Modelowanie wydajności i optymalizacja na współczesnych architekturach obliczeniowych

Miłosz Ciżnicki, Piotr Kopta

Poznań, 24.05.2017

Outline

- How fast is CPU?
- Application characteristic
- Tools available in PSNC
- Roofline model
- Example: EULAG

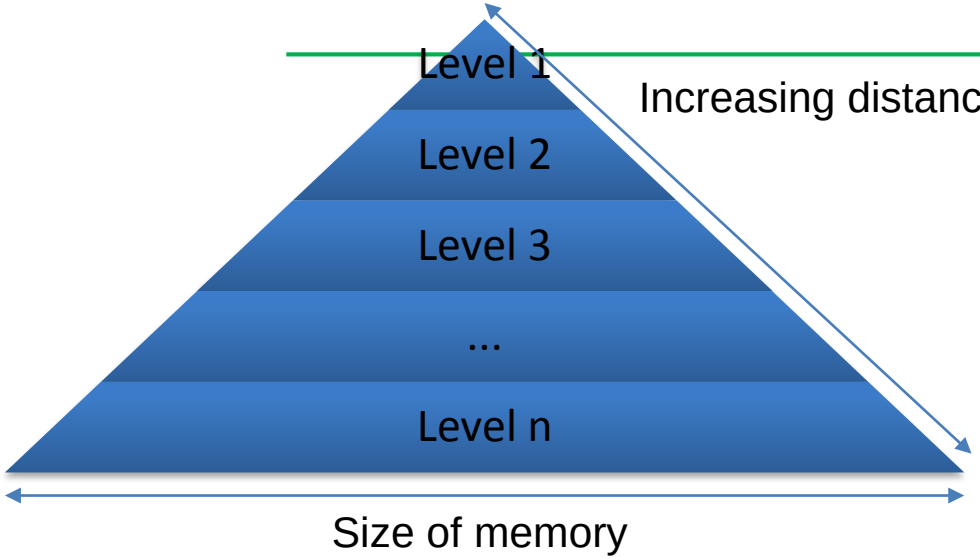
What determines performance

- How fast is CPU?
- How fast data can be moved?
- Application characteristic (CPU-bound or memory-bound)
- Performance optimization for one application may not work for the other
- Running benchmarks helps to understand limitations of the hardware
- Profiling code helps to understand application performance characteristic (and limitations)

CPU-bound vs Mem-bound

- Compute-bound – time to complete task is determined by the speed/number of CPUs
- Memory-bound - time to complete task is determined by the memory latency and/or bandwidth
- Nowadays it is less likely to have just one component being responsible for the time required to complete task
- However starting from multicore era, the memory latency and bandwidth became important bottleneck

How to run faster - cache



Increasing distance from CPU = greater access time

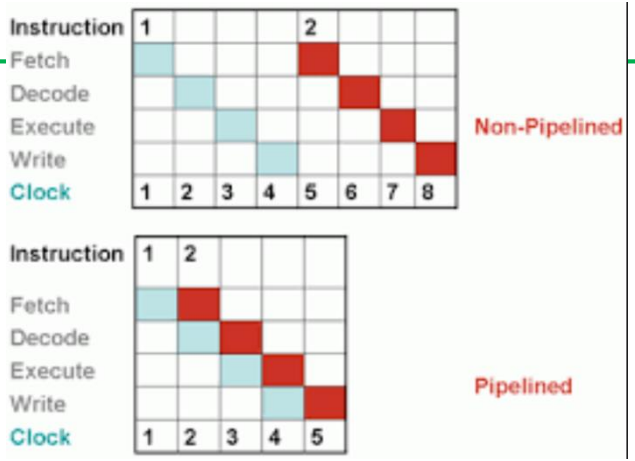
Example:

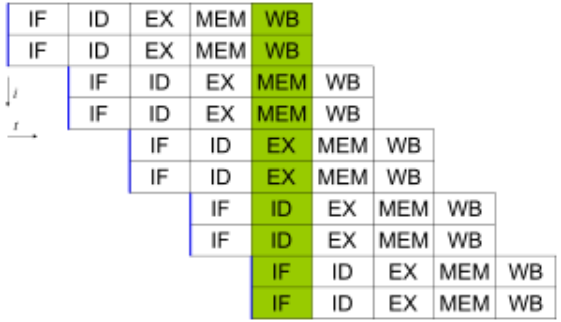
L1: 32 kB, latency 3 cycles
L2: 256 kB, latency 10 cycles
L3: 8MB, latency 40 cycles
DRAM: 16GB, latency 200 cycles
DISK: 1TB, latency 1.000.000 cycles

Cache hit – data found in cache
Cache miss – data not found in cache, thus must be copied from lower memory level
Capacity miss – cache runs out of space for new data
Conflict miss – more that one item is mapped to the same location in cache

How to run faster – cont.

