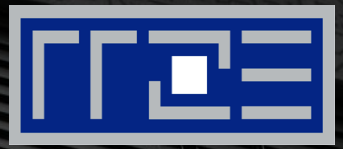
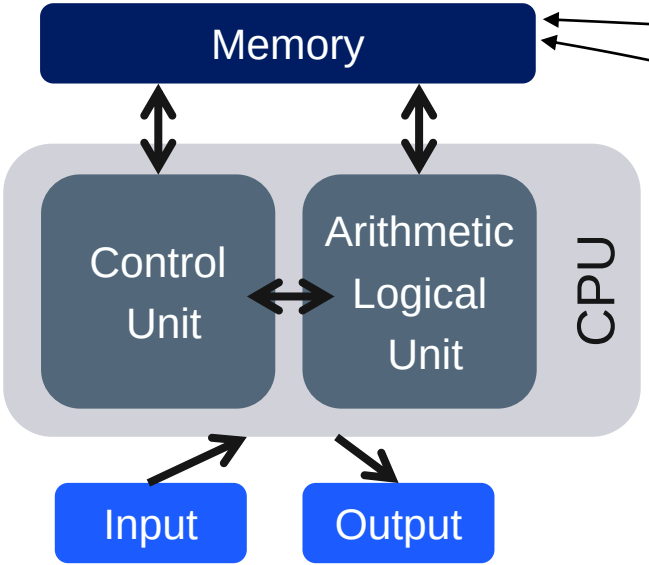
REGIONALES RECHENZENTRUM ERLANGEN [RRZE]



High Performance Computing in a Nutshell – Part 1

HPC Services, RRZE

The Stored Program Computer




```
for (int j=0; j<size; j++){
    sum = sum + V[j];
}
```

```
401d08:   f3 0f 58 04 82      addss   xmm0,[rdx + rax * 4]
401d0d:   48 83 c0 01         add     rax,1
401d11:   39 c7               cmp     edi,eax
401d13:   77 f3               ja      401d08
```

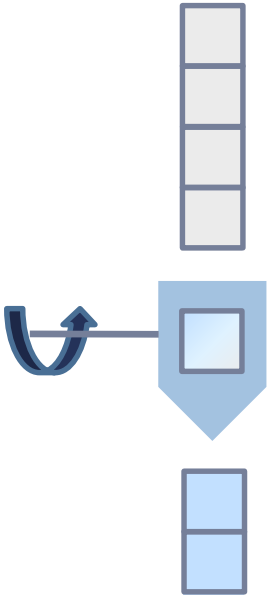
- Improvements for **relevant** software
- **Technical** opportunities
- **Economical** and **marketing** concerns

Execution and memory

Strategies

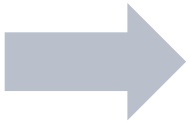
- Increase clock speed
- Parallelism
- Specialization

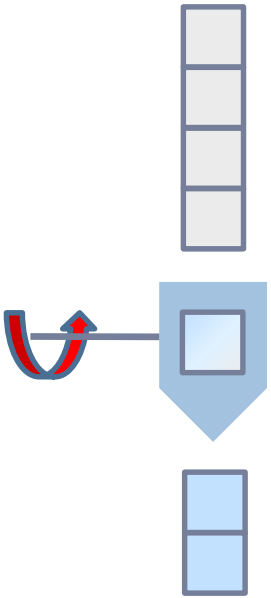
Performance increase by clock frequency increase



Throughput:

