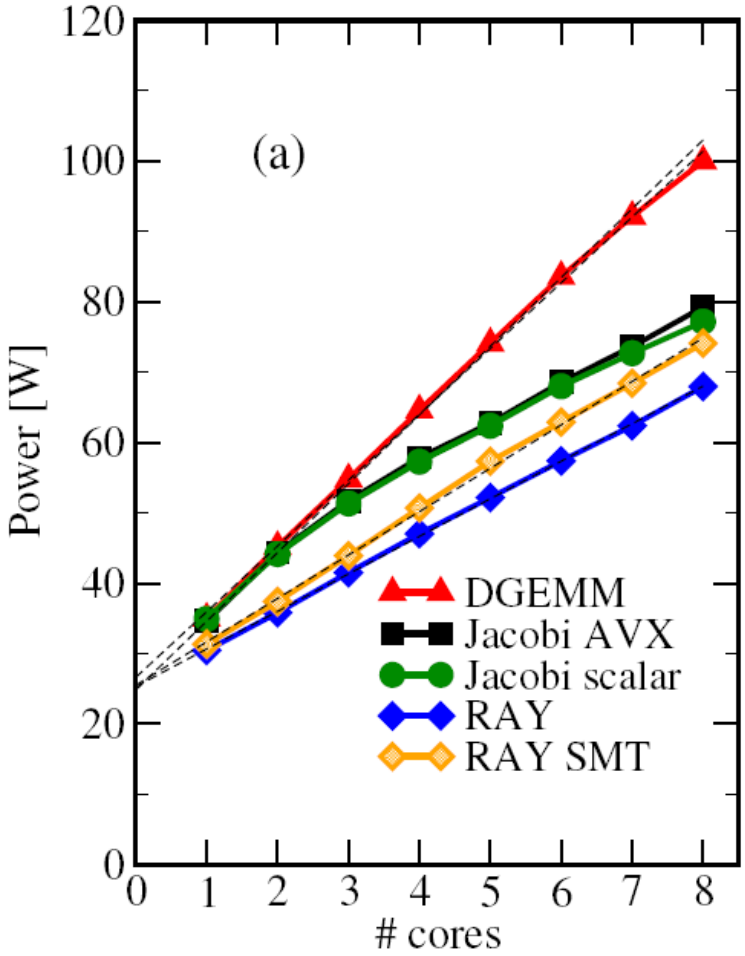


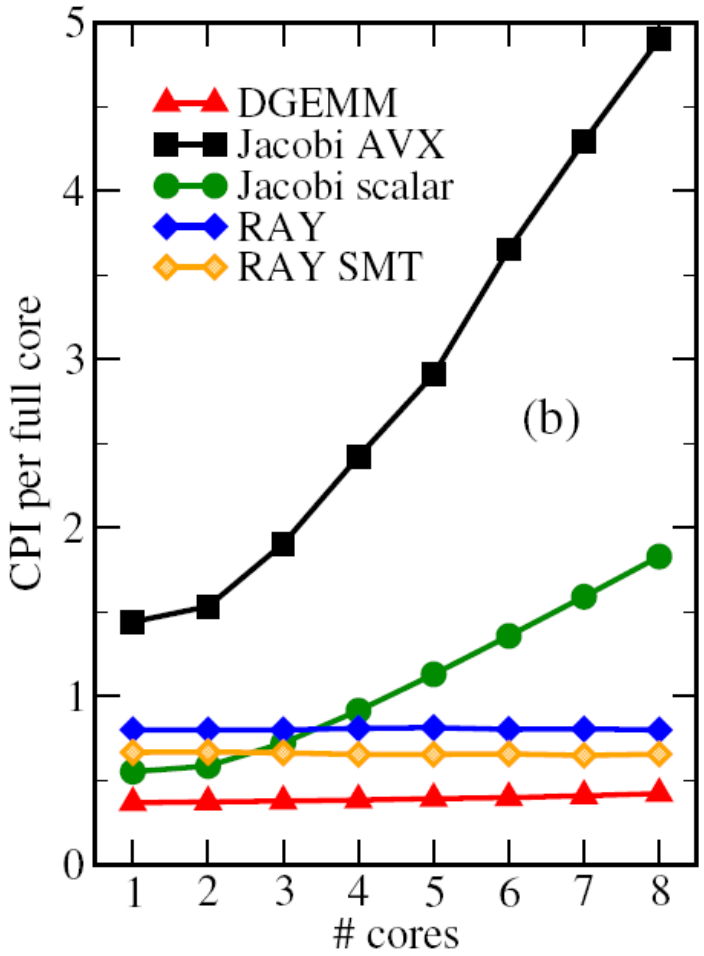


→ **Assumptions (1) and (4)**

How do we arrive at those assumptions?