- Pipelining
  - Multiple steps in one operation: fetch, decode, execute, memory, write back
  - Dependencies and branching may affect the pipeline
- Instruction level parallelism (superscalar)
  - Multiple instructions executed at the same time
  - Loop unroling
- Multicore
  - Combines caches, pipelining and superscalars





Fast and slow operations

- In terms of cost
- Fast and inexpensive: add, multiply, sub, fma (fused multiply add)
- Medium: divide, modules, sqrt
- Slow: trans (support in newer CPUs and GPUs)
- Very slow: power

- Use linear algebra (BLAS, LAPACK) and math libraries (Intel MKL)

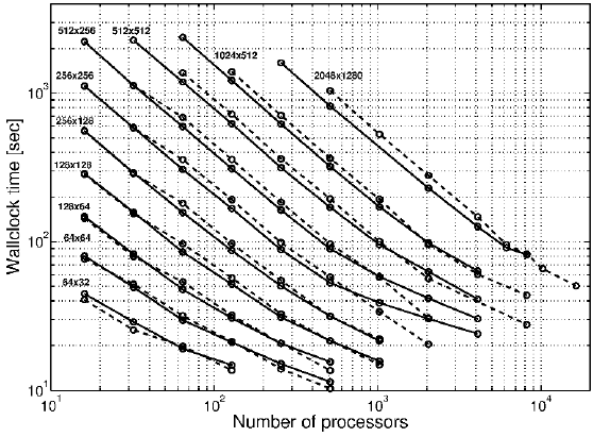
Code optimization

- Use math, BLAS, LAPACK libraries (for effcient math routines, to utilize max. of pipelining instructions)
- Efficient code is often not very portable (performance is decreased among different architectures)
- Use compilers optimizations (but can’t optimize everything)
- Profile your code
- Use accelerators (GPUs) and coprocessors (Intel Phi)
- Write generic code + specialized kernels

	MKL single core vs naive single core speedup	MKL (up to 28 cores) vs naive single core speedup
matrix (2048 x 2048) product	**52**x	**543**x
Cholesky (4096) decomposition	**12**x	**190**x

Assesing performance – strong vs. weak scaling

- Strong scaling: fixed problem size, measure speedup with more processors
  - Example: WRF – climate modelling application
- Weak scaling: test for time for fixed problem size per number of processors
  - Example: LINPACK benchmark, more efficient with more memory

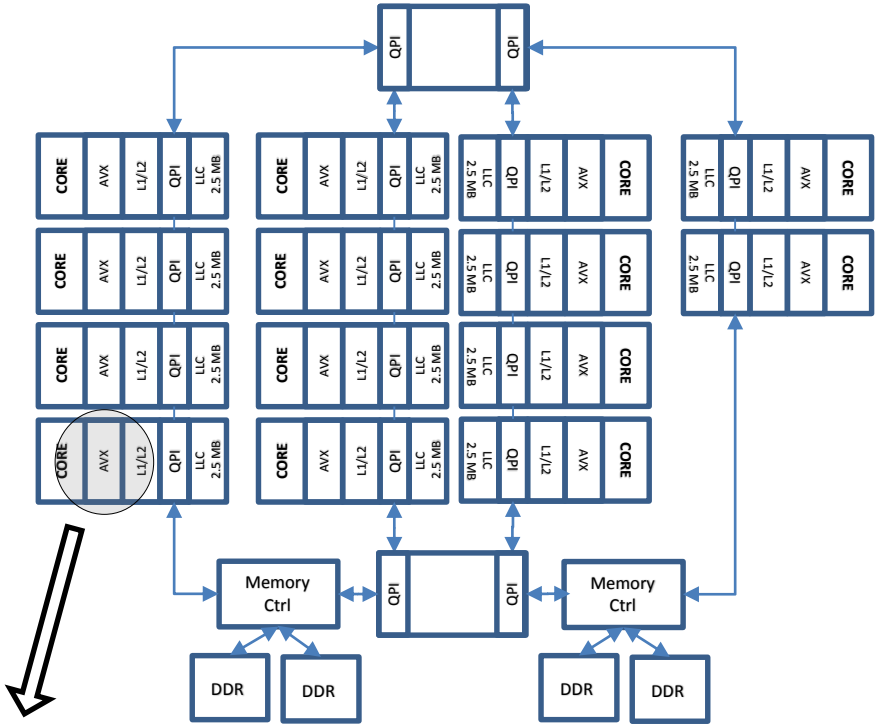


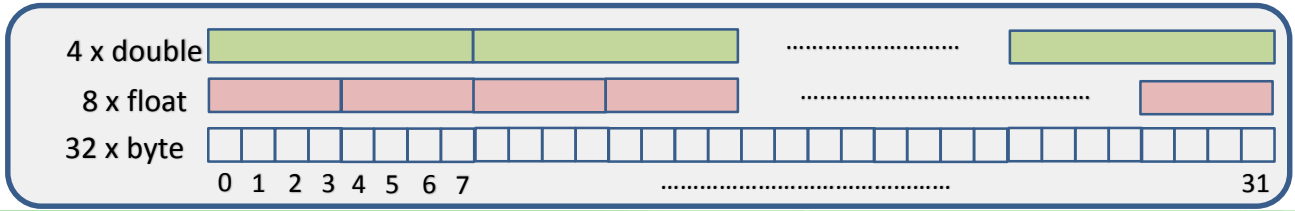
Assesing performance – time vs FLOPS

- Does execution time give enough information about application performance?
- FLOPS/s - how many floating points operation per second can your code do?
- Peak performance – clock rate * number of single/double precision add and or muls per clock or FMA
  - E.g. 2.0GHz * 8 FLOP / clock = 16 GFLOP/s
  - Can never be reach (data load/store)
- Suistained performance – application dependant
- Does FLOP/s gives enough information about performance?

