

# Node-Level Performance Engineering

## Introduction



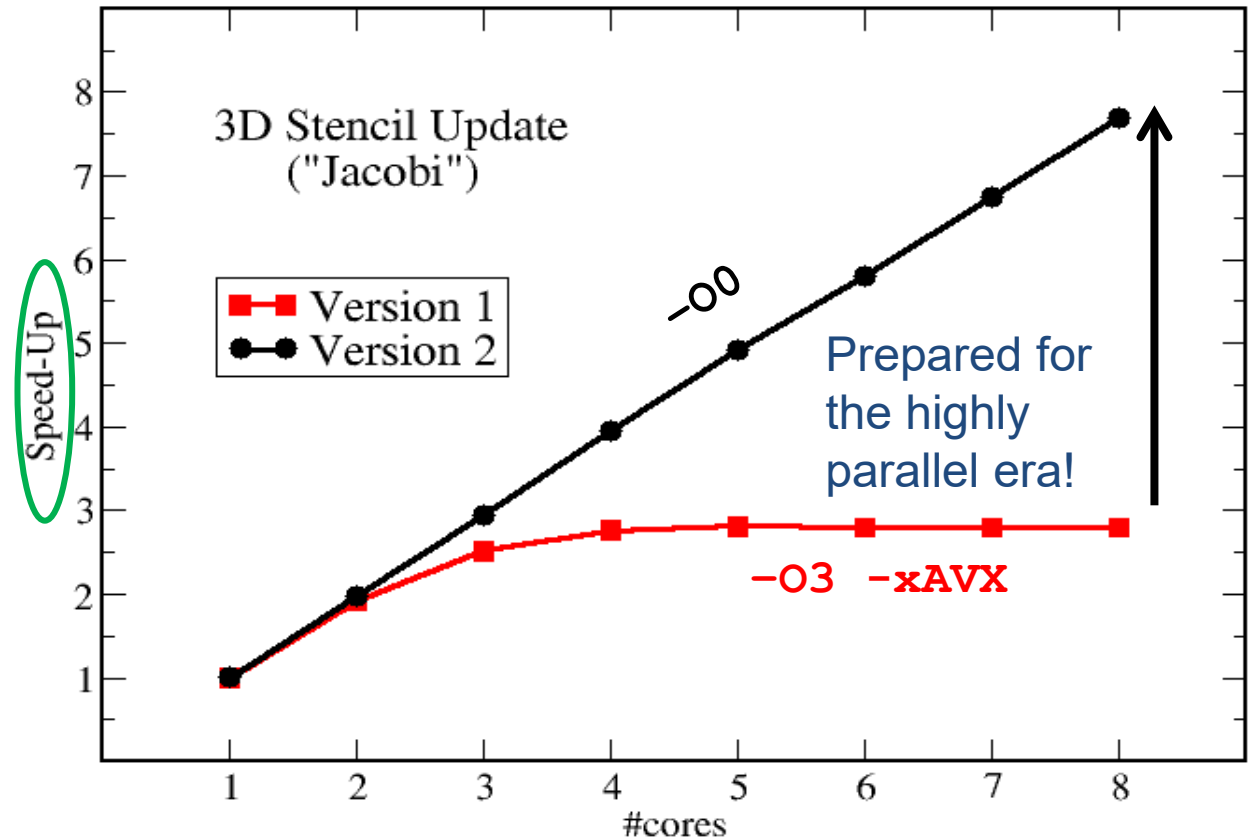
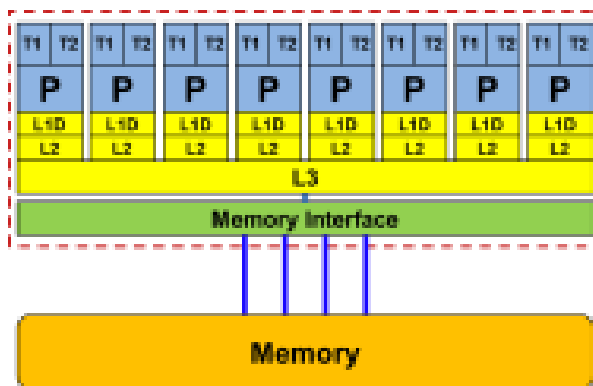
# Dimensional gymnastics

- 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:  $P_{\text{peak}} = 1536 \text{ Gflop/s} = 1.536 \text{ Tflop/s}$
  - How many Flops per cycle per core is that?  $\frac{1536 \cdot 10^9 \frac{\text{Flops}}{\text{s}}}{20 \text{ cores} \cdot 2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 32 \frac{\text{Flops}}{\text{cy} \cdot \text{core}}$
  - Typical duration of a double precision multiply: 4 cycles
    - › How much time is that?  $\frac{4 \text{ cy}}{2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 1.67 \cdot 10^{-9} \text{ s} = 1.67 \text{ ns}$
- Basic unit of traffic: Byte
- Unit of bandwidth: Bytes/s
  - Typical memory bandwidth: 160 Gbytes/s =  $1.6 \cdot 10^{11} \text{ Bytes/s}$
  - How many bytes per cycle is that?  $\frac{160 \cdot 10^9 \frac{\text{Bytes}}{\text{s}}}{2.4 \cdot 10^9 \frac{\text{cy}}{\text{s}}} = 67 \frac{\text{Bytes}}{\text{cy}}$

