KNL Performance Comparison: GPAW

April 2017

1. Compilation, Setup and Input

Compilation

Standard version of GPAW should be compiled using the Intel compile environment with Intel MKL and Intel MPI, e.g. by following the generic installation instructions for GPAW (https://wiki.fysik.dtu.dk/gpaw/install.html).

In this comparison, GPAW version 0.11.0 was used and the results from the 64-core KNLs (Xeon Phi 7210) were compared to results from 12-core Haswell CPUs (Xeon E5-2690v3). A single KNL was compared to a full node (two CPUs) to have comparable power consumptions.

Python+

Since ARCHER's KNL system has Sandy Bridge login nodes, one needs to build GPAW and it's underlying Python stack in two steps. First, Python and the other dependencies of GPAW should be built targeting the Sandy Bridge CPUs.

```
module swap PrgEnv-cray PrgEnv-intel/6.0.3
export CC=cc
export MPICC=cc
export CRAYPE_LINK_TYPE=dynamic
export CRAY_ADD_RPATH=yes

module swap craype-mic-knl craype-sandybridge

# … install Python etc.
```

GPAW

After Python and the other dependencies are built, one can switch to target the KNLs and proceed to build GPAW.

```
module swap craype-sandybridge craype-mic-knl
module load cray-memkind

# … install GPAW
```

The compiler wrapper (cc) takes care of using the correct compiler options for the target architecture. In the case of KNLs, it will add '-xMIC-AVX512' to enable the AVX512 vector sets supported by KNLs. The module 'cray-memkind' is also needed to get support for the high-bandwidth memory.

TBB & hugepages

To improve performance, one should also link to Intel TBB to benefit from an optimised memory allocator (tbbmalloc). This can be done during installation or at run-time by setting environment variable LD_PRELOAD to point to the correct libraries, i.e. for example:

```
export LD_PRELOAD=$TBBROOT/lib/intel64/gcc4.7/libtbbmalloc_proxy.so.2
export LD_PRELOAD=$LD_PRELOAD:$TBBROOT/lib/intel64/gcc4.7/libtbbmalloc.so.2
```

It is also beneficial to use hugepages together with tbbmalloc, e.g.:

```
module load craype-hugepages2M
export TBB_MALLOC_USE_HUGE_PAGES=1
```

As a side note, even though there is a special development version of GPAW aimed at MIC coprocessors (git branch 'mic'), it is actually an offload approach meant for KNCs (and potentially also for any upcoming coprocessors). On stand-alone KNLs, such as the ones in the ARCHER KNL platform, offloading is not needed nor even preferred.

Setup

The ARCHER KNL nodes were used in cache mode (quad_100) with all of the high-bandwidth MCDRAM memory used as an additional cache between the processor and conventional memory.

The results from the 64-core KNLs (Xeon Phi 7210) were compared to results from 12-core Haswell CPUs (Xeon E5-2690v3). A single KNL was compared to a full node (two CPUs) to have comparable power consumptions.

Input

Two test cases (used also in the PRACE Accelerator Benchmark for GPAW) were used to study the performance of GPAW. One of the benchmarks (Carbon nanotube) is aimed at smaller systems (up to 10 nodes) while the other one (Copper filament) is aimed at larger systems (up to 100 nodes). Both test cases were used to study the performance and scaling properties of GPAW up to 8 KNL nodes.

The benchmarks are available at:
https://github.com/mlouhivu/gpaw-benchmarks.git

Default input parameters were used for both benchmarks.

Case 1: Carbon nanotube is a ground state calculation for a carbon nanotube in vacuum. By default it uses a 6-6-10 nanotube with 240 atoms (freely adjustable) and serial LAPACK with an option to use ScaLAPACK. *Input file:* carbon-nanotube/input.py

Case 2: Copper filament is a ground state calculation for a copper filament in vacuum. By default it uses a 2x2x3 FCC lattice with 71 atoms (freely adjustable) and ScaLAPACK for parallelisation. *Input file:* copper-filament/input.py

2. Performance Data

GPAW runtimes were measured using two benchmarks (see above for details) on 64-core KNLs (Xeon Phi 7210) and 12-core Haswell CPUs (Xeon E5-2690v3). Only the runtime for the SCF cycle was used to exclude any differences in the initialisation overheads. A single KNL was compared to a full node (two CPUs) to have comparable power consumptions.

Different configurations were tested to find optimal performance on KNLs for both benchmarks. Summary of the results for Case 1 are shown in Table 1 and for Case 2 in Table 2. As can be seen from the results, the effect of switching to TBB for memory allocation is either negligible (Case 1) or only minor (Case 2) without the additional benefit of hugepages. If hugepages are enabled, performance is increased for both benchmarks regardless of the size of the hugepages.

Table 1. Average runtime in seconds for Case 1 when using *n* KNLs. Data shown for runs using standard memory allocator (*default*) as well as using tbbmalloc without (*TBB*) or with 2M, 4M, 8M, or 16M hugepages (*TBB + 2M* etc.).

n	default	TBB	TBB + 2M	TBB + 4M	TBB + 8M	TBB + 16M
1	329.2	330.0	319.9	320.8	321.1	
2	215.5	213.2	206.6			
3		145.5	141.3			
4			101.3	101.2	101.3	101.4

Table 2. Average runtime in seconds for Case 2 when using n KNLs. Data shown for runs using standard memory allocator (*default*) as well as using tbbmalloc without (*TBB*) or with 2M, 4M, 8M, or 16M hugepages (*TBB + 2M* etc.).

n	default	TBB	TBB + 2M	TBB + 4M	TBB + 8M
1	341.0	337.1	323.4	322.9	323.4
2		177.7	172.3	171.0	170.9
4	133.0	130.7	127.0	127.0	127.2
8		82.2	80.0	80.0	80.2

Additionally, for Case 1 the effect of using ScaLAPACK instead of serial LAPACK (which is the default in Case 1) was tested, but only degrading performance was achieved.

Since the size of the hugepages does not affect performance, results for runs using TBB together with 2M hugepages were chosen for a comparison with Haswell CPUs (Table 3). As can been seen from Table 3, for Case 1 the performance of a KNL is at best 79.4% of that of a Haswell CPU node. In contrast, for Case 2 the performance of a single KNL is 125.5% of that of a Haswell CPU node. When using more KNLs (or nodes), it is clear that KNLs do not scale as well as CPUs and thus already with 4 nodes/KNLs Case 2 is faster on CPUs.

Nevertheless, the results are quite comparable and depending on the system of interest KNLs may offer similar (or even better) performance than Haswell CPUs.

Table 3. Comparison of runtimes on Haswell CPU nodes (*CPU*) and KNLs (*KNL*) for both Case 1 and Case 2. Average runtimes are shown for 1, 2, 4, or 8 nodes consisting of two CPUs or a single KNL.

		1	2	3	4
Case 1	CPU	253.9	136.2	80.7	55.6
	KNL	319.9	206.6	141.3	101.3
Case 2	CPU	405.8	195.2	95.4	60.3
	KNL	323.4	172.3	127.0	80.0

3. Summary and Conclusions

GPAW has been shown to achieve similar performance on KNLs as on dual-CPU Haswell nodes, but with poorer scaling properties. Depending on the benchmark the performance is either slightly worse or better (from 79.4% to 125.5% for a single KNL/node). Unsurprisingly, the higher the computational burden is, the better KNL performs and scales when moving to multiple processors.

Based on the two benchmarks tested, it is recommended for optimal performance to use KNLs for workloads similar to Case 2 (Copper filament) if one is limited to using one or two nodes.