ERLANGEN REGIONAL
COMPUTING CENTER

http://goo.gl/Hwx6Nw

Node-Level Performance Engineering

Georg Hager, Jan Eitzinger, Gerhard Wellein
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg

Two-day short course
Jülich Supercomputing Centre
2016-04-28/29

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Course Schedule

http://goo.gl/Hwx6Nw

Day 1	
Computer architecture	
Tools: topology & affinity	
Microbenchmarking for architectural exploration	
Demo 1	
Roofline model: basics	
Tools: hardware performance counters	
Roofline case study: dense matrix-vector multiplication	
Day 2	
Roofline case study: sparse matrix-vector multiplication	
Roofline case study: Jacobi smoother	
Demo 2	
Optimal use of parallel resources: SIMD, ccNUMA, SMT	
ECM Model and/or energy-efficient code execution	
Demo 3	
Optional: Lattice-Boltzmann / SpMVM on heterogeneous systems	
Optional: Pattern-based performance engineering	

Cycle gymnastics

http://goo.gl/Hwx6Nw

- 1 cycle = smallest unit of time on a CPU ("heartbeat")
  - Clock speed of typical CPU: 3.0 Gcy/s (or GHz)
- Basic unit of work: Floating-point operation (Flop)
  - Typical peak performance of 8-core CPU: P_{peak} = 192 Gflop/s
  - How many Flops per cycle per core is that? $\frac{192 \cdot 10^9 \frac{Flops}{s}}{8\ cores \cdot 3.0 \cdot 10^9 \frac{cy}{s}} = 8 \frac{Flops}{cy \cdot core}$
  - Typical duration of a double precision multiply: 5 cycles
    - How much time is that? $\frac{5\ cy}{3.0 \cdot 10^9 \frac{cy}{s}} = 1.67 \cdot 10^{-9} s = 1.67\ ns$
- Basic unit of traffic: Byte
- Unit of bandwidth: Bytes/s
  - Typical memory bandwidth: 48 Gbytes/s = $4.8 \cdot 10^{10}$ Bytes/s
  - How many bytes per cycle is that? $\frac{48 \cdot 10^9 \frac{Bytes}{s}}{3.0 \cdot 10^9 \frac{cy}{s}} = 16 \frac{Bytes}{cy}$

PRELUDE:
SCALABILITY 4 THE WIN!

How to ask the right questions

A conversation

From a student seminar on “Efficient programming of modern multi- and manycore processors”

Student: I have implemented this algorithm on the GPGPU, and it solves a system with 26546 unknowns in 0.12 seconds, so it is really fast.

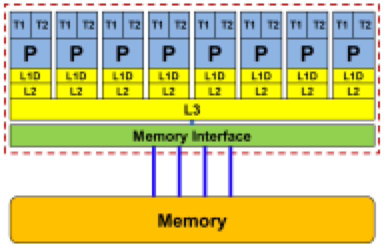
Me: What makes you think that 0.12 seconds is fast?

