Architectures & application performance improvement: the HPC user point of view

Matthieu Haefele[1]

[1] Maison de la Simulation

Acknoledgments: M. Snir, G. Hager, G. Wellein, L. Saugé, J. Bigot, M. Klemm, A. Koehler

9-13 May 2016, Saint Germain au Mont d'Or

External material

This presentation makes use of several materials that could be retrieved on the web here:

- Presentations from G. Hager and G. Wellein during PATC@LRZ on node level engineering[1]
  - Node architecture
  - Roofline model
  - Microbenchmarking
- Presentation from M. Klemm on Xeon Phi architecture during GENCI workshop
- Presentation from A. Koehler on OpenPower architecture
- Presentation from M. Snir on future architecture at ICS conference 2014

[1] http://moodle.rrze.uni-erlangen.de/course/view.php?id=274&username=guest&password=guest

Outline

- Hardware
  - CPU vs Accelerator type
  - Quick look in the future
- Software
  - Scalability and performance
  - Programming models
  - Performance improvement
- Production codes & human factor

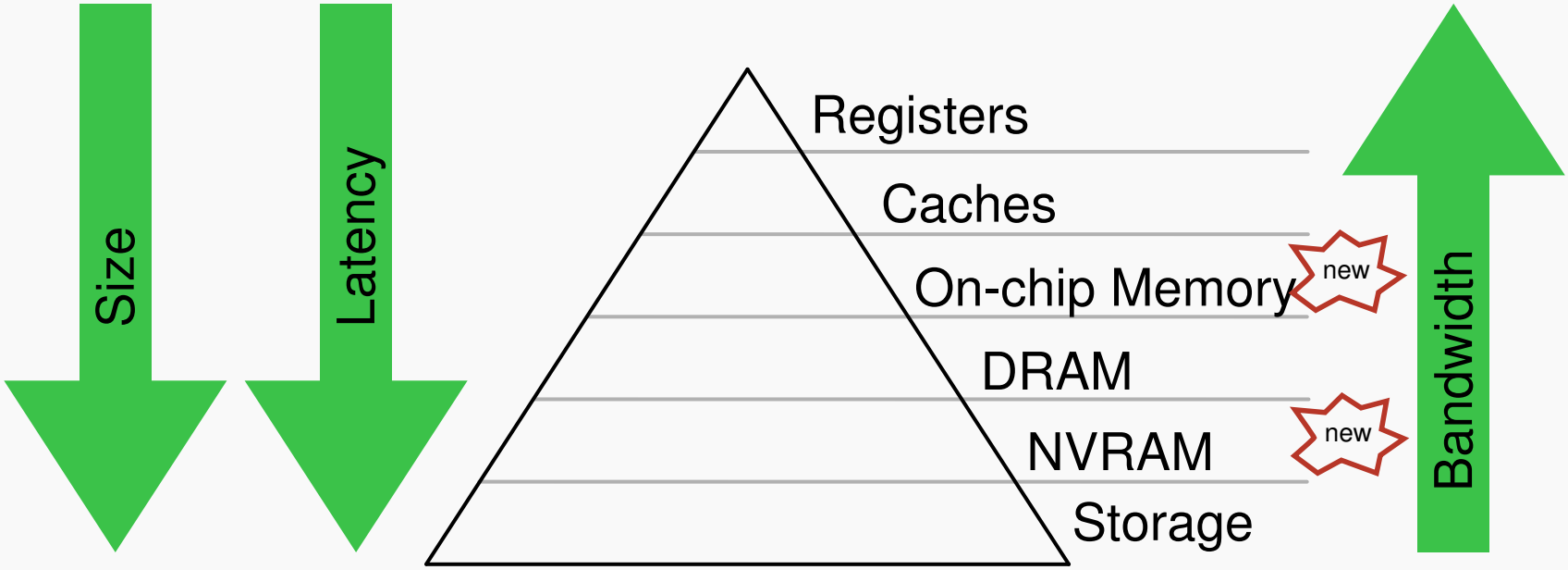
Outline: Hardware

- Recent CPU architectures (Intel Xeon and AMD Bulldozer)
  - General architecture of a cached based processor
  - Pipeline
  - Superscalar processors (ILP)
  - Simultaneous multi-threading (SMT)
  - Single Instruction Multiple Data (SIMD)
  - Memory hierarchy
  - UMA vs ccNUMA
  - Peak performance
- Intel Xeon Phi KNL
- IBM OpenPower: IBM Power CPU + NVidia GPU
- Comparison
- Hardware perspectives in the future

