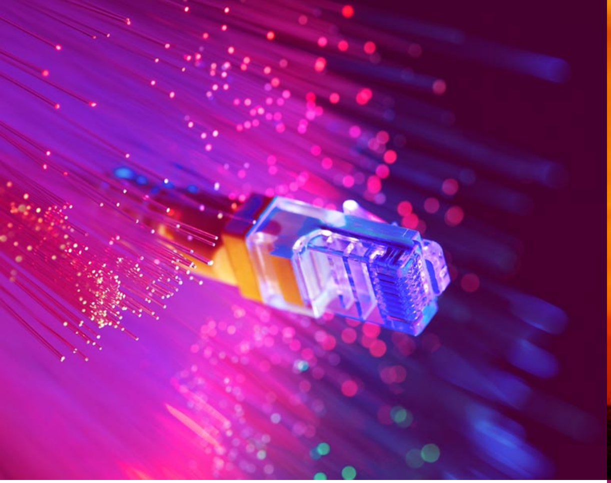


GPAW performance optimisation and energy consumption on KNLs

Martti Louhivuori, CSC

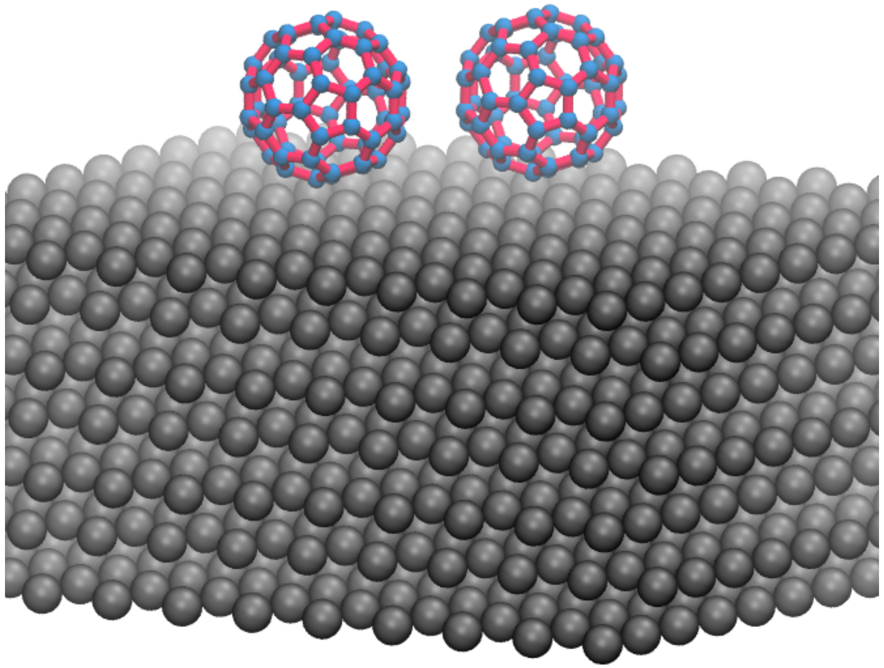
IXPUG spring 2018, Bologna 2018-03-05

CSC – Finnish expertise in ICT for research, education and public administration

GPAW

- Density-functional theory (DFT) program for ab initio electronic structure calculations
- Code written mostly in Python
  - computational kernels in C
  - leverages external libraries (ScaLAPACK etc.)
- Parallelisation based on message-passing (MPI)
- Code freely available under GPL:

  `https://gitlab.com/gpaw/gpaw`



C60 fullerenes next to a Pb sheet

PRACE Accelerator Benchmarks

`www.prace-ri.eu/ueabs/`

- Unified European Applications Benchmark Suite (UEABS) developed by PRACE contains now benchmarks also aimed at accelerators
- For GPAW there are two benchmarks:
  - Small case: Carbon Nanotube (up to ~10 nodes)
  - Large case: Copper Filament (up to ~100 nodes)
- Both are ground state calculations in vacuum, but the *Copper Filament* benchmark is more computationally intensive (and able to scale up better)