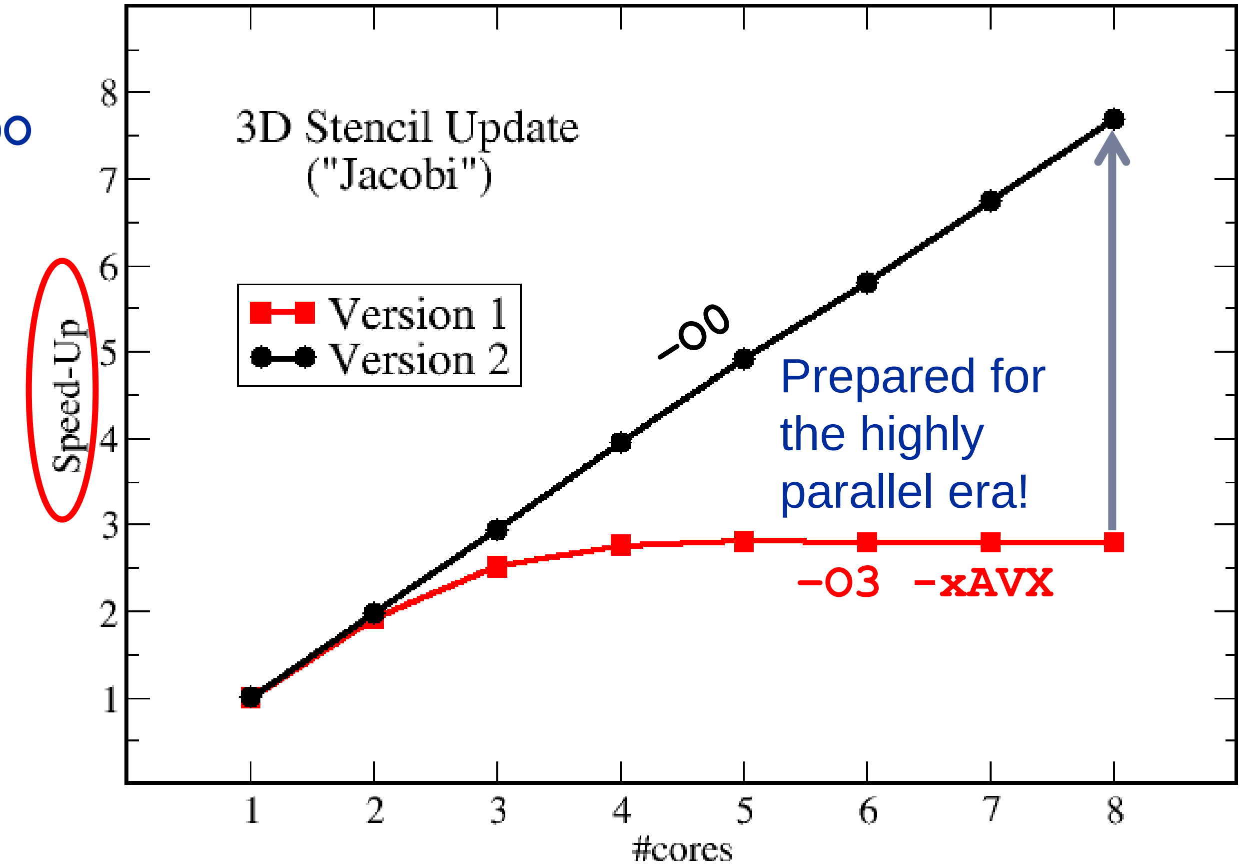
Student: It is fast because my baseline C++ code on the CPU is about 20 times slower.

Scalability Myth: Code scalability is the key issue

```
!$OMP PARALLEL DO
do k = 1 , Nk
   do j = 1 , Nj; do i = 1 , Ni
       y(i,j,k)= b*(  x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                      x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
     enddo; enddo
enddo
!$OMP END PARALLEL DO
```

