

Node-Level Performance Engineering

Georg Hager, Gerhard Wellein, Jan Eitzinger

Three-day online tutorial

VSC, TU Wien

March 10-12, 2021

<http://tiny.cc/NLPE-VSC>

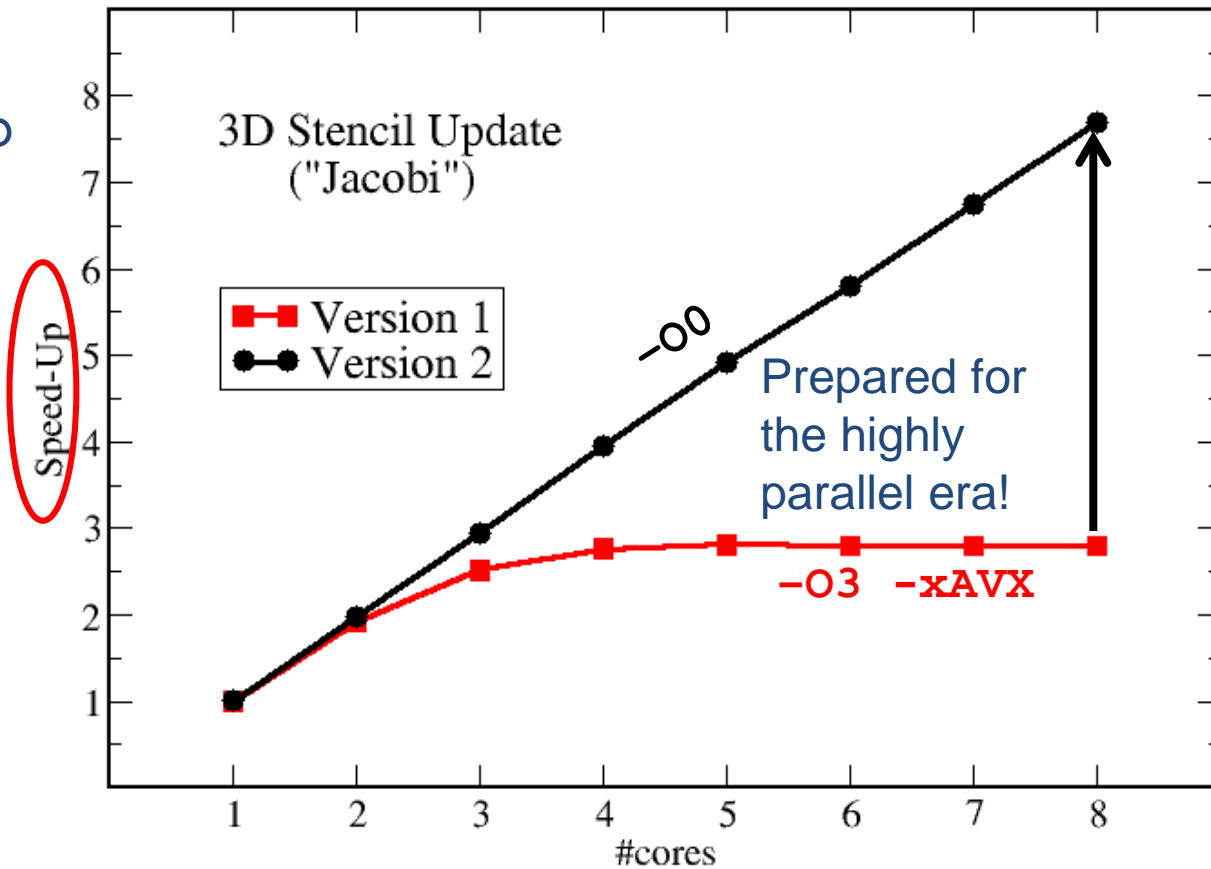
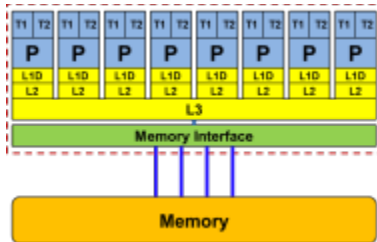
- 1 cycle = smallest unit of time on a CPU (“heartbeat”)
 - Clock speed of typical CPU: **2.4 Gcy/s (or GHz)**
- Basic unit of work: Floating-point operation (Flop)
 - Typical peak performance of 20-core CPU: **$P_{\text{peak}} = 768 \text{ Gflop/s}$**
 - How many Flops per cycle per core is that? $\frac{768 \cdot 10^9 \frac{\text{Flops}}{\text{s}}}{20 \text{ cores} \cdot 2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 16 \frac{\text{Flops}}{\text{cy} \cdot \text{core}}$
 - Typical **duration** of a double precision **multiply: 5 cycles**
 - › How much time is that? $\frac{5 \text{ cy}}{2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 2.08 \cdot 10^{-9} \text{ s} = 2.08 \text{ ns}$
- Basic unit of traffic: **Byte**
- Unit of bandwidth: **Bytes/s**
 - Typical memory bandwidth: **$62 \text{ Gbytes/s} = 6.2 \cdot 10^{10} \text{ Bytes/s}$**
 - How many bytes per cycle is that? $\frac{62 \cdot 10^9 \frac{\text{Bytes}}{\text{s}}}{2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 25.8 \frac{\text{Bytes}}{\text{cy}}$