Intel Xeon E5 2697 v3

- 2.6 GHz (3.6 GHz Turbo)
- 14 cores
- Theoretical performance ~650 GFlops
- Memory bandwidth 68 GB/s
- AVX 2





Wyzwania współczesnych architektur obliczeniowych

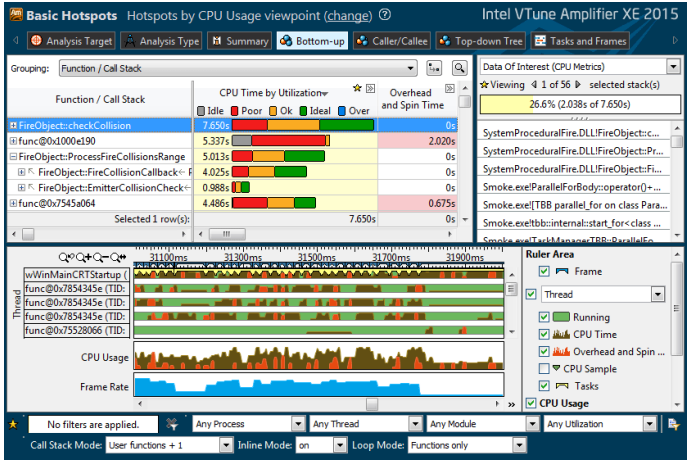
- 650 Gflops only for **FMA** (ang. *fused multiply-add*) $\boldsymbol{d} = \boldsymbol{a} \times \boldsymbol{b} + \boldsymbol{c}$

  325 Gflops for the other operations
- ~ **10 flops / B**

  for (single precision float) $d = a \times d + c$ (a, c constants) - 2 flops, 8 B
- 68 GB/s, 27 G(single precision float)/s, < 9 G(double precision float)s
- Vectorization – SIMD
- Multithreading – 14 cores

Hardware performance counters

- Set of special-purpose hardware registers to store counts of hardware-related activities
- Can help in spotting the application bottlenecks
- Allow for low-level performance analysis and tuning, though implementation may be somehow difficult
- Examples
  - Cache misses
  - Branch mispredictions
  - Memory latency and bandwidth



Data access

- Across nodes and to main memory
- Data access time is measured in terms of bandwidth and latency
- Stream benchmark
- Latency is the startup time for memory, important for short memory transfers
- Time = latency + lenght/bandwidth

- Solution – cache memory and multithreading

Narzędzia dostępne na zasobach PCSS

Intel Parallel Studio XE 2017 (najnowsza dostępna wersja)

- Kompilatory

  C++

  C++ 11, C99, OpenMP 4.1, automatyczna wektoryzacja

  Fortran

  Fortran IV/77/90/95/2003/2008, DO CONCURRENT, OpenMP 4.0

- Biblioteki

  Intel® Math Kernel Library (Intel® MKL)

  zestaw zoptymalizowanych funkcji z dziedzin algebry liniowej (BLAS, LAPACK), transformat Fourier'a, sieci neuronowych, funkcji statystycznych, zagadnień własnych

  Rogue Wave ® IMSL ® Fortran Numerical Library

  zestaw zoptymalizowanych komercyjnych bibliotek z dziedziny analizy numerycznej oraz statystyki

  Intel® Data Analytics Acceleration Library (Intel® DAAL)

  biblioteka zawierająca funkcje wykorzystywane w uczeniu maszynowym, realizujące wszystkie etapy analizy danych (preprocessing, transformacje, modelowanie, weryfikację)

  Intel® Integrated Performance Primitives (Intel® IPP)

  zestaw algorytmów do przetwarzania sygnałów, obrazów, kompresji, kryptografii (możliwa integracja z biblioteką OpenCV)

  Intel® Threading Building Blocks

  model tworzenia programów równoległych w języku C++, dostarcza mechanizmów do allokacji pamięci, zarządzania przepływem danych i operacji synchronizujących w środowisku równoległym na platformach heterogenicznych

Narzędzia dostępne na zasobach PCSS

Intel Parallel Studio 2017 XE (najnowsza dostępna wersja)

- Narzędzia

Intel® VTune™ Amplifier XE

profiler aplikacji równoległych (CPU/GPU) wyposażony w interfejs GUI dający możliwość analizy aplikacji pod względem wykorzystania CPU, skalowalności, dostępu do pamięci oraz wykorzystania jednostek wektorowych

Intel® Inspector

debugger kodu ułatwiający znajdowanie błędów związanych synchronizacją kodu wielowątkowego oraz dostępem do pamięci

Intel® Advisor

narzędzie wspomagające optymalizację kodu pod kątem wektoryzacji oraz przetwarzania wielowątkowego, wskazuje miejsca oraz metody które pomogą zwiększyć wydajność oraz skalowalność aplikacji

Intel® Trace Analyzer and Collector

profiler dla aplikacji równoległych wykorzystujących wymianę komunikatów (MPI), dostarcza zestawu statystyk pozwalających na analizę komunikacji oraz balansu obciążenia aplikacji, dzięki małemu narzutowi na wydajność oraz możliwości pracy wsadowej umożliwia badanie wielkoskalowych aplikacji (>10k procesów).