Changing only a the compile options makes this code scalable on an 8-core chip



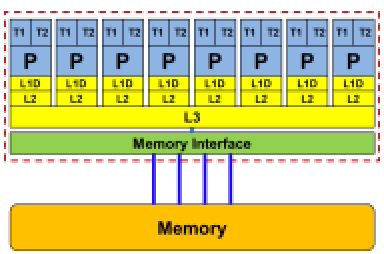


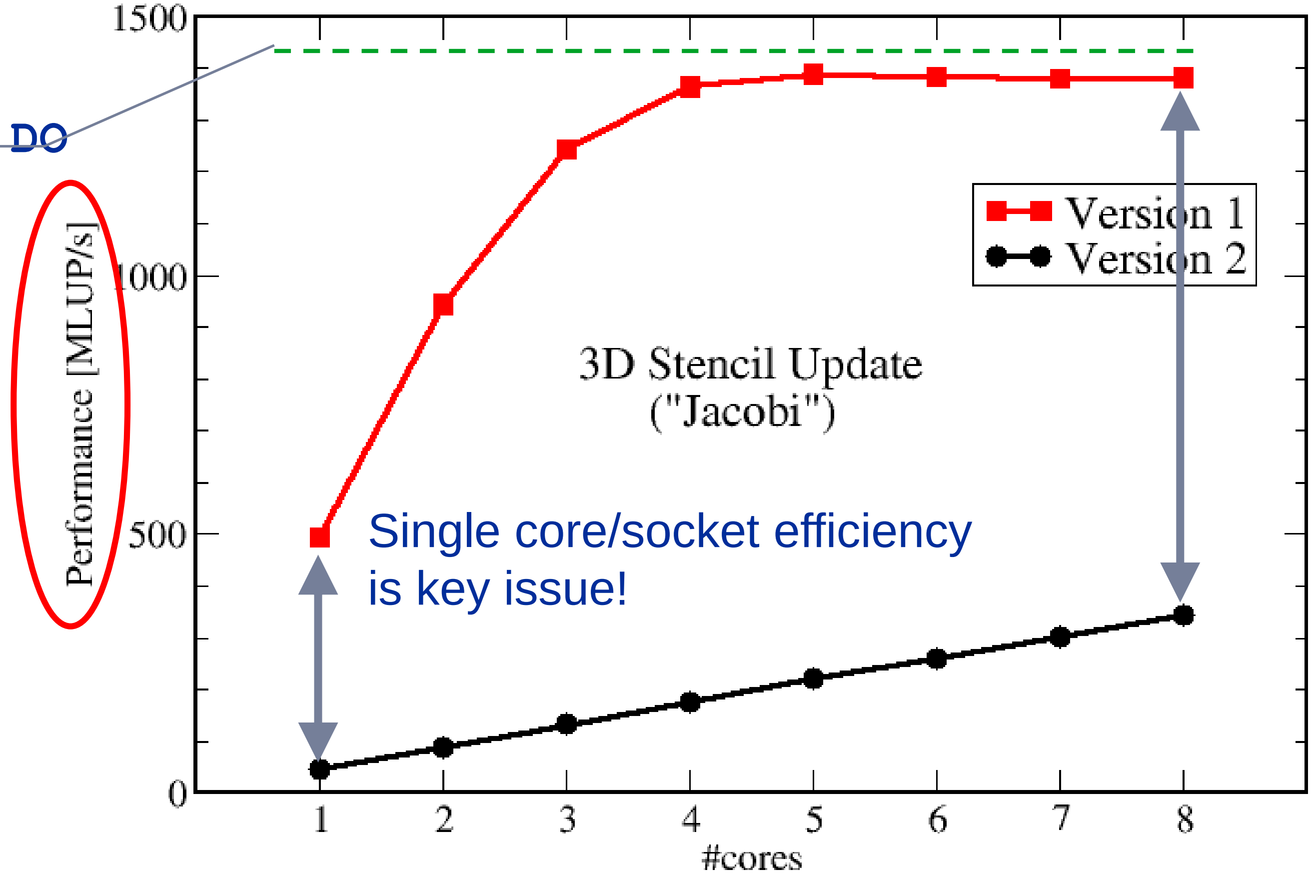
Scalability Myth: Code scalability is the key issue

```
!$OMP PARALLEL DO
do k = 1 , Nk
  do j = 1 , Nj; do i = 1 , Ni
     y(i,j,k)= b*(  x(i-1,j,k)+ x(i+1,j,k)+ x(i,j-1,k)+
                    x(i,j+1,k)+ x(i,j,k-1)+ x(i,j,k+1))
    enddo; enddo
enddo
```

L DO

Upper limit from simple performance model:
35 GB/s & 24 Byte/update





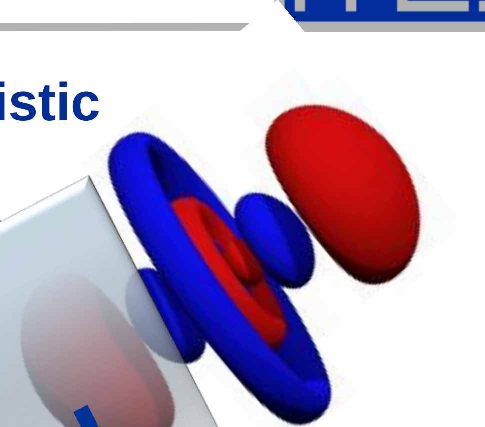
How model-building works: Physics

Newtonian mechanics

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

Fails @ small scales!

