



Erlangen Regional
Computing Center



Performance Engineering

Fundamentals



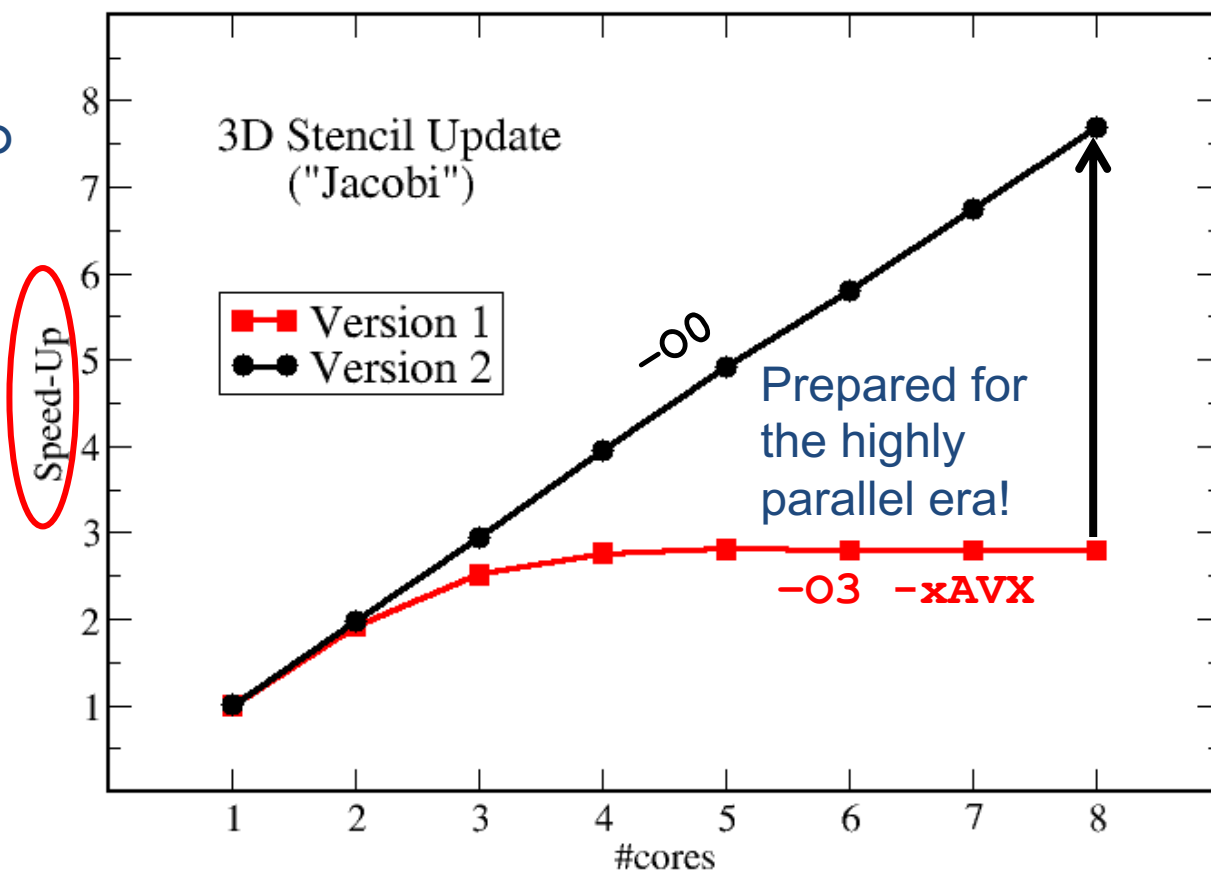
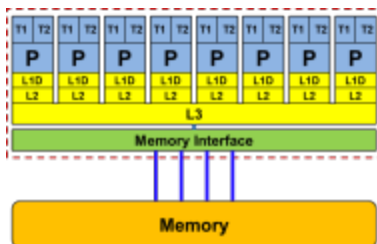
Factors influencing performance:

