High Performance Computing in a Nutshell – Part 1

HPC Services, RRZE
The Stored Program Computer

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

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>401d08</td>
<td>f3 0f 58 04 82</td>
<td>addss xmm0,[rdx + rax * 4]</td>
</tr>
<tr>
<td>401d0d</td>
<td>48 83 c0 01</td>
<td>add rax,1</td>
</tr>
<tr>
<td>401d11</td>
<td>39 c7</td>
<td>cmp edi,eax</td>
</tr>
<tr>
<td>401d13</td>
<td>77 f3</td>
<td>ja 401d08</td>
</tr>
</tbody>
</table>

**Strategies**
- Increase clock speed
- Parallelism
- Specialization

**Execution and memory**
Performance increase by clock frequency increase

Throughput: 1 unit per second

Limit: power dissipation!

Throughput: 4 units per second
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

<table>
<thead>
<tr>
<th>I5</th>
<th>I4</th>
<th>I3</th>
<th>I2</th>
<th>I1</th>
</tr>
</thead>
</table>

Single instruction takes 5 cycles

<table>
<thead>
<tr>
<th>cyc</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

Throughput:
1 instruction per cycle

Speedup by factor 5

**Superscalar execution**

4-way superscalar

Throughput:
4 instructions per cycle
Data-parallel execution units (SIMD)

```c
for (int j=0; j<size; j++){
}
```

Register width: 1 operand

Scalar execution
Data parallel execution units (SIMD)

```
for (int j=0; j<size; j++)
```

Register widths

- 1 operand
- 2 operands (SSE)
- 4 operands (AVX)
- 8 operands (AVX512)
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

Chip (up to 28 cores)

L2 Cache
L1 Cache
FPU
ALU
LSU

Socket
Memory

Node (2 chips)

~ 8 billion transistors on 500 mm²
Node-level floating-point performance

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

- chips per node
- cores per chip
- super-scalarity
- FMA factor
- SIMD factor
- clock speed

\[ P_{\text{node}} = 2 \cdot 26 \cdot 2 \cdot 2 \cdot 8 \cdot 2.1 \times 10^9 \frac{\text{operations}}{\text{second}} = 3494 \times 10^9 \frac{\text{operations}}{\text{second}} \]
History of Intel chip performance

Trade cores for frequency

<table>
<thead>
<tr>
<th>Year</th>
<th>MFlops/s, MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>1998</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td></td>
</tr>
</tbody>
</table>
The real picture

```
<table>
<thead>
<tr>
<th>Year</th>
<th>Processor</th>
<th>MFlops/s, MB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>Pentium Pro 200</td>
<td>10^2</td>
</tr>
<tr>
<td>1996</td>
<td>Pentium 2450</td>
<td>10^3</td>
</tr>
<tr>
<td>1998</td>
<td>Intel Pentium 3 933</td>
<td>10^4</td>
</tr>
<tr>
<td>2000</td>
<td>Intel Pentium 3 600</td>
<td>10^5</td>
</tr>
<tr>
<td>2002</td>
<td>Intel Core 2 Duo 3.0</td>
<td>10^6</td>
</tr>
<tr>
<td>2004</td>
<td>Intel Core 2 Quad 3.4</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>Intel Westmere 2.93</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Intel Nehalem 3.2</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Intel Sandy Bridge 2.9</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Intel Ivy Bridge 2.7</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>Intel Haswell 2.3</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>Intel Skylake 2.1</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>Intel Broadwell 2.2</td>
<td></td>
</tr>
</tbody>
</table>
```

- **SSE2**
- **AVX**
- **AVX512**
- **FMA**

(single core) → (multicore)
A HPC System consists of many Cabinets!
System-level floating point performance

\[ P_{\text{system}} = n_{\text{cabinets}} \cdot n_{\text{chassis}} \cdot n_{\text{blades}} \cdot n_{\text{nodes}} \cdot P_{\text{node}} \]

- # cabinets
- Chassis per cabinet
- Blades per chassis
- Nodes per blade
- Node performance

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

960 nodes 49920 cores **Power consumption** 350kW
Shared and distributed main memory

Intra-node:
- Communication via main memory

Inter-node:
- Communication via message exchange

Shared memory

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 programming HPC systems
- Example: Summing up a long sequence of numbers

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

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

**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];
}

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

- **Environment variables** control execution parameters
  
  ```bash
  $ 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

```c
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

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

\[ h(t) = h_0 - \frac{1}{2} g t^2 \]
Model to algorithm: an example from fluid dynamics

\[ \rho \frac{D \vec{v}}{D t} = \rho \left( \frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} \right) = -\nabla p + \mu \Delta \vec{v} + (\lambda + \mu) \nabla (\nabla \cdot \vec{v}) + \vec{f}. \]
From algorithm to program code

```fortran
  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
  do 260 continue
  of 29 5c c7 30 movaps %xmm3,0x30(%rdi,%rax,8)
  48 83 c0 08 add $0x8,%rax
  78 a6 js 401b50 <triad_asm+0x4b>
```

Compiler (+OpenMP +MPI)
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