KNL Performance Comparison: COSA

Adrian Jackson
a.jackson@epcc.ed.ac.uk
March 2017

1. Introduction

COSA is a finite volume, compressible, Navier-Stokes simulation code. The core solver in COSA is a harmonic balance Navier-Stokes computational fluid dynamics solver. The harmonic balance method is a nonlinear frequency-domain technique that reduces the runtime for calculating periodic solutions of ordinary differential equations with respect to the conventional time-marching approach.

COSA is used for accurate unsteady aerodynamic analysis of fluid flows and fluid/structure interaction problems (e.g. flow-induced structural vibrations) in renewable energy, mechanical and aeronautical engineering. It is implemented in Fortran, and parallelised with MPI. There are 2D and 3D versions of COSA.

2. Compilation, Setup and Input

Compilation

The latest version of the COSA source code (3D version) was compiled using the Intel compilers (default versions on each login node, so v15.0.2.164 on the main ARCHER system and v17.0.0.098 on the KNL system) and the Cray MPI libraries. These same compile flags were used for both versions (the ARCHER IvyBridge and ARCHER KNL executables):

```
-132 -O3 -I$(MKLROOT)/include -fpp -m64 -r8  -D MPI -fno-fnalias
```

The same link flags were used for both versions of the executable as well:

```
-Wl,--start-group  $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
            $(MKLROOT)/lib/intel64/libmkl_core.a
          $(MKLROOT)/lib/intel64/libmkl_sequential.
               a -Wl,--end-group -lpthread -lm
```

The KNL version was compiled on the ARCHER KNL login nodes, and therefore had the `craype-mic-knl` module loaded (which sets up the correct compiler flags to generate executables for the KNL instruction set, i.e. `-xMIC-AVX512`). The IvyBridge version was compiled on the ARCHER login nodes, and therefore has the module `craype-ivybridge` loaded.

COSA produces a significant amount of output I/O (restart and data files). For these benchmarks the large output I/O was turned off.

Setup

We ran the same test case the standard ARCHER nodes, the KNL quadrant flat (`aoe:quad_0`) nodes, and the KNL quadrant cache (`aoe:quad_100`) nodes. There are only two KNL nodes configured in `aoe:quad_0` mode, so we ran up to two nodes on the KNL system, and up to 34 nodes on the main ARCHER system.

Hyperthreading on the Xeon and Xeon Phi processors were tested for performance but hyperthreading gave no benefit for either system so the results are not presented here.

Input

We used an 800 block simulation with 3,689,952 grid cells (NREL5MW_GRID32_HB_SECTOR). The memory requirements for this simulation means that it cannot fit in MCDRAM (16 GBs) on a single node, or in the MCDRAM on two KNL nodes. The input parameters are provided in

Appendix A. The domain decomposition implemented in COSA splits the 800 blocks as evenly as possible across MPI processes. This means that the following are sensible MPI process counts (i.e. divide the 800 blocks evenly) for this simulation:

- 20
- 25
- 50
- 100
- 200
- 400
- 800

As there are only 2 KNL flat nodes configured on the KNL system we ran up to 100 MPI processes on the KNL nodes, and up to 800 MPI processes on the ARCHER nodes.

When running we aimed to split MPI processes across the available nodes as evenly as possible. This means on the KNL nodes using an `aprun` line like this:

```
aprun -n 100 -N 50 ./cosa.mpi
```

Ensuring that the 100 MPI processes were distributed evenly across the two KNL nodes, rather than having 64 MPI processes on the first node and 36 on the second.

When running on the flat memory mode nodes we ran purely using main memory (DRAM), and we also ran using the MCDRAM using the numactl tool to target that memory. As the use cases does not fit into the 16GB of MCDRAM (or even the 32GB of MCDRAM across the 2 flat nodes) we had to use the `-p` flag to request data be allocated in the MCDRAM in the first instance, falling back to the main memory when the MCDRAM is full (as opposed to the `-m` flag which forces all memory to be allocated in the MCDRAM and fails if the memory is exhausted).

3. Performance Data

The results presented below were collected using 3 independent runs for each data point, with the fastest results presented. There was a less than 5% variation in runtime across the three data points.