- **Implementation**
Programming language, optimization techniques
- **Code generation**
Compiler used and compiler options
- **System configuration**
BIOS and OS settings, configurable Options
- **Execution environment**
Thread affinity and resource allocation

```

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

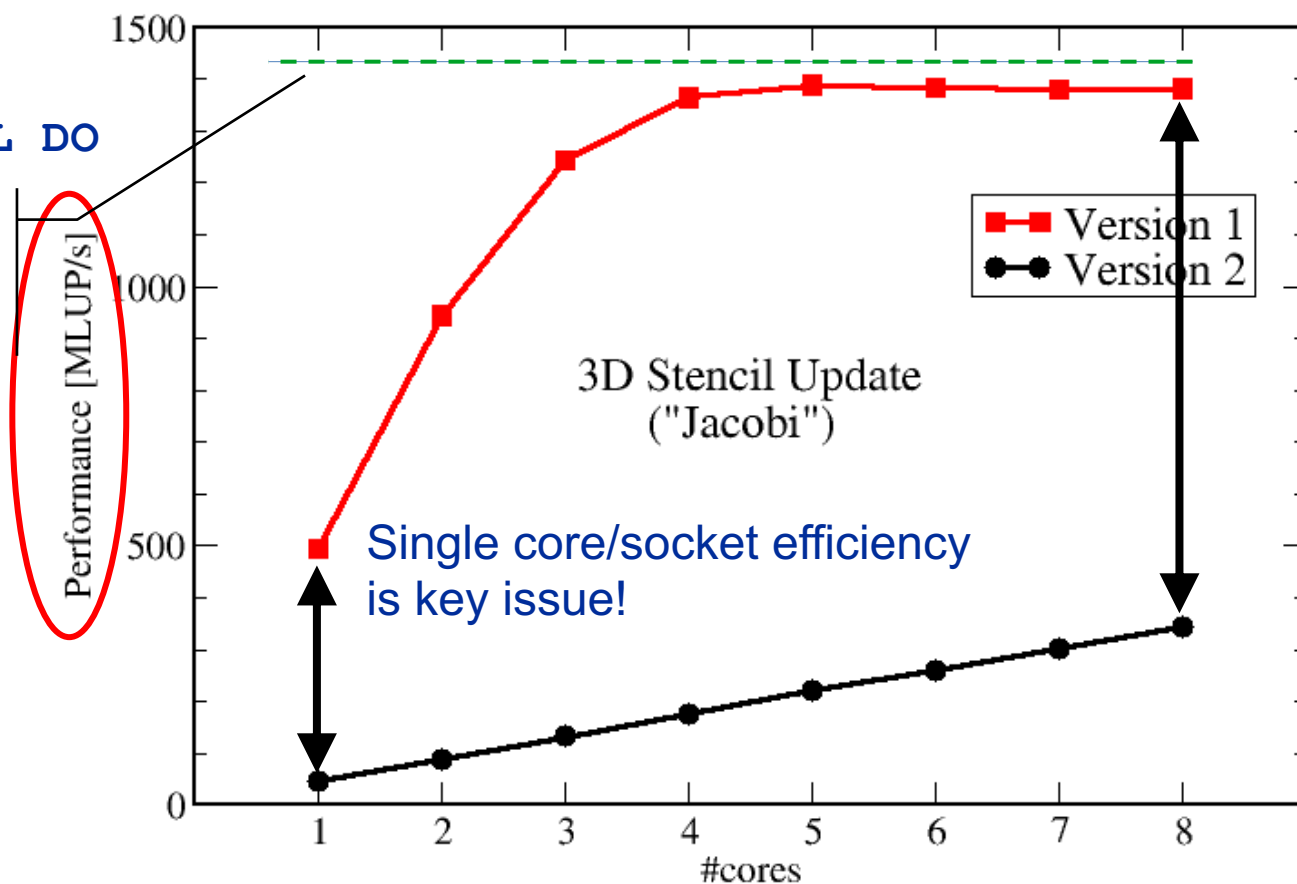
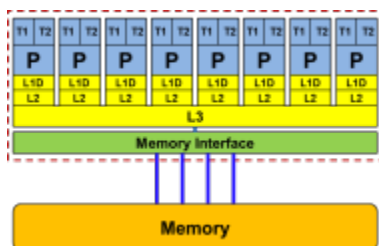
Changing only the compile options makes this code scalable on an 8-core chip



```

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

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



- Metrics often used in PE publications:
 - MFlops/s
 - Cache miss rate
 - Speedup

Time to solution is all that matters!

- Every activity adds a **runtime contribution**
- Simplest case: all runtime contributions accumulate to **time to solution**
- Due to **concurrency** runtime contributions can **overlap** with each other
- **Critical path** is the series of runtime contributions, that do not overlap and form the total runtime