1 unit per second



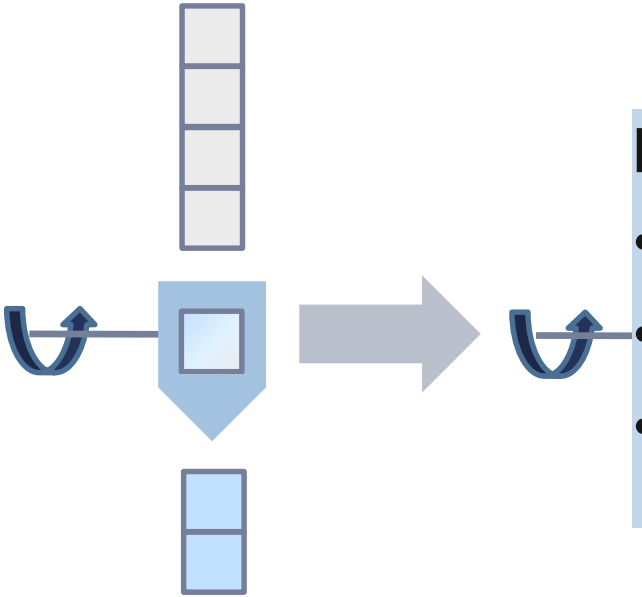


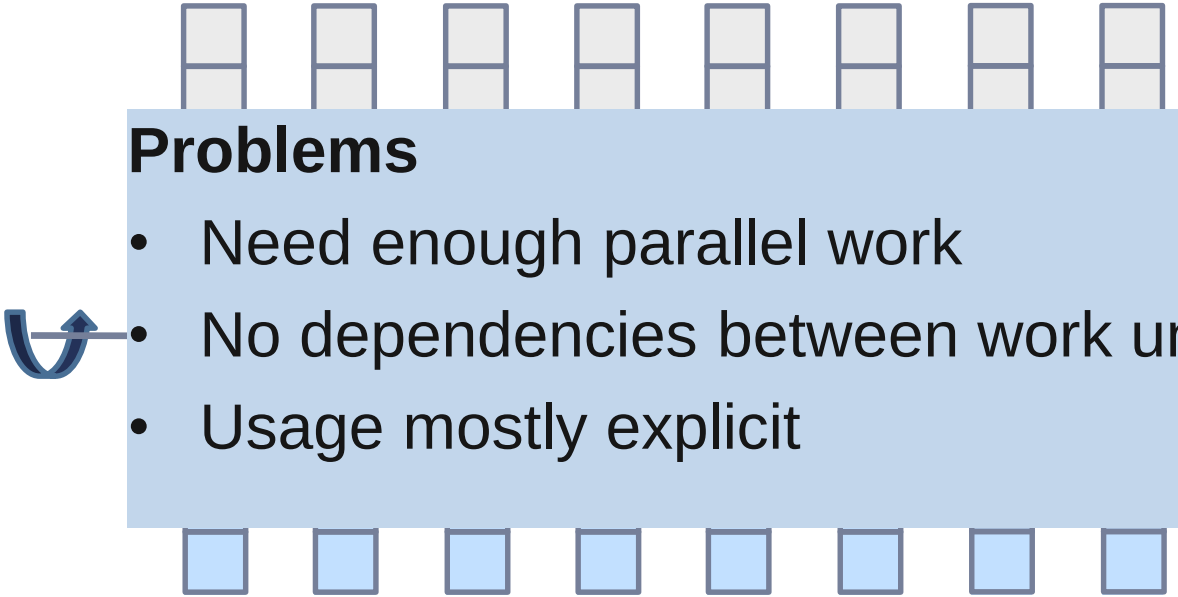
Throughput:

4 units per second

Limit: power dissipation!

Performance increase by parallelization





Problems

- Need enough parallel work
- No dependencies between work units
- Usage mostly explicit

Throughput:

1 unit per second

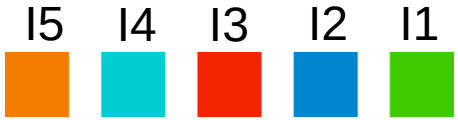
Throughput:

8 units per second

Instruction-level parallelism (ILP)

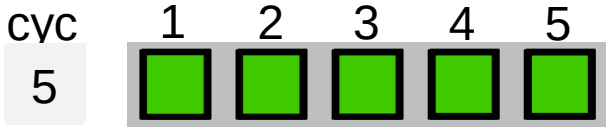
Pipelining

Instructions



Single instruction takes 5 cycles

Stages

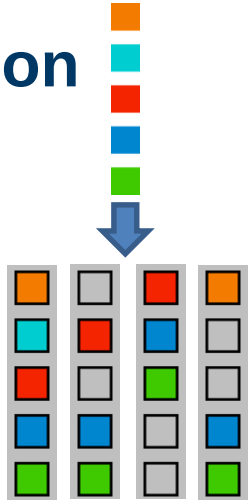


Throughput:

1 instruction per cycle

Speedup by factor 5

Superscalar execution



4-way superscalar

Throughput:

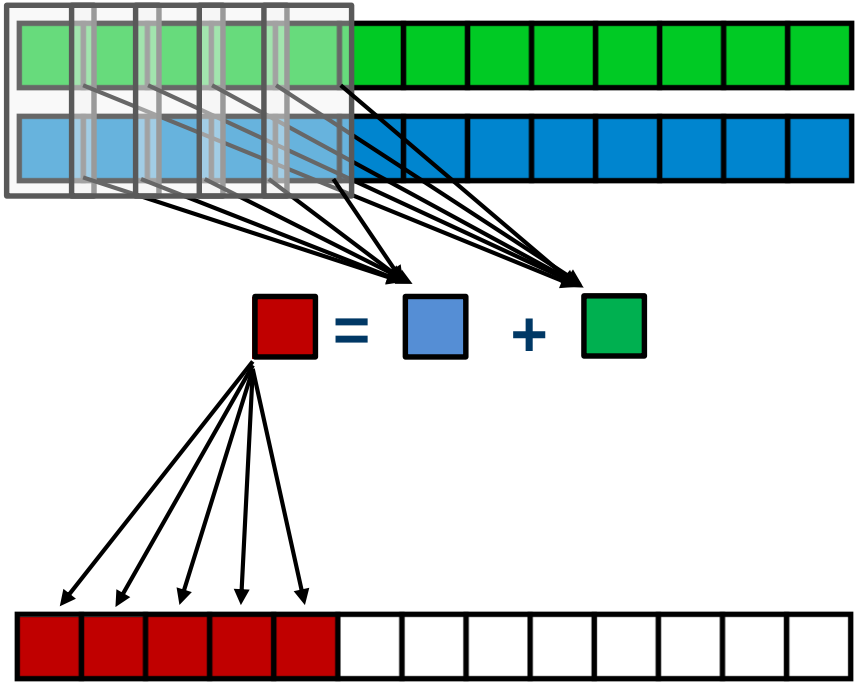
4 instructions per cycle

Data-parallel execution units (SIMD)

```
for (int j=0; j<size; j++){
    A[j] = B[j] + C[j];
}
```

Register width: 1 operand

Scalar execution



Data parallel execution units (SIMD)

```
for (int j=0; j<size; j++){
    A[j] = B[j] + C[j];
}
```