Przegląd narzędzi wspomagających programowanie (2017.05.31-06.01)

- 2-dwu dniowe szkolenie dotyczące narzędzi Intel dostępnych na klastrze **Eagle**
- Intel Parallel Studio XE
- Intel C++ & Fortran Compilers
- OpenMP 4
- Intel MKL
- Intel Vtune
- Miejsce: PCSS
- Zapisy: **https://szkolenia.man.poznan.pl**

Performance modelling

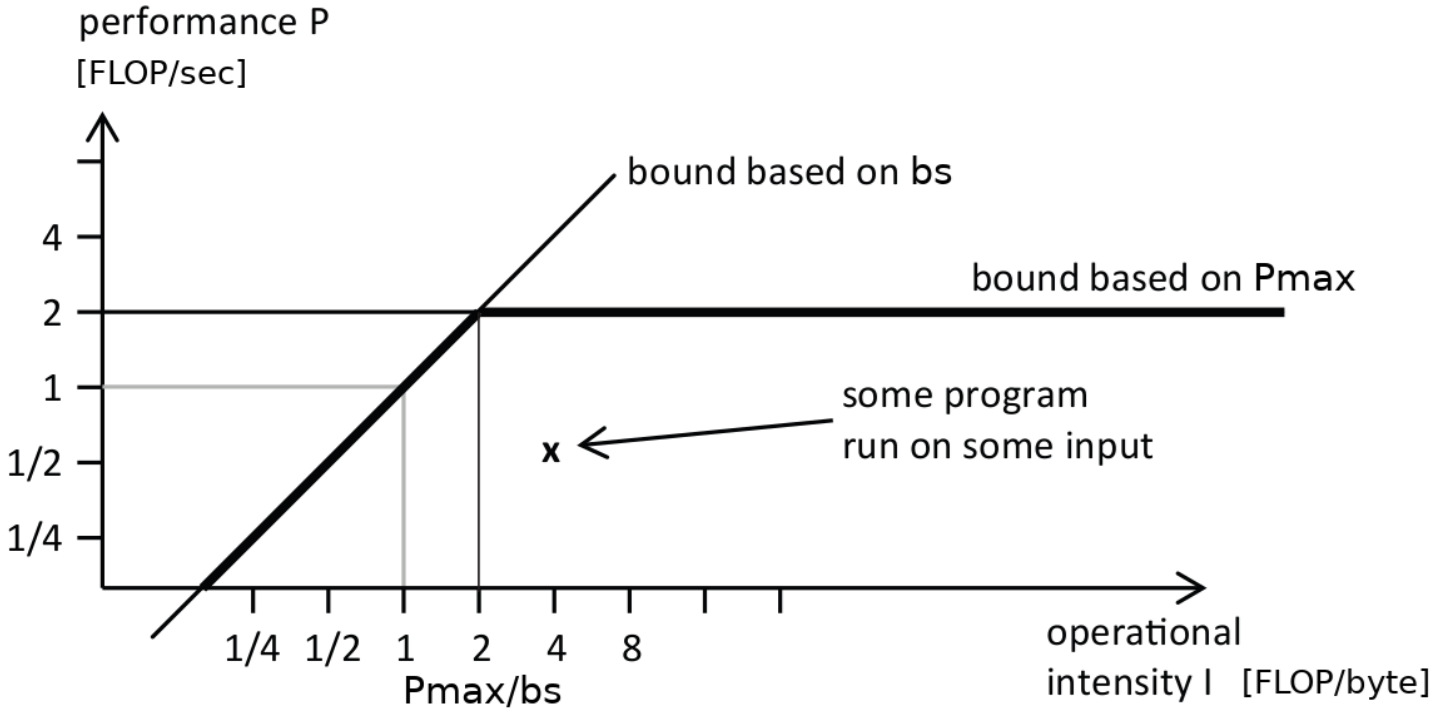
- How to measure performance
  - Execution time
  - Energy usage
- Different models to choose
- The old way (when data movement was not so expensive) : FLOPS/s – how many floating points operation per second
- Performance may be memory-latency or memory-bandwidth bound rather than compute-bound
- Performance modelling is used to determine best possible performance on your hardware (does execution time tell it?)