Nonrelativistic quantum mechanics



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

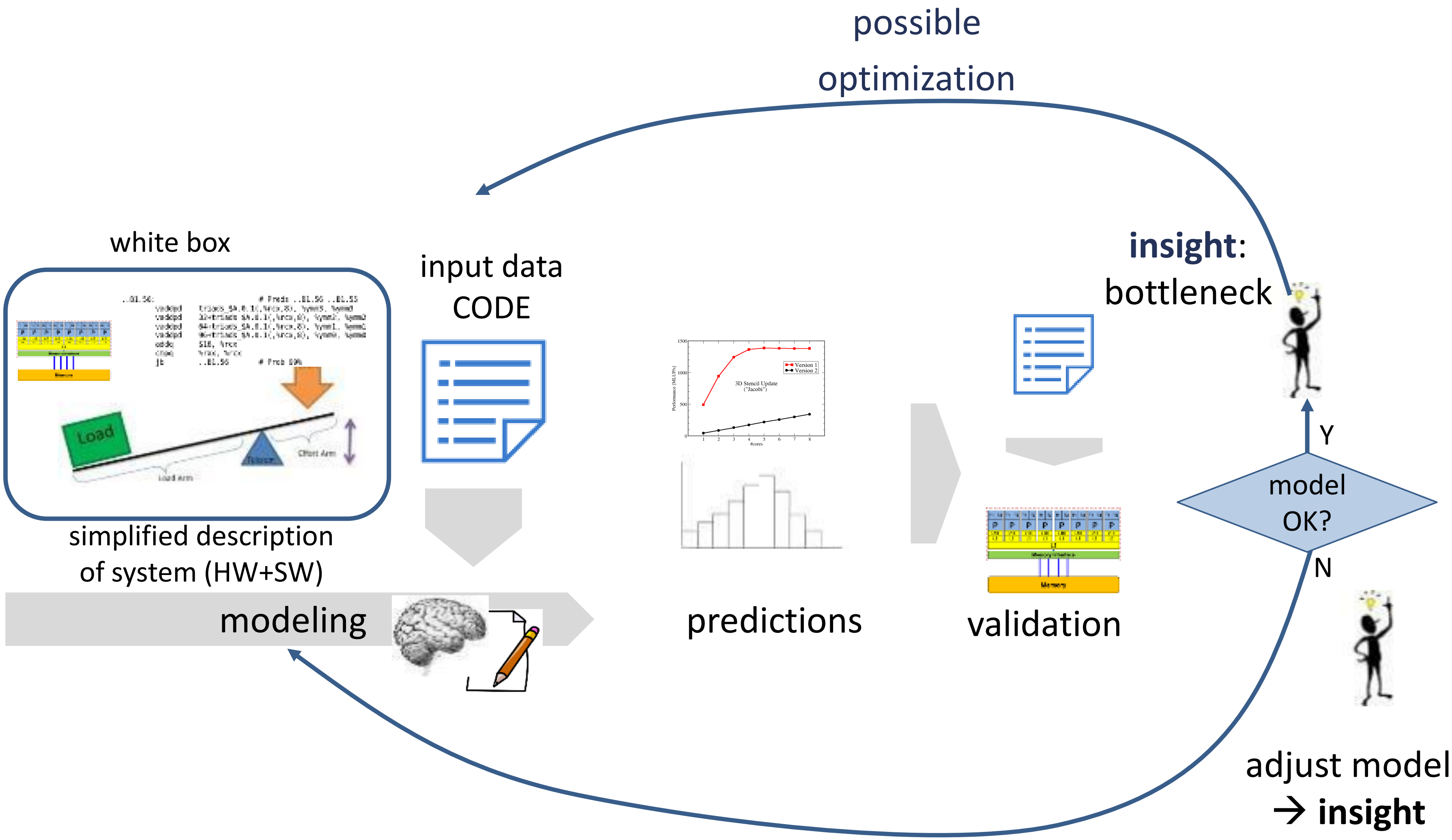
Fails @ even smaller scales!

If a model fails, we learn something!

Relativistic quantum field theory

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

Performance Engineering – no black boxes!



Questions to ask in high performance computing

- **Do I understand the performance behavior of my code?**
  - Does the performance match a model I have made?
- **What is the optimal performance for my code on a given machine?**
  - High Performance Computing == Computing at the bottleneck
- **Can I change my code so that the "optimal performance" gets higher?**
  - Circumventing/ameliorating the impact of the bottleneck
- **My model does not work – what's wrong?**
  - This is the good case, because you learn something
  - Performance monitoring / microbenchmarking may help clear up the situation
- **Use your brain! Tools may help, but you do the thinking.**