Register widths

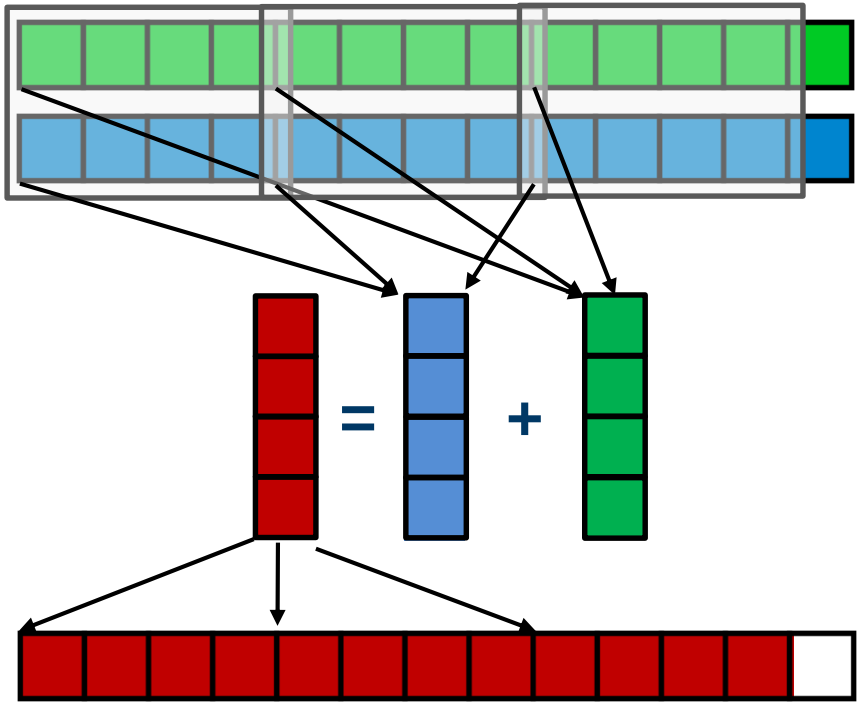
- 1 operand

- 2 operands (SSE)

- 4 operands (AVX)

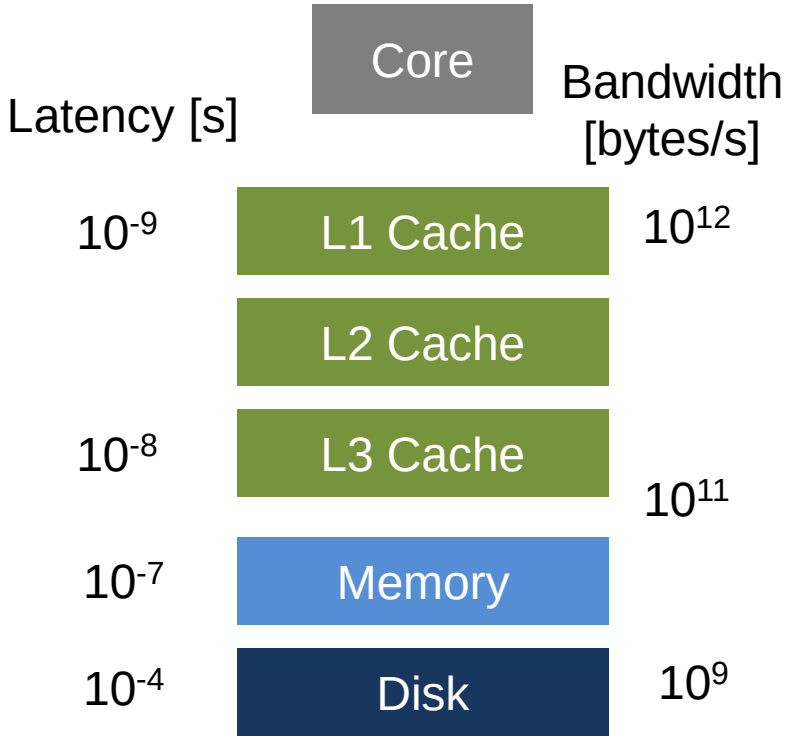
- 8 operands (AVX512)