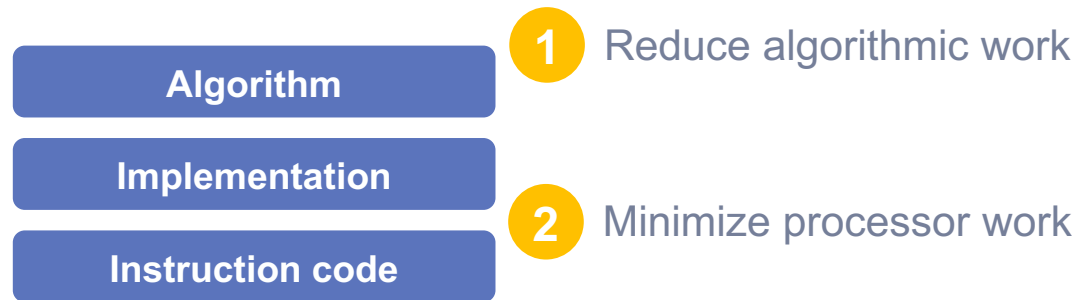
Anything that takes time is only relevant for optimization if it appears on the **critical path!**

Factors limiting speedup:

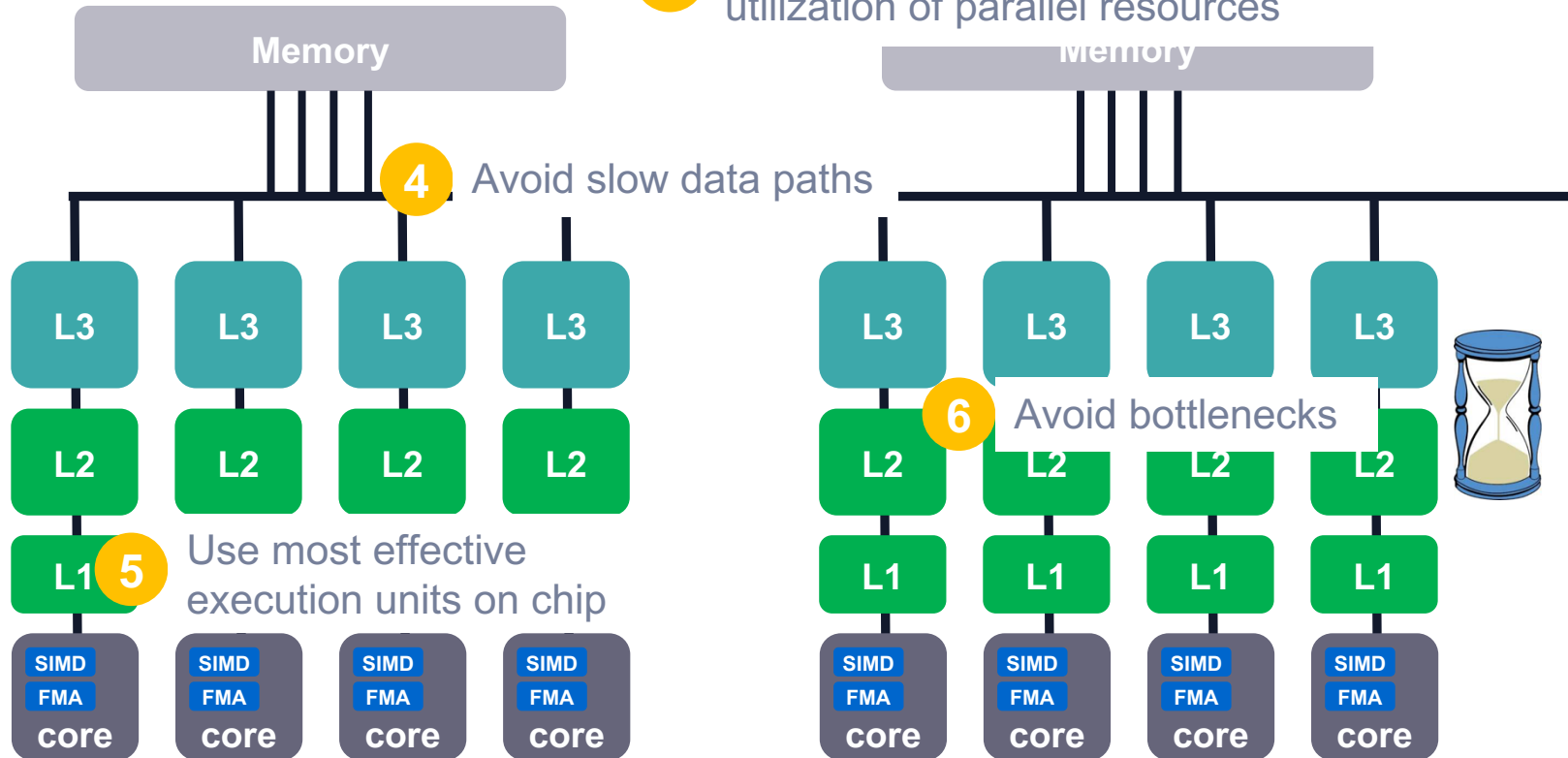
- **Load balancing:** Can work be perfectly distributed on parallel execution resources?
- **Dependencies:** Are there dependencies that add to the critical path?
- **Communication overhead:** Is there any data transfer time that adds to the critical path?
- **Instruction overhead:** Additional work that is required to implement an efficient parallel execution.