External material

P6-17 from Node architecture slides

Memory hierarchy



External material

P18-21 from Node architecture slides

External material

Xeon Phi KNL presentation

External material

OpenPower presentation

Architecture comparison

Proc.	2x Intel Haswell	Intel KNL	NVidia Pascal
Techno.	22nm	14nm	16nm
#cores	24	up to 72	??
SMT	2	4	N.A
Mem	128-512 GiB DRAM	16GB MCDRAM + 384 GB DRAM	16GB on Chip
Peak perf.	0.96 TF/s	3 TF/s	3 TF/s
Mem band.	100GB/s	500GB/s MCDRAM? + ??DRAM	1TB/s? on chip + 50GB/s? NVLink
Power	240W	??	??

Architecture comparison

Crucial issues:

- Memory management
- Intra-node parallelization

Justification for these new architectures: power requirement

External material

M. Snir presentation

Outline: Software

- Scalability and performance
  - Definition
  - Performance model: roofline model
- Programming models
  - A not exhaustive list
  - Zoom on OpenMP
  - Topology awareness of the programming model
- Performance improvement
  - Performance measurement
  - Code optimisation
    - Load balancing
    - Vectorization
    - Arithmetic intensity increase: blocking

External material

Introduction to the roofline model and the microbenchmarking.

Programming models

- Distributed-memory (inter-node)
  - **MPI**
  - PVM (gone)
  - PGAS: CoArray Fortran, UPC
  - X10, Chapel, Julia, ...
- Shared-memory (intra-node)
  - **OpenMP**
  - Posix threads, Intel Threading Building Blocks (TBB), Cilk+, ...
  - Accelerator: Cuda, OpenCL, OpenAcc
  - Runtime systems: StarPu, XKaapi, OmpSs, ...

Programming models cont.

- Hybrid
  - **MPI+OpenMP**
  - Pure MPI
  - MPI + any shared-memory model
  - MPI (+OpenMP) + any accelerator
  - MPI + any runtime systems

OpenMP programming models

1. Coarse grain: work distribution w.r.t. thread id (MPI like)
2. OMP DO (or FOR): Fine grain
3. OMP Task: much more tasks than threads $\Rightarrow$ need for scheduling

$\Rightarrow$ Runtime systems can be plugged in here

4. OMP Target: offload directives for accelerators

OpenMP is a norm gathering now several programming models!

Performance measurement

”Measuring is always better than guessing”

Brian Wylie, Scalasca developer

- Application/routine level
  - Hot spots
  - Load imbalances
  - Communication / IO overheads
- Kernel level
  - Performance models
  - Code optimisation