Performance and Optimisation

Hardware

- ARCHER Knights Landing Testing and Development Platform by Cray
  - single 64-core KNL processor (Intel Xeon Phi 7210) running at 1.3GHz at each node
  - 96GB of standard memory per node
  - 16GB of high-bandwidth MCDRAM memory per KNL
  - 12 nodes in total, of which 8 in cache mode and 4 in flat mode
- Results compared to CSC's Sisu supercomputer (Cray XC40)
  - two 12-core Haswell CPUs (Intel Xeon E5-2690v3) running at 2.6GHz at each node
  - 64GB of standard DDR4 memory per node

Compiling Python and GPAW

- ARCHER's KNL system has Sandy Bridge login nodes, so GPAW and the underlying Python stack need to be built in two steps
  - Intel compiler (17.0.0) used for everything
  - Cray compiler wrapper (`cc`) takes care of correct compiler options (e.g. `-xMIC-AVX512` on KNLs)
- Python+
  - target SNB (`module load craype-sandybridge`)
- GPAW
  - target KNL (`module load craype-mic-knl`)
  - memkind module is needed to get support for the high-bandwidth memory (`module load cray-memkind`)

Compiling Python and GPAW

- Intel TBB + huge pages
  - using huge pages together with the memory allocator from Intel TBB (tbbmalloc) allow for more optimised memory allocation on KNLs
  - for GPAW, performance increase is up to 5%
  - size of huge pages is not significant for GPAW (2M pages were used)
  - environment variable LD_PRELOAD was used to swap the standard memory allocator with the one from Intel TBB (no code modifications!)

Performance comparison, Haswell vs. KNL

GPAW runtimes (in seconds) with *n* nodes

		1	2	4	8
Carbon Nanotube	Xeon E5-2690v3 x2	242.2	148.5	81.1	55.4
	Xeon Phi 7210*	319.9	206.6	141.3	101.3
Copper Filament	Xeon E5-2690v3 x2	405.0	191.5	86.9	60.5
	Xeon Phi 7210*	323.4	172.3	127.0	80.0

*using tbbmalloc and 2M huge pages
in CACHE / QUAD mode

- KNLs faster than CPUs for the *Copper Filament* benchmark when using one or two nodes
- Compared to CACHE mode, FLAT mode is over 50% slower (data not shown)

Code modifications

- Profiled with VTune Amplifier 2017 on KNLs and potential targets for optimisation were identified in the C kernels
  - triple nested loops with single step pointer incrementations to advance the position of input and/or output arrays
- Code changes:
  - use OpenMP SIMD pragmas
  - use explicit indexing instead of pointer incrementation OR do pointer incrementation in larger blocks at an upper loop level
- Allowed for better vectorisation of the loops by the compiler

Merged to code base

- Obsolete, unnecessary code sections were also identified in the iterator and were removed

Merged to code base

Example code modifications to a kernel

c/bmgs/fd.c

vectorisable?

vectorisable!

```
for (int i1 = 0; i1 < s->n[1]; i1++)
  {
    for (int i2 = 0; i2 < s->n[2]; i2++)
      {
        T x = 0.0;
        for (int c = 0; c < s->ncoefs; c++)
          x += aa[s->offsets[c]] * s->coefs[c];
        *bb++ = x;
        aa++;
      }
    aa += s->j[2];
  }
```

Example code modifications to a kernel

c/bmgs/fd.c

① Explicit indexing in two inner-most loops

② Pointer incrementation at the outer loop level

③ OpenMP SIMD pragma to guide vectorisation of the two inner-most loops

```
     for (int i1 = 0; i1 < s->n[1]; i1++)
       {
+#pragma omp simd 3
         for (int i2 = 0; i2 < s->n[2]; i2++)
           {
             T x = 0.0;
             for (int c = 0; c < s->ncoefs; c++)
-              x += aa[s->offsets[c]] * s->coefs[c];
-            *bb++ = x;
-            aa++;
+ 1            x += aa[s->offsets[c] + i2] * s->coefs[c];
+            bb[i2] = x;
           }
-        aa += s->j[2];
+ 2      bb += s->n[2];
+        aa += s->j[2] + s->n[2];
       }
```