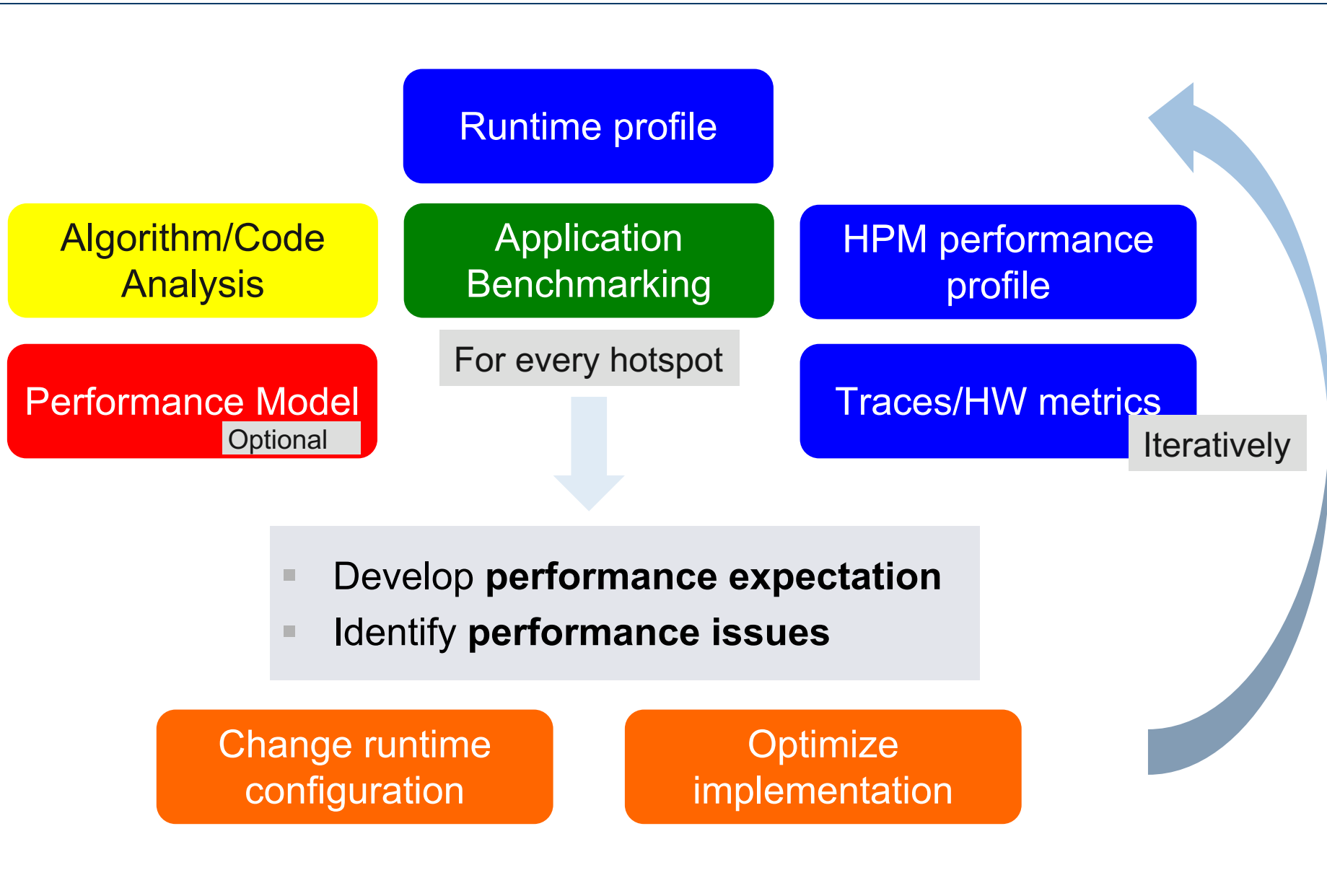
All but the last one (which adds to the processor work) are experienced as additional **waiting times**.

Optimizing code: The big picture



3 Distribute work and data for optimal utilization of parallel resources





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.

Do I understand the performance behavior of my code?

What is (or should be) the principal **hardware bottleneck** (design limit)?

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 profiling / microbenchmarking may help clear up the situation

Use your brain! Tools may help, but you do the thinking.