Power and CPI vs. Number of active cores:





→ **Assumption (2)**

Model predictions

$$E = \frac{W_0 + (W_1 f + W_2 f^2)t}{\min\left((1+\Delta\nu)tP_0, P_{\max}\right)}$$

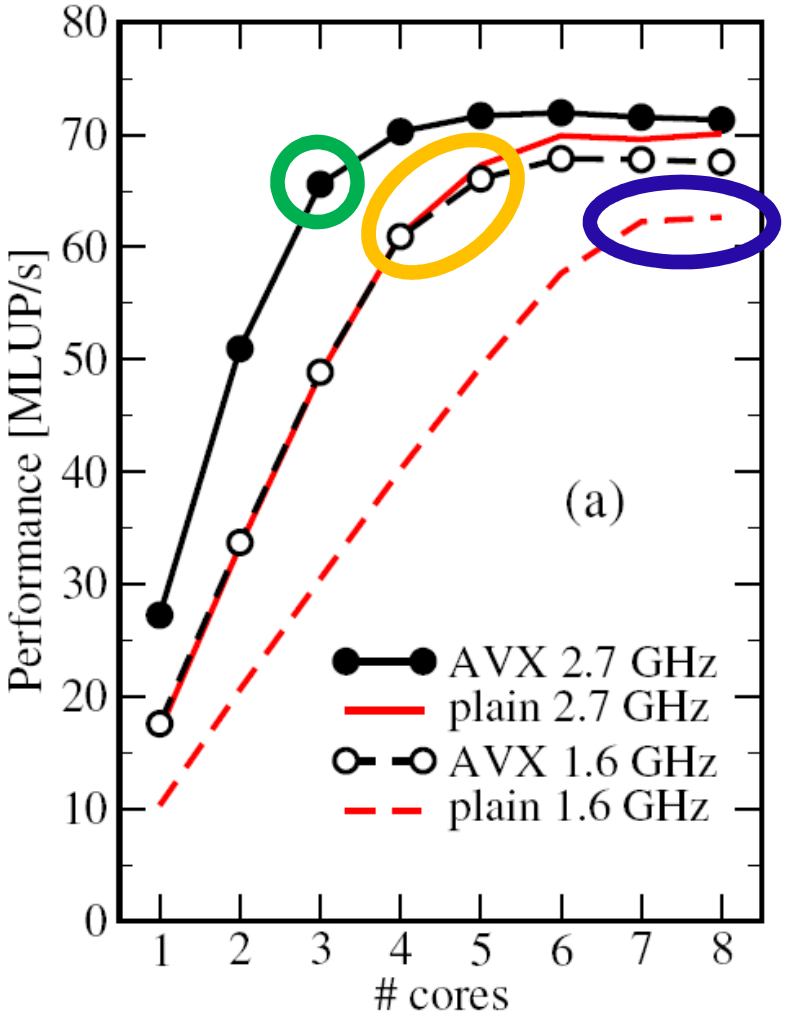
1. **If there is no saturation, use all available cores to minimize *E***
2. **There is an optimal frequency f_{opt} at which *E* is minimal in the non-saturated case, with**

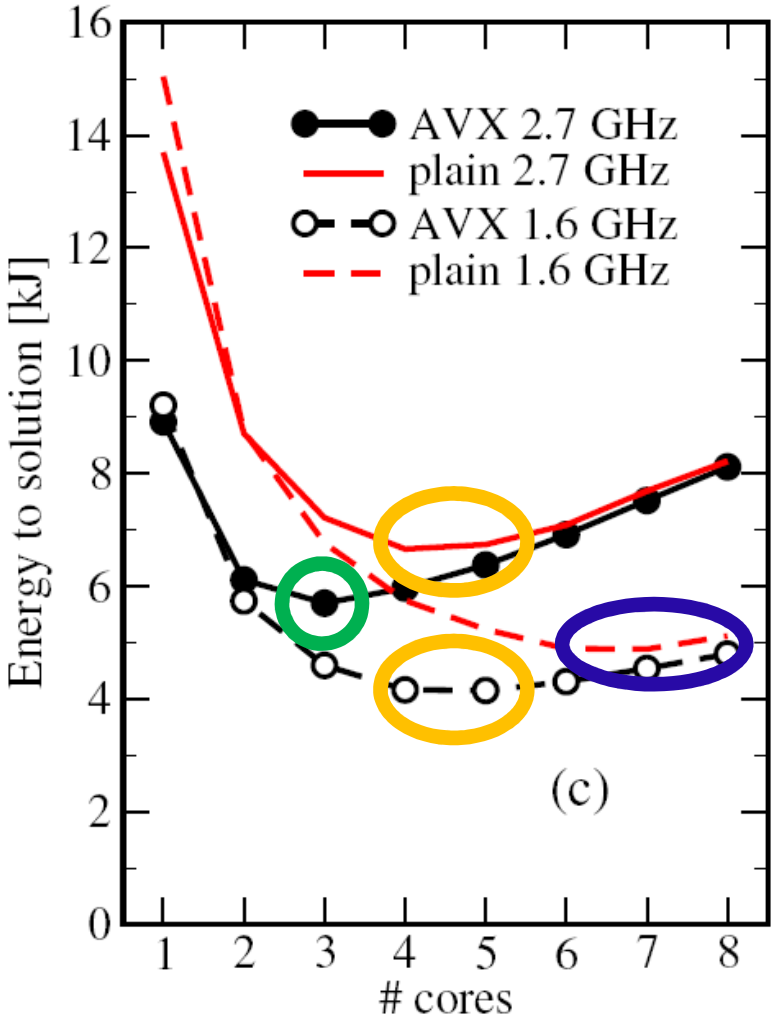
   $f_{\mathrm{opt}} = \sqrt{\frac{W_0}{W_2 t}}$, **hence it depends on the baseline power**