The Roofline Model

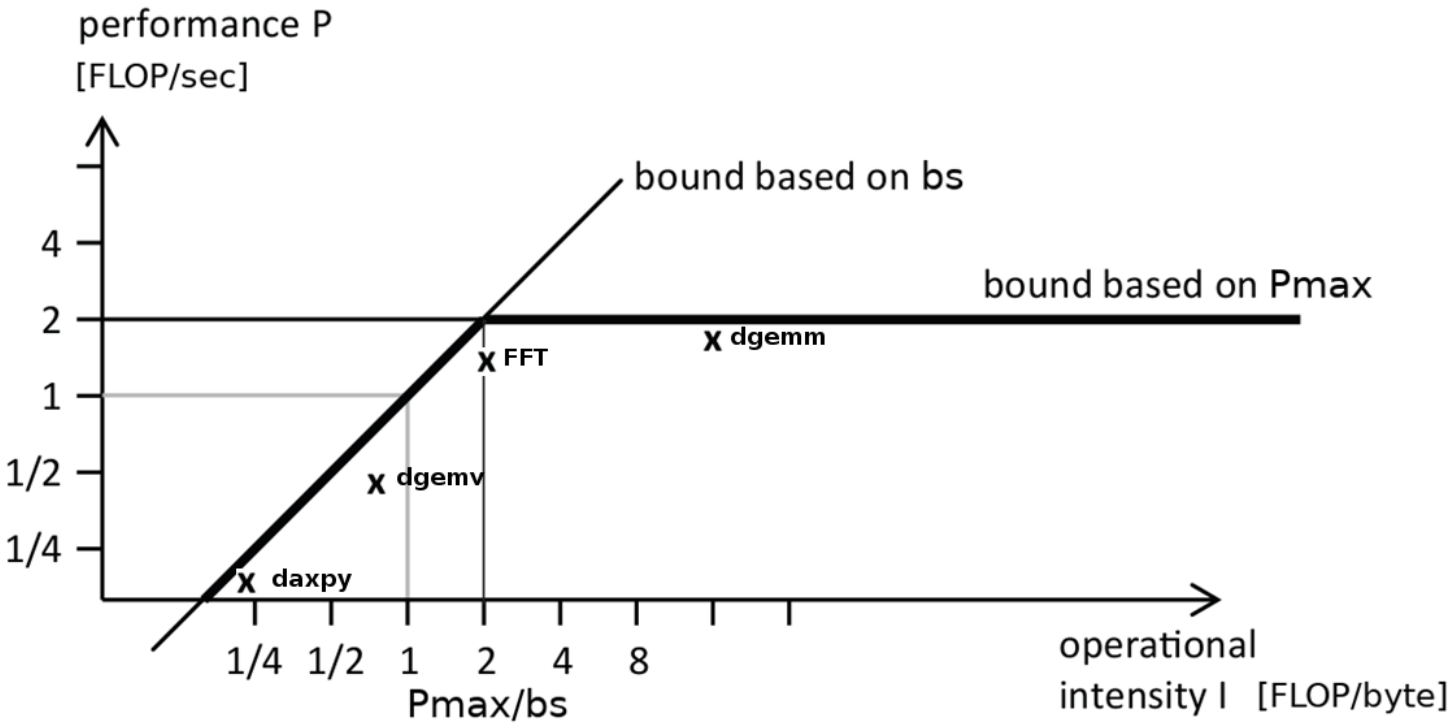
- P_{max} = Applicable peak performance, assuming data comes from L1 cache
- I = Computational intensity ("work" per byte transferred) over the slowest data path utilized ("the bottleneck")
  - I = W/Q
  - W – FLOPs
  - Q - bytes

- b_s = Applicable peak bandwidth of the slowest data path utilized
- Estimated performance:
  - $P = \min(P_{max}, I * b_s)$

The Roofline Model



The Roofline Model



The Roofline Model

Methodology

- To construct roofline plots we need to measure three code specific quantities: W, Q and T
- Measuring W and Q
- How to measure it?
  - Code analysis
  - Hardware Performance Counters
  - Instrumentation

Code analysis

Methodology

- Consider the daxpy routine

```
for(j = 0; j < 10; j++)
for(i = 0; i < n; i++)
        c[i] = a[i] + b[i] * scalar
```