Effects of OpenMP SIMD pragmas and explicit indexing

GPAW runtimes (in seconds) and performance increase with *n* KNLs

		1	2	4	8
Carbon Nanotube	reference	319.9	206.6	141.3	101.3
	optimised	269.3	177.3	123.2	91.3
	speed-up	*1.188*	*1.165*	*1.147*	*1.110*
Copper Filament	reference	323.4	172.3	127.0	80.0
	optimised	280.7	150.0	116.1	74.0
	speed-up	*1.152*	*1.149*	*1.094*	*1.081*

up to 15-18% speed-up

load inbalance between the MPI tasks is now the main bottleneck

Reference on CPUs		1	2	4	8
Carbon Nanotube	Xeon E5-2690v3 x2	242.2	148.5	81.1	55.4
Copper Filament	Xeon E5-2690v3 x2	405.0	191.5	86.9	60.5

GPAW on Skylake

GPAW runtimes (in seconds) on a single SKL node compared to HSW and KNL

	AVX-512	AVX-512*	AVX2	AVX2*	HSW	KNL*
Carbon Nanotube	116.2	102.4	118.7	118.3	242.2	269.3
Copper Filament	156.8	153.7	150.3	150.1	405.0	280.7

*use code optimisations (OpenMP SIMDs & array indexing)

- Dual 26-core Skylake @ 2.1 GHz (Intel Xeon Platinum 8170) with 192 GB of DDR4 memory
  - HSW: dual 12-core Haswell @ 2.6 GHz (Intel Xeon E5-2690v3)
  - KNL: single 64-core Knights Landing @ 1.3 GHz (Intel Xeon Phi 7210)
- All results on Skylake and Knights Landing using tbbmalloc and 2M huge pages

Conclusions on performance

see full report at: `github.com/cschpc/gpaw-on-KNL`

- GPAW achieves similar performance on KNLs as on dual-CPU Haswell nodes, but with poorer scaling properties
  - Benchmarks with higher computational burden fare better on KNLs and also show better scaling properties
- Best performance achieved when using CACHE mode for MCDRAM and tbbmalloc with huge pages
- Some performance improvement (up to 18.8%) was achieved on KNLs by using OpenMP SIMDs and array indexing on three computational kernels

Energy consumption

Hardware and software used for energy measurements

- PRACE Pre-Commercial Procurement (PCP) for energy efficient HPC solutions
  - Atos-Bull KNL pilot system (at CINES/GENCI) ←
  - E4 Power-8/Pascal pilot system (at CINECA)
  - Maxeler data-flow pilot system (at JUELICH)
- Atos-Bull KNL pilot system
  - 56 Atos-Bull Sequana X1210 blades + water-cooled power
  - 168 compute nodes with a single 68-core KNL (Intel Xeon Phi 7250) + HDEEM FPGA for energy monitoring
- Bull Energy Optimizer (BEO)
  - energy monitoring of the whole system (at 100Hz)
  - HDEEVIZ may be used for more detailed profiles (at 1kHz)