SIMD execution



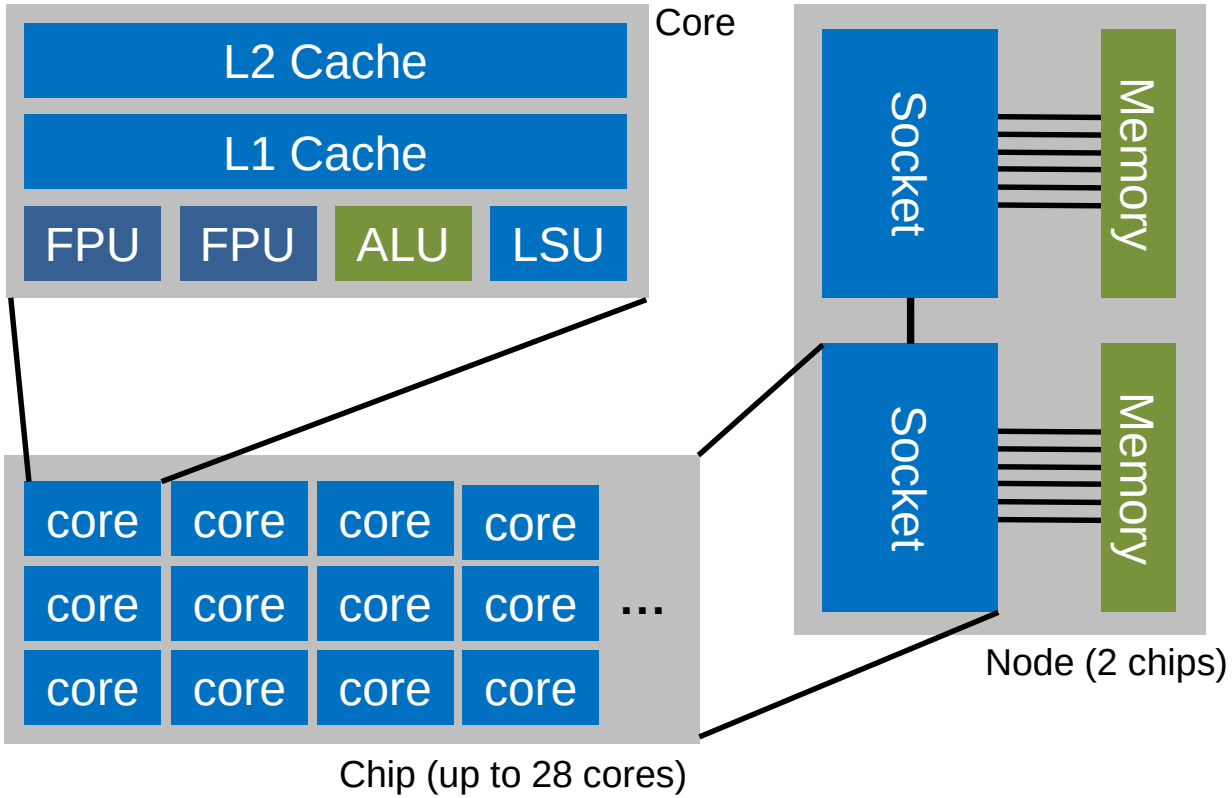
Memory hierarchy

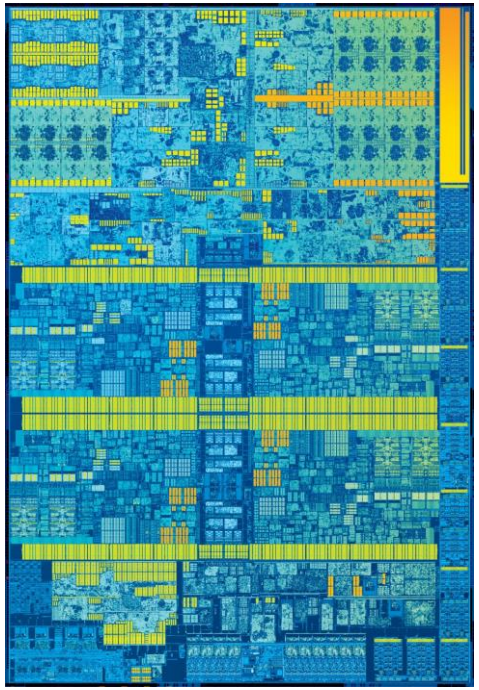
You can **either** build a *small* and *fast* memory **or** a *large* but *slow* memory.



Purpose of many optimizations is therefore to load data from fast memory layers.