- Quantities for the double precision

W = 2n

Qr >= 16n

Qw >= 8n

Q = Qr + Qw >= 24n

I = 2/24 = 1/12

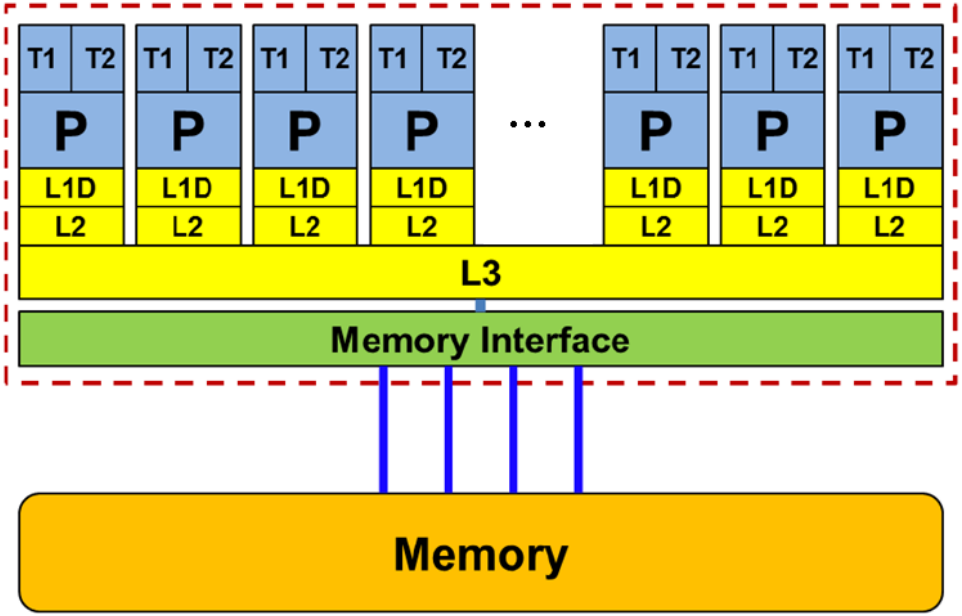
- For n = 10^8

W = 2 GFLOP

Q >= 24 GB

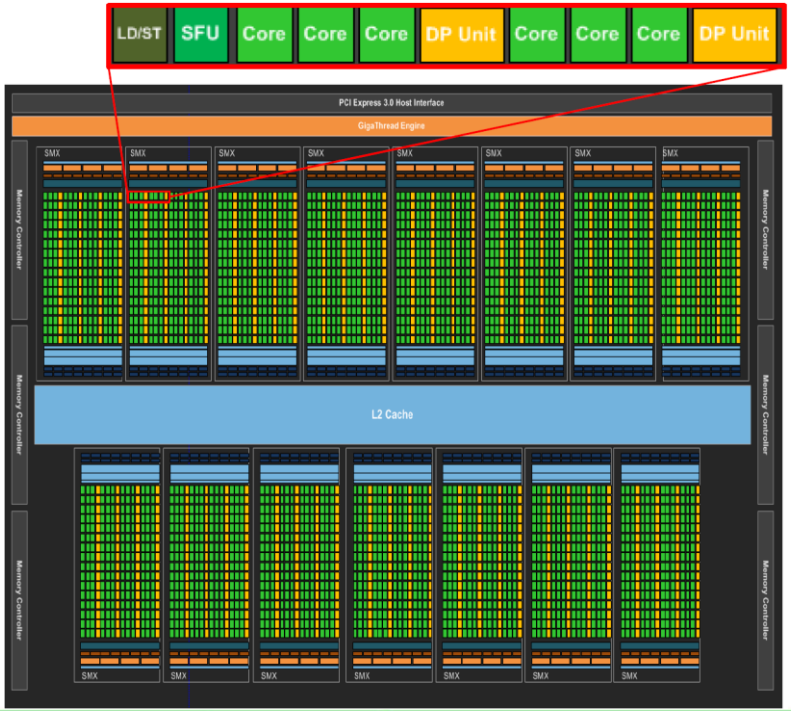
Intel Xeon CPU E5-2697 v3

- Haswell 14 cores 1.2 - 2.6 GHz (3.2 - 3.6GHz turbo)
- 32 kB L1 data cache per core
- 256 kB L2 cache per core
- 35 MB L3 cache per chip
- DDR4 memory interface
- 256 bit SIMD FP unit - AVX
- ~325 GFLOP/s DP peak



Nvidia Tesla K20m

- Kepler GK110 K20m 2496 cores 0.32 - 0.7 GHz
- 13 SMX units with 192 cores each
- 1.25 MB L2 cache
- 1.17 TFLOP/s DP peak
- PCI Express
- 3:1 SP:DP performance



CPU vs GPU

	Intel Xeon E5-2697 v3 Haswell	Nvidia K20m Kepler
Process size	22 nm	28 nm
Transistor count	5.69 Billion	7.1 Billion
Release date	Q3 2014	Q1 2013
Cores@Clock	14 @ 2.9GHz AVX	2496 @ 0.7 GHz
Performance/core	23.2 GFLOP/s	0.47 GFLOP/s
Total performance	324.8 GFLOP/s	1170 GLOP/s
Stream BW	68 GB/s	173 GB/s (ECC=0)
TDP	145 W	225 W
Performance/Watt	2.24 GFLOP/W	5.2 GFLOP/W

Code analysis

Methodology

- On Intel Xeon E5-2697 v3

b_s = 68GB/s

I = 2/24 = 1/12

$P = \min(P_{max}, I * b_s)$

$I * b_s = 1/12 * 68 =$ **5.66 GFLOP/s (1.7% of peak performance)**

P_{max} = 324.8 GLOP/s (14 FP units x 2 x 4 FLOP/cy x 2.9 GHz)