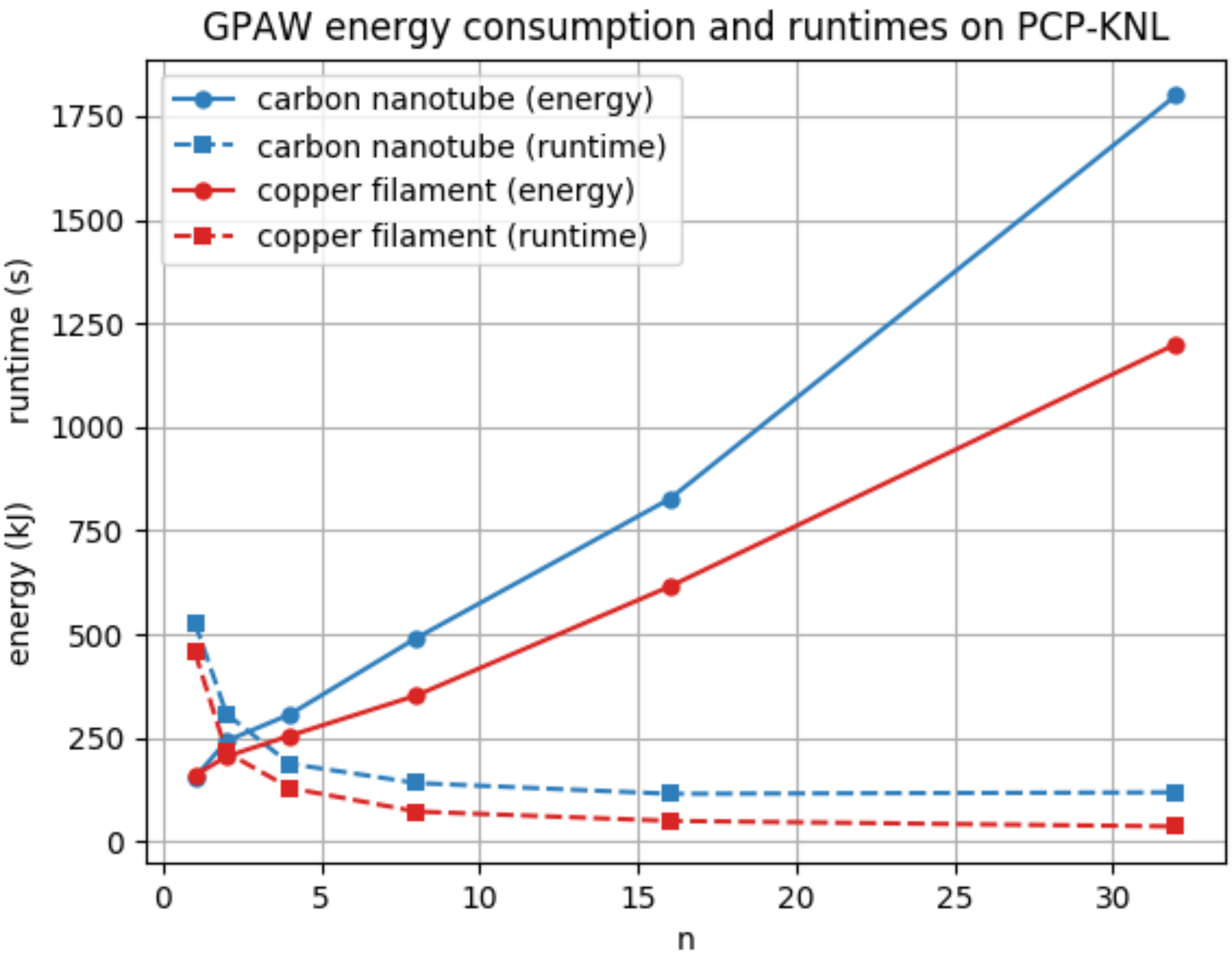
Energy consumption & runtimes on PCP-KNL

GPAW energy consumption and runtimes on KNLs with *n* nodes

		1	2	4	8	16	32
Carbon Nanotube	energy (kJ)	154	243	307	491	826	1800
	runtime (s)	527.3	307.2	187.3	140.8	114.8	118.3
Copper Filament	energy (kJ)	159	205	255	352	615	1200
	runtime (s)	456.5	214.8	128.7	72.0	49.5	36.0

FLAT / QUAD mode

- Energy consumption seems to scale linearly with the number of KNLs used
- Absolute scaling limit reached for the *Carbon Nanotube* benchmark (< 32 KNLs)


GPAW energy consumption and runtimes on PCP-KNL
carbon nanotube (energy)
carbon nanotube (runtime)
copper filament (energy)
copper filament (runtime)
runtime (s)
energy (kJ)
n
0
250
500
750
1000
1250
1500
1750
0
5
10
15
20
25
30

Conclusions on energy consumption

- Energy consumption seems to grow linearly with the number of KNLs in use
  - maximum energy efficiency for runs with only a single KNL
- Minimum energy to solutions for the PRACE accelerator benchmarks on the PCP-KNL in FLAT/QUAD mode:
  - Carbon Nanotube: (154 +/- 3) kJ
  - Copper Filament: (159 +/- 3) kJ
- Slightly lower energy to solution results expected for KNLs running in CACHE mode instead of the FLAT mode (simply due to faster run times)

CSC

Dr. Martti Louhivuori

HPC Programming Support
CSC – IT Center for Science Ltd.

martti.louhivuori@csc.fi

https://www.facebook.com/CSCfi

https://twitter.com/CSCfi

https://www.youtube.com/c/CSCfi

https://www.linkedin.com/company/csc---it-center-for-science