Performance measurement tools

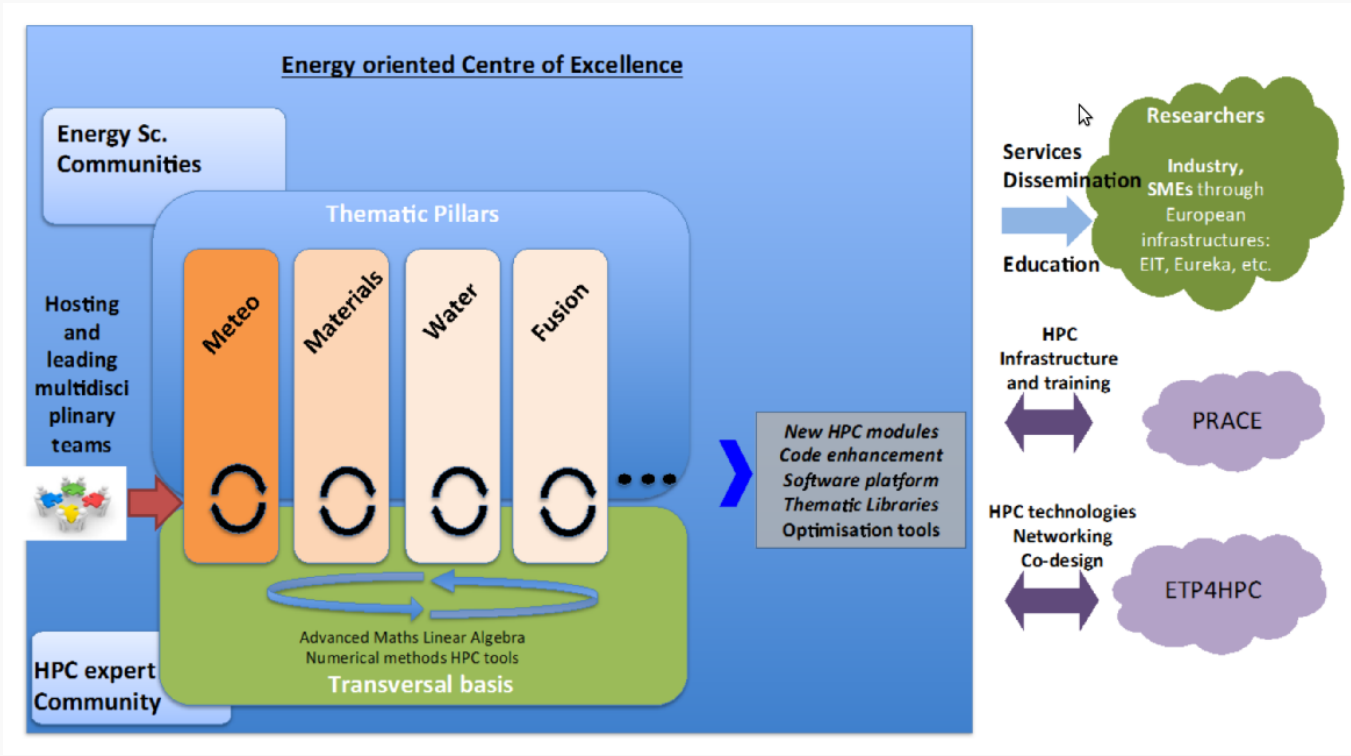
- Application/routine level
  - **Alinea performance report**: nice performance overview of a code with information gathered from a single run
  - **ScoreP/Scalasca**: profiling and tracing based on source code instrumentation methods
  - **Intel VTune** : profiling and tracing based on runtime hardware counter sampling methods
  - **Extrae/Paraver**: profiling and tracing based on runtime hardware counter sampling methods
  - **Darshan** : IO measurements
  - **Unix time command**: time measurements
  - **ldrMem**: memory footprint measurements
  - **PAPI**: hardware counters extraction

Performance measurement tools

- Kernel level
  - **Intel advisor**: vectorization quality evaluation
  - **Compiler logs**: vectorization quality evaluation
  - **Manual instrumentation**: performance measurement

EoCoE

Energy oriented Centre of Excellence



 M. Haefele - 2016

EoCoE: Performance evaluation

- 23 codes in the consortium
- Definition of a single set of 31 standard metrics
- Automated extraction process with JUBE
  - 4 compilations
  - 7 runs

Vectorization

```
do i = ibeg_w, iend_w
   vsumzk0=0.0d0
   do j = nummove+1,num
      if (i==j) then
         vsumzk0=vsumzk0+q(j)*sqrpieta
      else
         zij=z(i)-z(j)
         zijsq=zij*zij
         rerf = erf(eta*zij)
         vsumzk0=vsumzk0+q(j)*((sqrpieta*exp(-etasq*zijsq))+&
            (pi*zij*rerf))
      end if
   enddo
   cgpot(i)=cgpot(i)-vsumzk0
enddo
```

Vectorization

- The *if* statement introduces an issue
- The iteration $j = i$ executes different code than $j = i - 1$ and $j = i + 1$

$\Rightarrow$ This code cannot be SIMD

$\Rightarrow$ The whole j loop is not vectorized

- The *else* part is also true for $j = i$

$\Rightarrow$ Just suppress the if statement to win a factor 3.2 on this kernel with AVX