Multicore node architectures





~ 8 billion transistors on 500 mm^2

© Intel

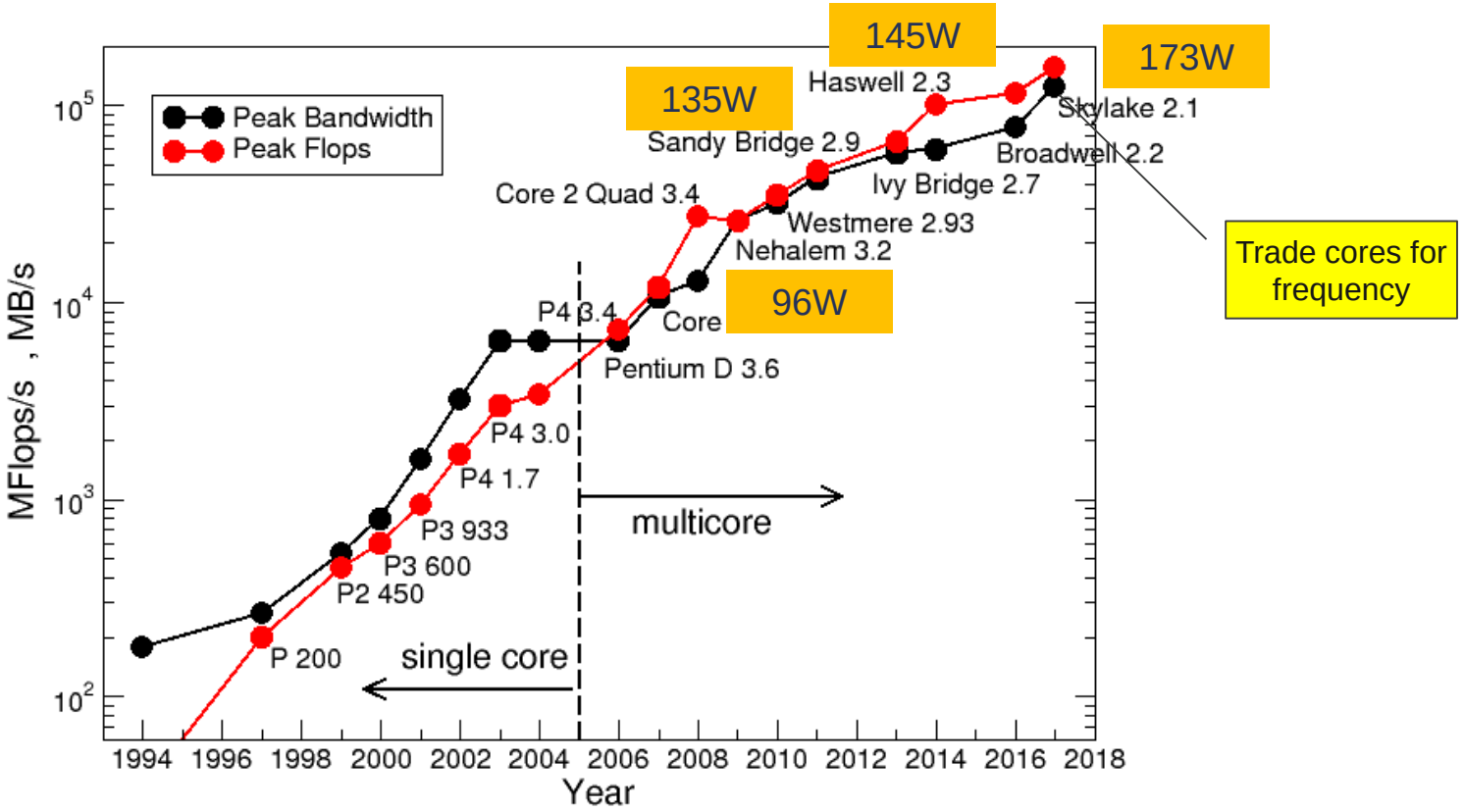
Node-level floating-point performance

$$P_{node} = n_{chips} \cdot n_{cores} \cdot n_{super}^{FP} \cdot n_{FMA} \cdot n_{SIMD} \cdot f$$

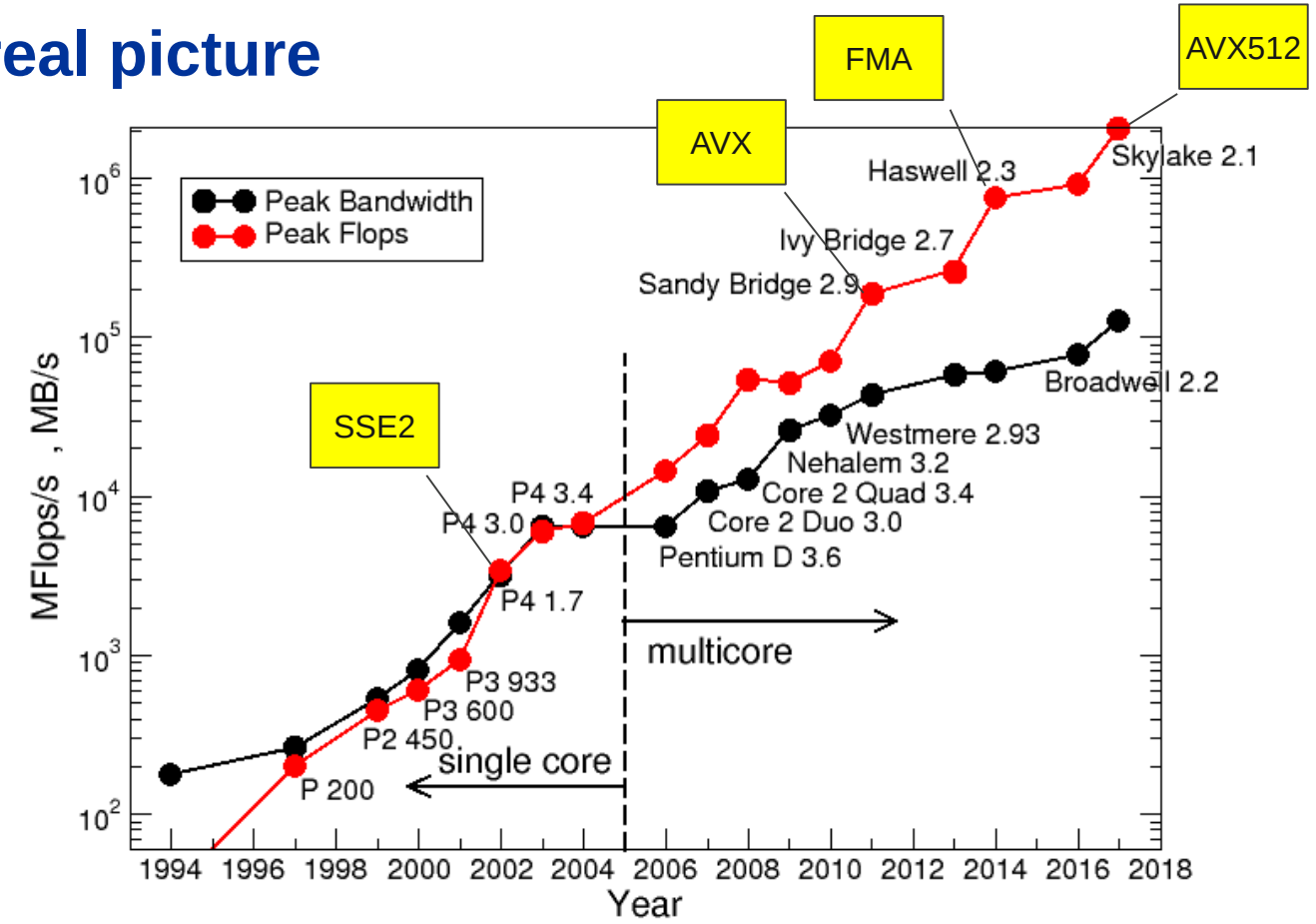
- n_{chips}: chips per node
- n_{cores}: cores per chip
- n_{super}^{FP}: super-scalarity
- n_{FMA}: FMA factor
- n_{SIMD}: SIMD factor
- f: clock speed

$$P_{node} = 2 \cdot 26 \cdot 2 \cdot 2 \cdot 8 \cdot 2.1 \times 10^9 \frac{\text{operations}}{\text{second}} = 3494 \times 10^9 \frac{\text{operations}}{\text{second}}$$