P = min(324.8, 5.66) = 5.66 GFLOP/s

Code analysis

Methodology

- On Kepler K20m

b_s = 173GB/s

I = 2/24 = 1/12

$P = \min(P_{max}, I * b_s)$

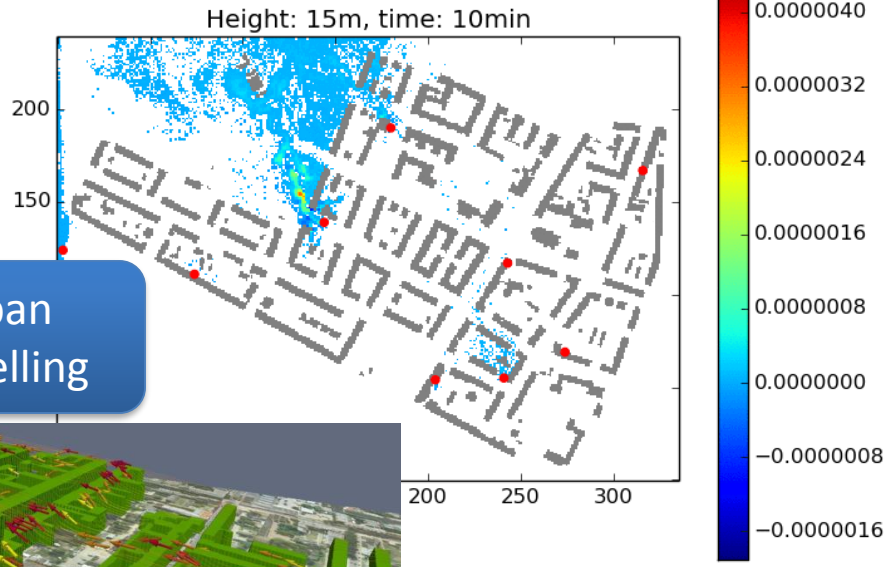
$I * b_s = 1/12 * 173 =$ **14.41 GFLOP/s (1.2% of peak performance)**

P_{max} = 1170 GLOP/s (13 SMX x 64 DP units x 2 FLOP/cy x 0.705 GHz)

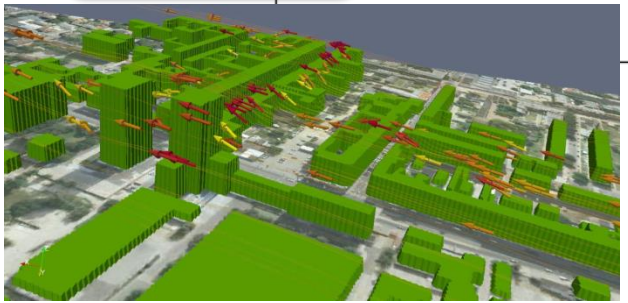
P = min(1170, 14.41) = **14.41 GFLOP/s**

EULAG activities

Adaptation to novel hardware architectures

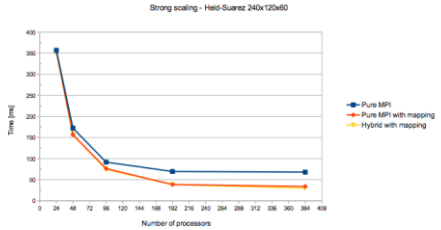


Urban modelling



Contamination dispersion

Hybrid MPI+OpenMP parallelisation



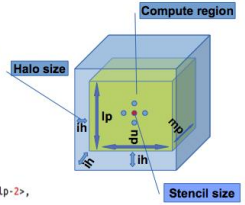
Inula cluster - AMD Opteron 6234 with 24 cores per node.

Current work - mutli-CPU and multi-GPU parallelisation

- Stencil framework - single code base for CPU and GPU

```
struct StencilFunc {
  DEFINE_DO(...arguments...) {
  // Computations for each cell
  }
};

stencil_function<
  Type,
  DomainSize<np,mp,lp>,
  HaloSize<th,th,th,th,th,th>,
  StencilSize<1,1,1,1,1,1>,
  BlockSize<32,8,1>,
  StencilFunc,
  ComputeRegion<1,np-2,1,mp-2,1,lp-2>,
  Cache<true, size>
  > (...arguments...);
```



Grant of the Polish National Science Center

Project description

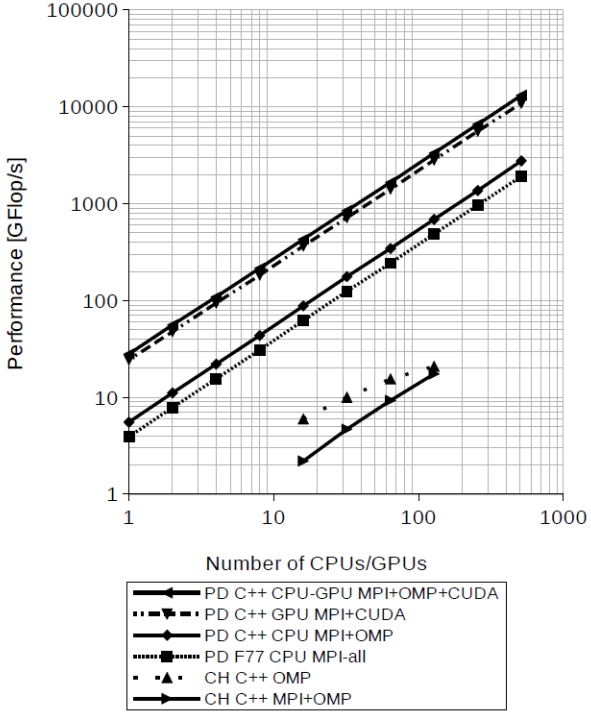
- „Methods and algorithms for organization of computations in the class of anelastic numerical models for geophysical flows on modem computer architectures with realization in the EULAG model”
- Goal:
  - parallelize GCR solver and 3D MPDATA algorithm on modern hybrid CPU-GPU clusters and NUMA architectures
- Consortium:
  - Czestochowa University of Technology
  - Institute of Meteorology and Water Management - National Research Institute
  - Poznan Supercomputing and Networking Center