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
    
```

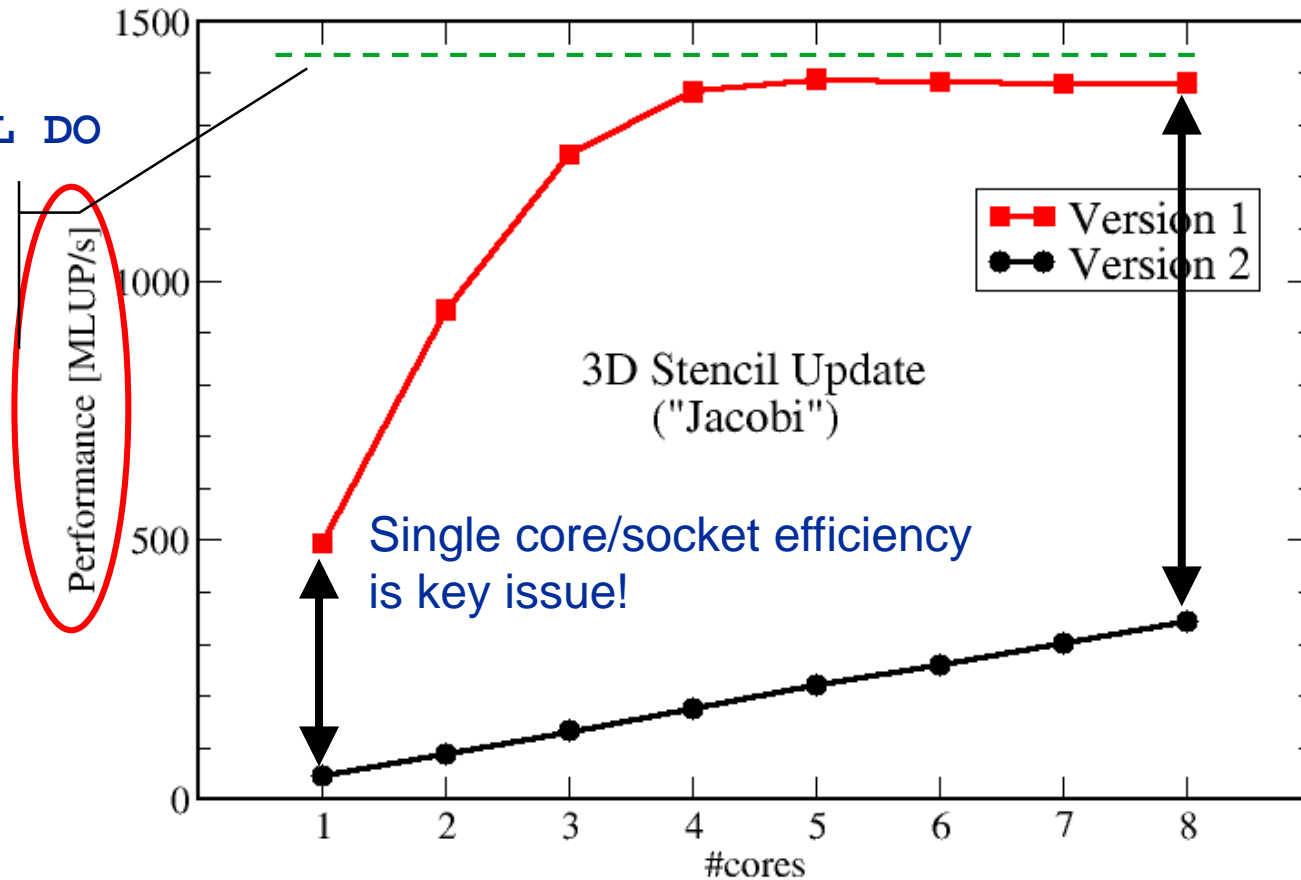
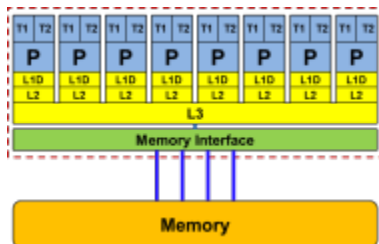
Changing only the compile options makes this code scalable on an 8-core chip



```

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

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



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.