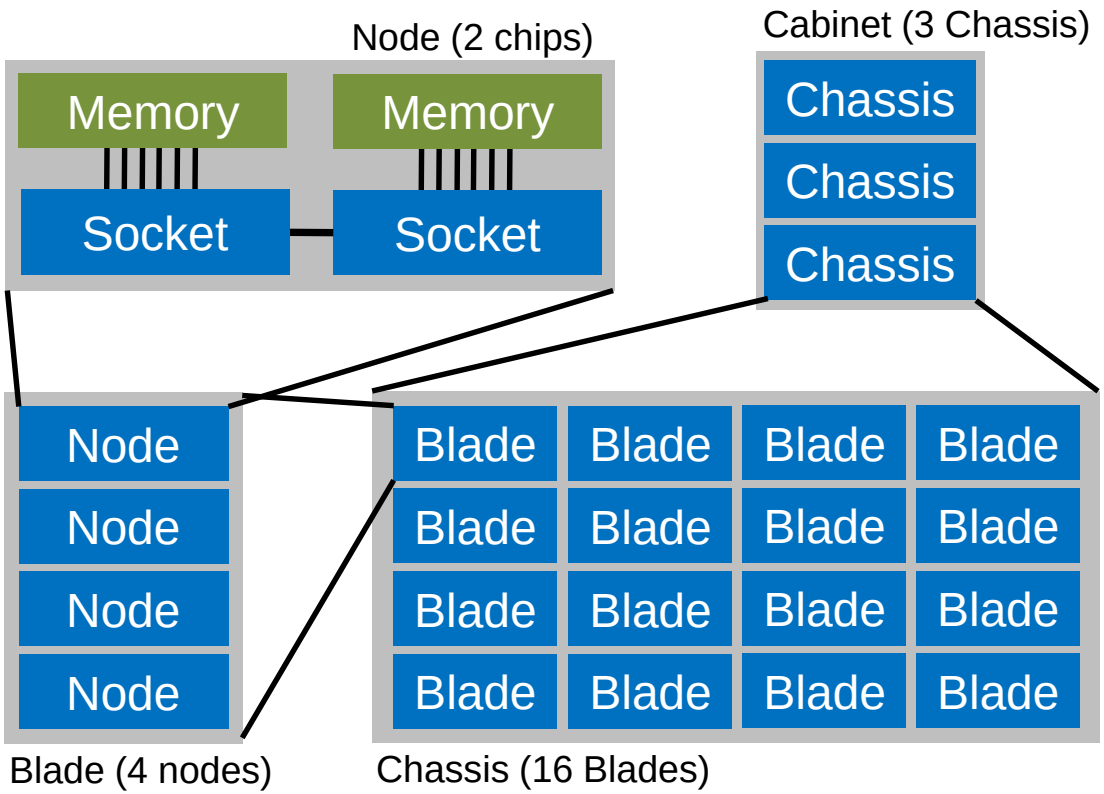
History of Intel chip performance



The real picture



Topology of supercomputers



SuperMUC © LRZ

A HPC System consists of **many** Cabinets!

System-level floating point performance

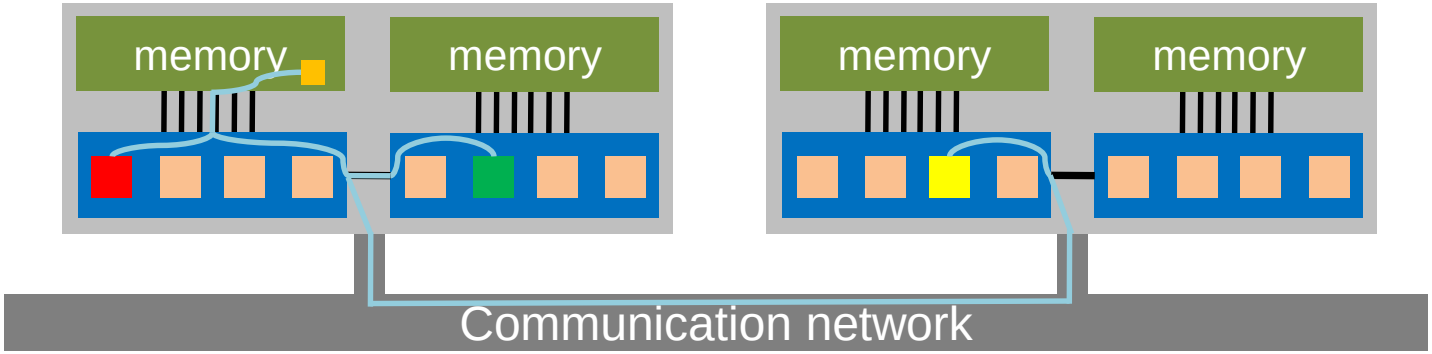
$$P_{system} = n_{cabinets} \cdot n_{chassis} \cdot n_{blades} \cdot n_{nodes} \cdot P_{node}$$