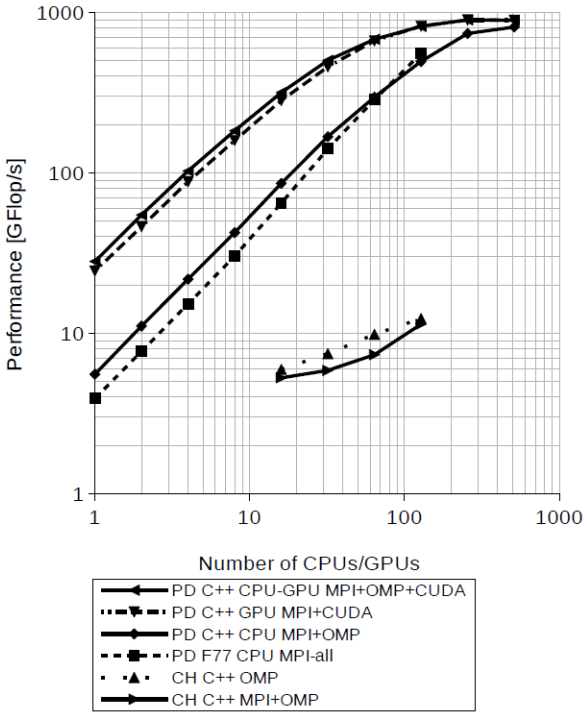
Multi-CPU and multi-GPU performance scaling

Project description

Architecture	Speedup
CPU -> opt CPU	1.4
CPU -> GPU	6
CPU -> opt CPU+GPU	7

Weak scaling





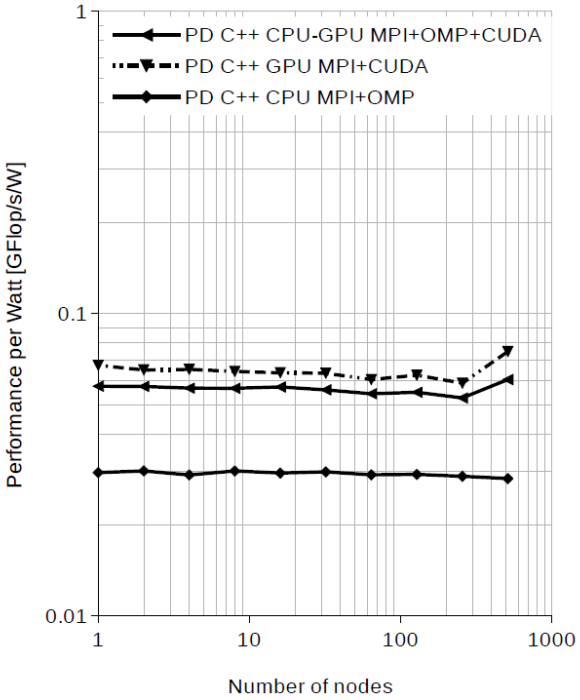
Strong scaling

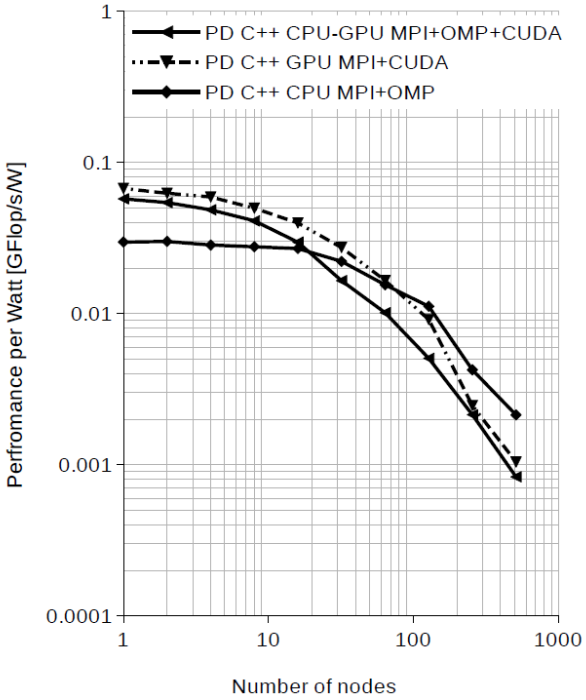
Multi-CPU and multi-GPU performance per Watt scaling

Project description

Architecture	Power efficiency
opt CPU -> GPU	2.13
opt CPU -> optCPU+GPU	1.88

Weak scaling





Strong scaling

Pytania

THANK YOU

Poznan Supercomputing and Networking Center

affiliated to the Institute of Bioorganic Chemistry of the Polish Academy of Sciences,

ul. Noskowskiego 12/14, 61-704 Poznań, POLAND,
Office: phone center: (+48 61) 858-20-00, fax: (+48 61) 852-59-54,
e-mail: office@man.poznan.pl, http://www.psnc.pl