   → **"Clock race to idle" if baseline accommodates whole system!**
3. **If there is saturation, *E* is minimal at the saturation point**
4. **If there is saturation, absolute minimum *E* is reached if the saturation point is at the number of available cores**
5. **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

Validation using the lattice-Boltzmann example

Performance and energy to solution vs. cores on SNB





Conclusions

- **Performance Engineering == Performance Modeling with “bells and whistles”**
- **PE is more than just finding out about hot spots and trying to change “something in the code” to make it faster. It is about insight into the interaction of hardware and software!**
- **PM works out best if it does not work ☺**
- **Saturation effects are ubiquitous; understanding them gives us opportunity to**
  - Find out about optimization opportunities
  - Save energy
- **Simple models work best. Do not try to complicate things unless it is really necessary!**

Make it as simple as possible, but not simpler.

Albert Einstein

Bundesministerium
für Bildung
und Forschung

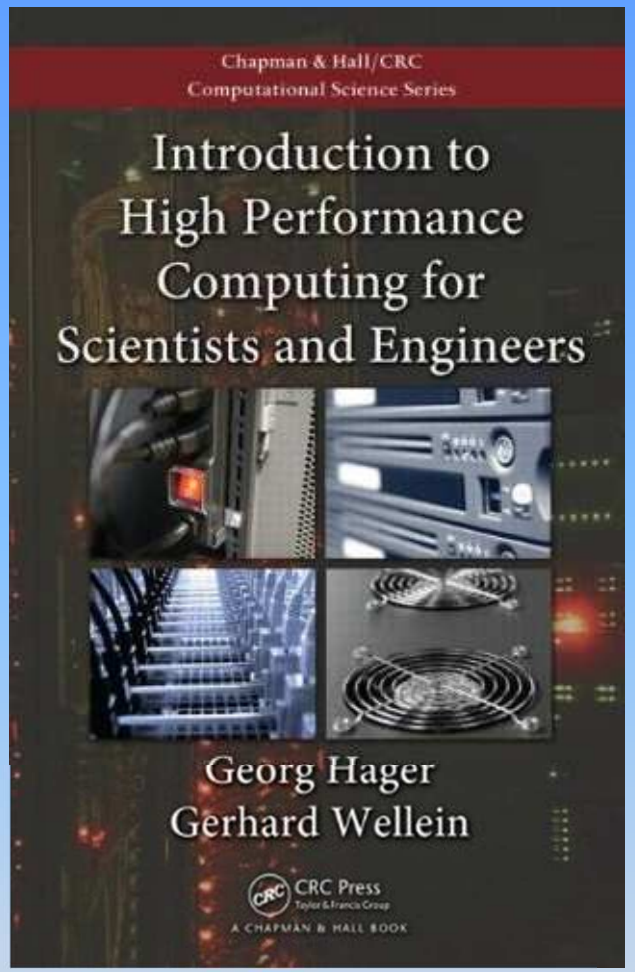
hpcADD

Thank you.

OMI4papps

New HPC textbook

Georg Hager and Gerhard Wellein:
Introduction to High Performance Computing for Scientists and Engineers



CRC Press, ISBN 978-1439811924
356 pages
July 2010

"Georg Hager and Gerhard Wellein have developed a very approachable introduction to high performance computing for scientists and engineers. Their style and descriptions are easy to read and follow. … This book presents a balanced treatment of the theory, technology, architecture, and software for modern high performance computers and the use of high performance computing systems. The focus on scientific and engineering problems makes it both educational and unique. I highly recommend this timely book for scientists and engineers. I believe it will benefit many readers and provide a fine reference."

— From the Foreword by Jack Dongarra, University of Tennessee, Knoxville, USA

References

- J. Treibig and G. Hager: *Introducing a Performance Model for Bandwidth-Limited Loop Kernels.* Proceedings of the Workshop "Memory issues on Multi- and Manycore Platforms" at PPAM 2009, the 8th International Conference on Parallel Processing and Applied Mathematics, Wroclaw, Poland, September 13-16, 2009. DOI: 10.1007/978-3-642-14390-8_64

- G. Schubert, H. Fehske, G. Hager, and G. Wellein: *Hybrid-parallel sparse matrix-vector multiplication with explicit communication overlap on current multicore-based systems.* Parallel Processing Letters **21**(3), 339-358 (2011). DOI: 10.1142/S0129626411000254

- 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