- $n_{cabinets}$: # cabinets
- $n_{chassis}$: Chassis per cabinet
- n_{blades}: Blades per chassis
- n_{nodes}: Nodes per blade
- P_{node}: Node performance

$$P_{system} = 5 \cdot 3 \cdot 16 \cdot 4 \cdot 3494 \times 10^{9} \frac{\text{operations}}{\text{second}} = 3{,}35 \times 10^{15} \frac{\text{operations}}{\text{second}}$$

960 nodes 49920 cores **Power consumption** 350kW

Shared and distributed main memory



Intra-node:

- Communication via main memory

Shared memory

Inter-node:

- Communication via message exchange

Distributed memory

Programming for HPC systems

- **C/C++** and **Fortran** are **best supported**

- **MPI library:** de-facto standard for programming HPC systems (shared and distributed memory)
  - Point-to-point communication, collective communication, parallel file IO, and more
  - Language bindings for C and Fortran (77, 95)
- Many options for threaded shared memory programming, but compiler #pragma-based **OpenMP** standard is the popular option in HPC

Finding parallelism

- Key activity in programing HPC systems
- Example: Summing up a long sequence of numbers

$$\sum = s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + \cdots + s_{999999} + s_{1000000}$$

$$\sum = ((\ldots((((((s_1 + s_2) + s_3) + s_4) + s_5) + s_6) + \cdots + s_{999999}) + s_{1000000})$$

sequential summation

$$\sum = ((s_1 + s_2) + (s_3 + s_4)) + ((s_5 + s_6) + \cdots) + \cdots + (s_{999999} + s_{1000000}))$$

(stepwise) parallel summation

OpenMP for shared-memory parallelism

Compiler-supported parallelization by **source code directives**

```
double s=0.0;
for(i=0; i<N; ++i) {
  b[i] = sqrt(z[i]);
  s = s + a[i]*b[i];
}
```

Distribution of workload among threads

```
double s=0.0;
#pragma omp parallel for reduction(+:s)
for(i=0; i<N; ++i) {
  b[i] = sqrt(z[i]);
  s = s + a[i]*b[i];
}
```

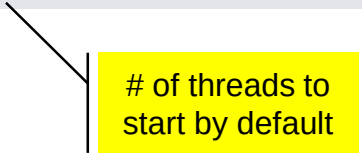
Compiling and running an OpenMP program

- OpenMP support must be **enabled** in the compiler

```
$ gcc -fopenmp code.c -o code.exe
```

- **Environment variables** control execution parameters

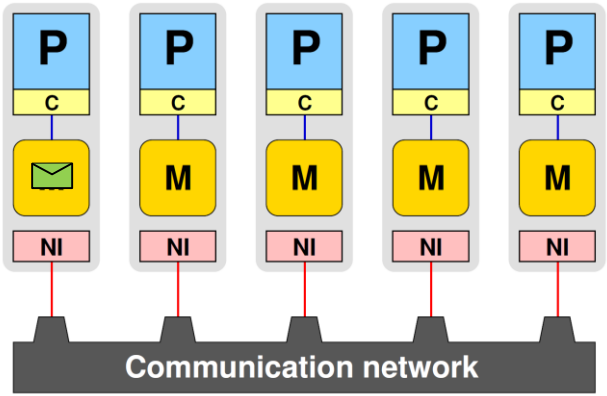
```
$ OMP_NUM_THREADS=4 ./code.exe
```



- Many other performance-relevant settings!

Programming distributed-memory systems: message passing

- Programming model: independent processes with no shared data
- Each process is an autonomous instance of the program code
- Processes communicate via explicit messages



- Parallelization often entails a massive restructuring of the code!

Distributed-memory parallelism via the Message Passing Interface (MPI)

MPI is a library of subroutines for inter-process communication

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int mylen = N/size + (rank>=N%size?0:1);
double s=0.0;
for(i=0; i<mylen; ++i) {
  b[i] = sqrt(z[i]);
  s = s + a[i]*b[i];
}
MPI_Reduce(MPI_IN_PLACE, &s, 1, MPI_DOUBLE,
           MPI_SUM, 0, MPI_COMM_WORLD);
```

Distribution of workload among processes with no concept of shared memory

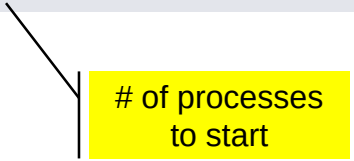
Compiling and running an MPI program

- MPI libraries and headers are typically supplied by compiler scripts