# 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
```

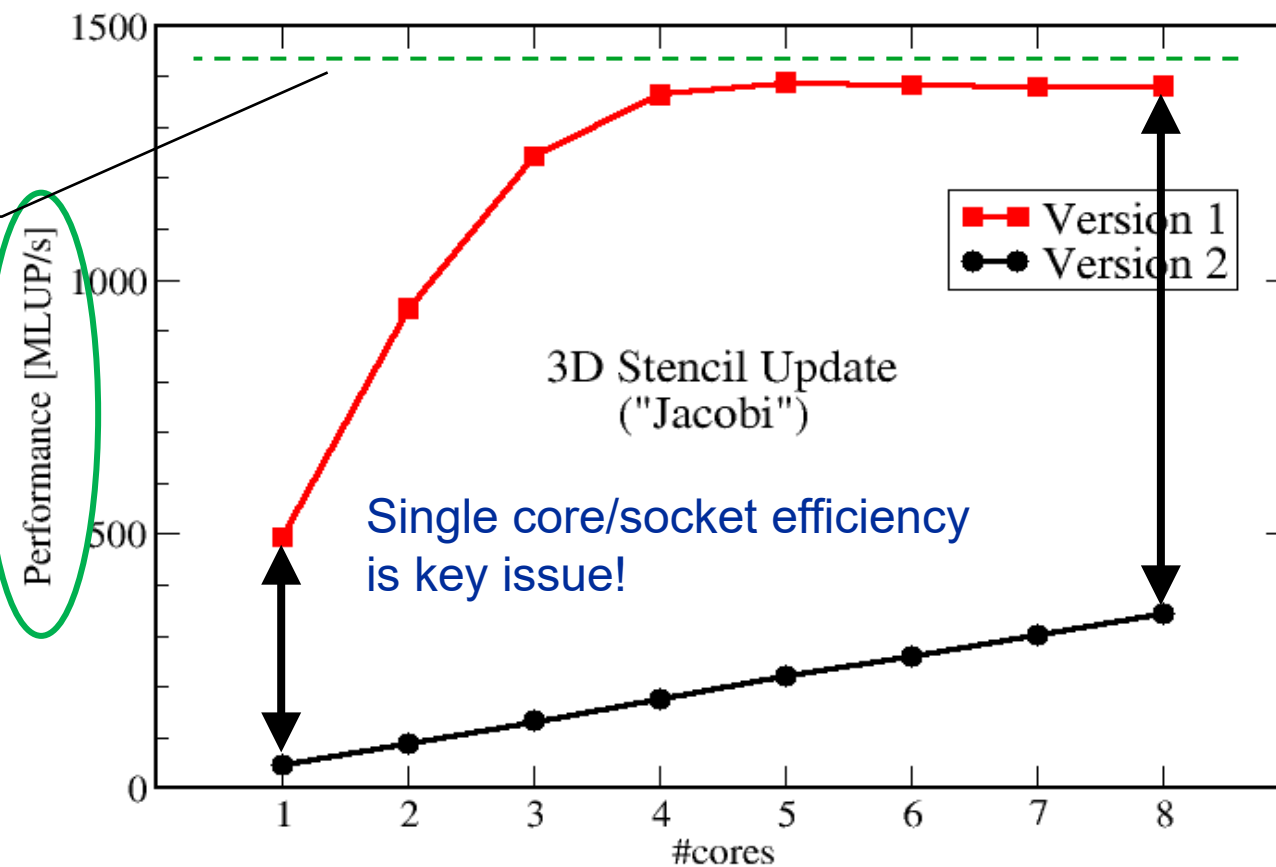
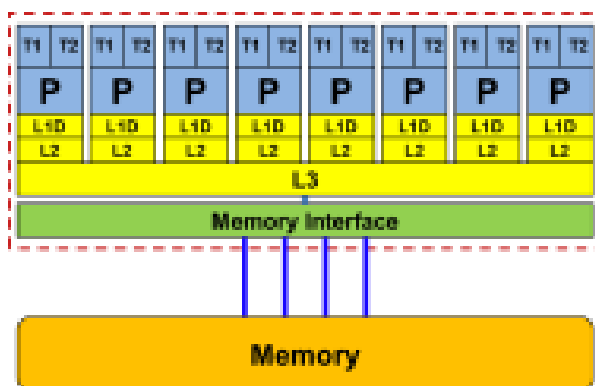
Changing only the compile options makes this code scalable on an 8-core chip



# 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
```

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



# 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.

# Questions to ask in high performance computing

---

- Do I understand the performance behavior of my code?
  - Does the performance **behave in accordance with a model** I have made?
- What is the optimal performance for my code on a given machine?
  - **High Performance Computing == Computing at a bottleneck**
- Can I change my code so that the “optimal performance” gets higher?
  - **Circumventing/ameliorating** the impact of the **bottleneck**
- My model yields **wrong predictions** – what’s wrong?
  - This is the **good case**, because you **learn something**
  - Performance monitoring / microbenchmarking may help clear up the situation