Production codes & human factor

- Example: the Gysela5D code
- Software engineering
- The next step

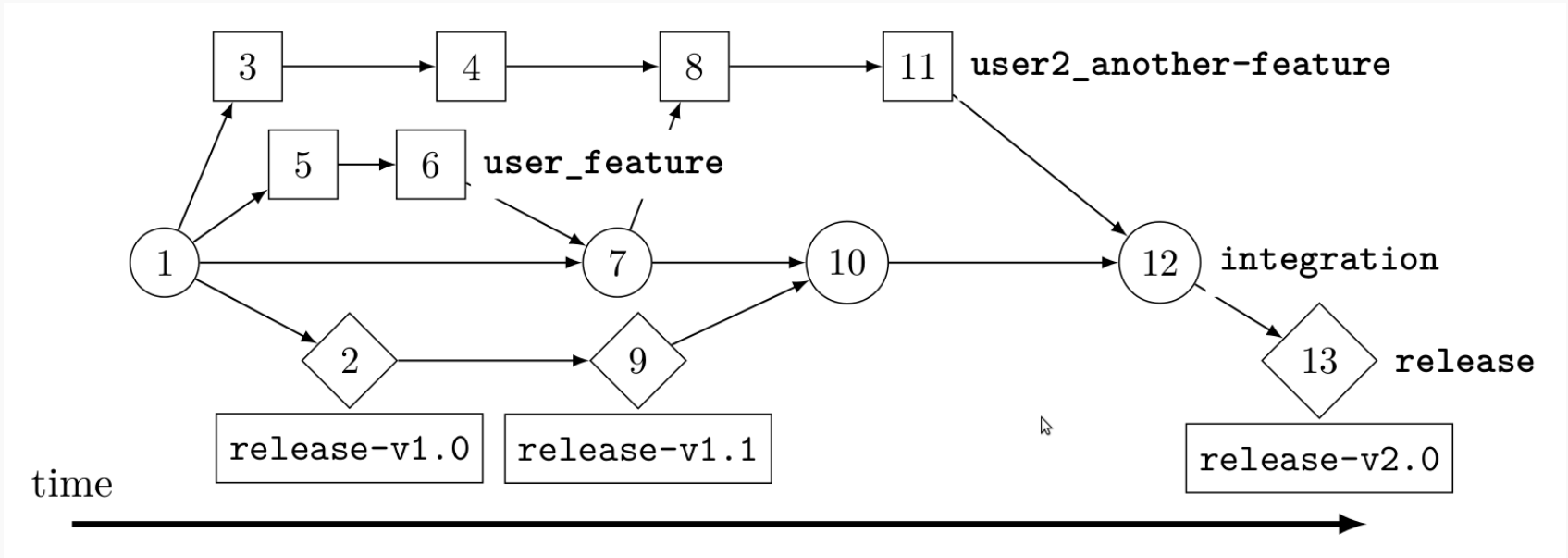
The Gysela5D code: institution

- Developed at CEA (IRFM)
- Scientific area: turbulence in tokamaks (fusion devices)
- Gyrokinetic model + quasi-neutrality equation (5D + t)
- 91% relative efficiency on the full BlueGene/Q @ JSC (1.8 M threads)

The Gysela5D code: software

- 5 core developers: mostly computer scientists and applied mathematicians
- 10 users: physicists
- 60k lines of Fortran90 + 13k lines of C
- 174k lines modification spread over 1700 commits (year 2014)
- 33 releases: 16 features + 17 bugfix (year 2014)
- 10 target paltforms at any point in time

An HPC development workflow



Continuous integration

Several tests implemented:

- 82 different compilations (30 minutes)
- 13 bitwise comparison test runs (4 minutes)
- 15 ”unit tests” (4 minutes)
- 5 system tests on 64 cores (11 minutes)

Execution:

- Automated with Jenkins
- run within VMs on the INRIA CI platform
- run on MdlS poincare cluster

Still not enough...

- InKS (Independent Kernel Scheduling) disentangles the algorithm from the implementation
- PDI (Parallel Data Interface) disentangles the IO package used from the implementation
- Continuous integration platform on production machines set up within EoCoE

EoCoE: CI on HPC infrastructure

- Hardware and software stack different on different machines
- Needs for running tests on production machines
- Security issues

An HPC continuous integration infrastructure will be deployed @IDRIS

EoCoE: PDI

Parallel Data Interface

- The appropriate IO package depends on your needs and on the machine
- Your needs evolve, the machines too !
- Now: for each evolution, new implementation with ifdefs
- With PDI: no code change, changes localized in separate configuration files

Conclusion

- Hardware: exascale machines will be around for 2022 or so and will be likely accelerator based
- Software: MPI+OpenMP will likely remain the standard

$\Rightarrow$ The jump to make in programming style is for pre-exascale machines

- Maintaining HPC production codes is a complex task from the software engineering point of view

Conclusion cont.

- OpenMP Task + RunTime systems sound like a nice solution for intra-node parallelization
- Beyond exascale, dedicated hardware is expected to emerge. It will likely not be X86 compatible and will probably not have a fortran compiler. That will become interesting ;)

Messages:

- To mathematicians:
  - At order zero, computing is free, moving data is expensive
  - Privilege algorithms that could benefit from asynchronism
- To compiler people:
  - Help us generating efficient source codes
  - Help us managing software engineering issues