```
$ mpicc code.c -o code.exe
```

- MPI runtime environment is responsible for starting processes on compute nodes (not standardized!)

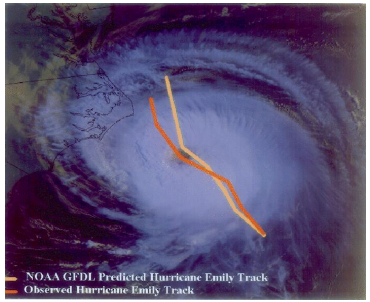
```
$ mpirun -np 256 ./code.exe
```

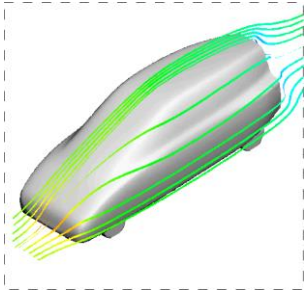


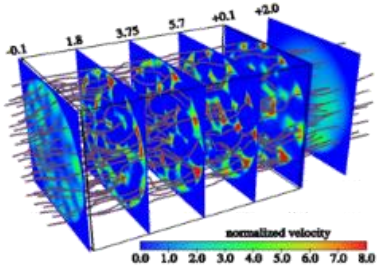
- Many other performance-relevant settings!

Application fields of HPC systems (examples)

- Automotive engineering
  - Crash simulation
  - Aerodynamics
  - Acoustics
- Meteorology
  - Weather forecast
  - Hazard forecast
  - Climate studies
- Materials science
  - Stress and crack propagation
  - Composition of new materials







Application fields of HPC systems (examples)

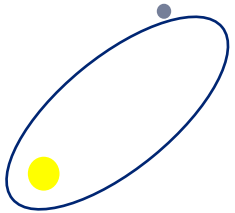
- Biology/biochemistry/medicine
  - “Drug design”
  - Reaction processes
- Physics
  - Fundamental interactions of particles
  - Structure of matter
- Movies
  - Rendering/CGI
  - Realistic physics (particles, explosions,…)
- Growing fields of application outside natural science and engineering, e.g., economics, linguistics, sports, …

From physics to model

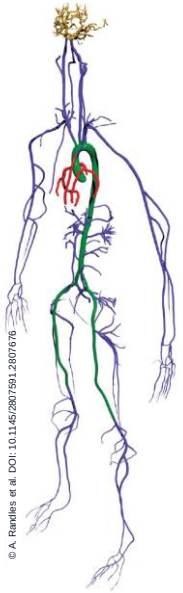
$$F_{12} = \frac{\gamma m_1 m_2}{r_{12}^2}$$

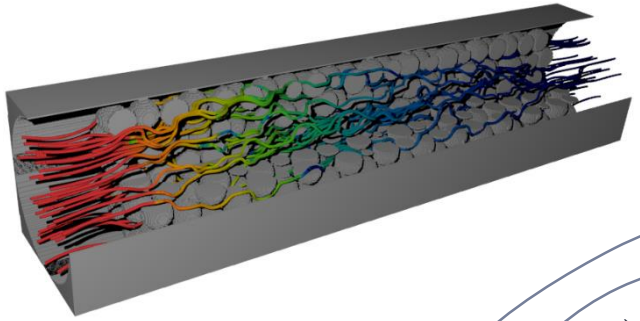
$$F = m \cdot a$$

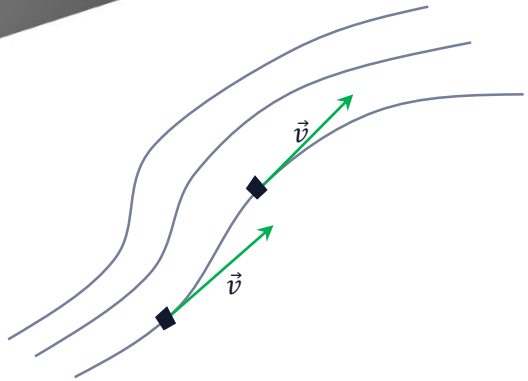
$$h(t) = h_0 - \frac{1}{2} g\, t^2$$

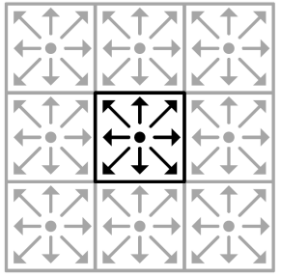


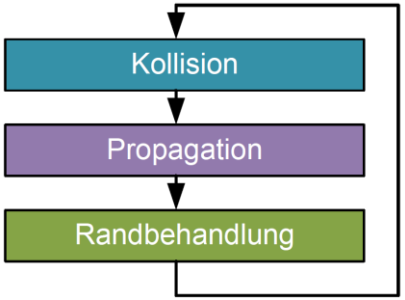
Model to algorithm: an example from fluid dynamics





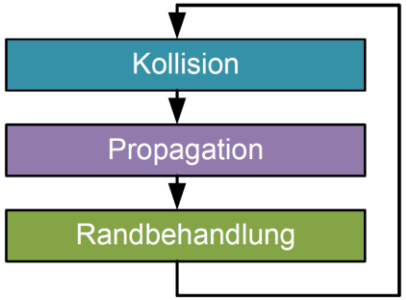






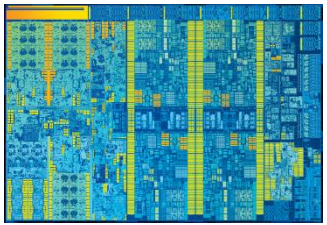
$$\rho\dot{\vec{v}} = \rho\left(\frac{\partial\vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\vec{v}\right) = -\nabla p + \mu\Delta\vec{v} + (\lambda+\mu)\nabla(\nabla\cdot\vec{v}) + \vec{f}.$$

From algorithm to program code



```
      do 260 k = 1,kmax
        do 260 i = 1,imax
          etaref = spec2(i,1,k,mb)
          do 265 j = 1,jmax
            spec2(i,j,k,mb) = spec2(i,j,k,mb)-etaref
265       continue
260   continue
```

Compiler (+OpenMP +MPI)



```
0f 29 5c c7 30      movaps %xmm3,0x30(%rdi,%rax,8)
48 83 c0 08         add    $0x8,%rax
78 a6               js     401b50 <triad_asm+0x4b>
```

Outlook

- Structure and programming of high performance computers is complicated
- Very few automatic mechanisms exist
- An awareness of the limitations of the hardware and software is required

- Don’t Panic.

Upcoming: Part 2 of “HPC in a Nutshell”

- Thursday May 9, 15:00, RRZE 2.049
  - RRZE’s HPC systems
  - Getting access
  - Running jobs
  - Data management
  - Performance issues

Thank you very much

HPC@RRZE