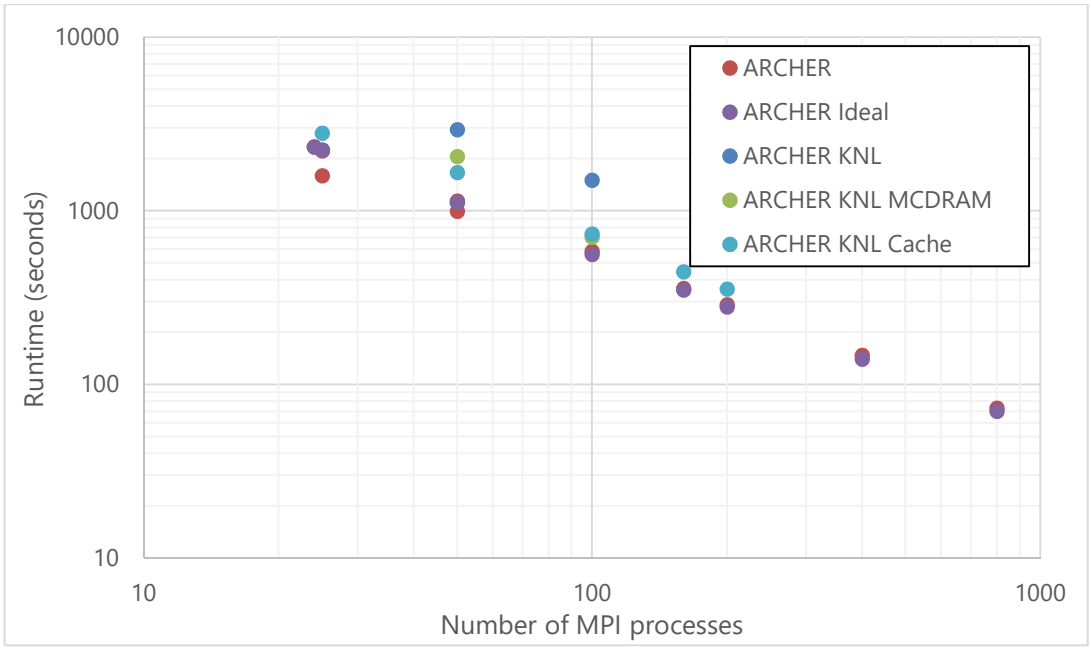

Figure 1: Performance of COSA on IvyBridge and KNL processors, comparing performance on number of MPI processes

Note, in Figure 1 the ARCHER and ARCHER Ideal results are almost identical, hence the reason that the ARCHER data points are not visible. The Ideal data points are calculated by taking the runtime at the lowest process count and then dividing it by the factor of cores extra being used at the rest of the data points (i.e. If the initial data point is on 20 MPI processes, then at 50 MPI processes the Ideal data point is calculated by dividing the time on 20 processes by 50/20).

When underpopulating (using less MPI processes on the node than available cores) we did not try to distribute the MPI processes on the KNL processor, we simply accepted the process binding that `aprun` provides as a default. This means that for small process counts (i.e. 20 MPI processes) it may be possible to boost the performance somewhat by applying a more sensible process binding (i.e. using `-d` flag for `aprun` to distribute MPI processes more evenly across the quadrants on the KNL).

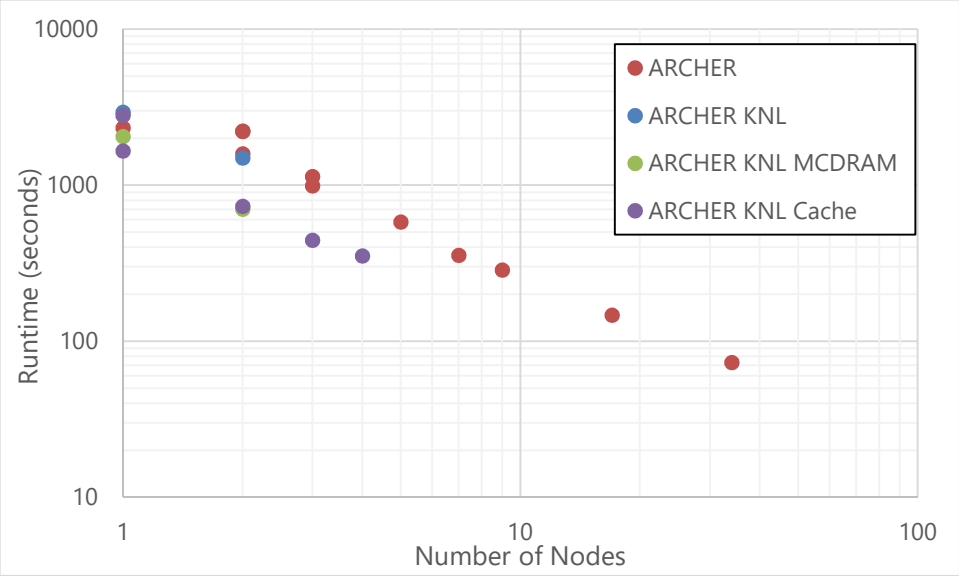

Figure 2: COSA Performance on IvyBridge and KNL processors, comparing number of nodes used

Note, for Figure 2 the KNL MCDRAM and KNL Cache results are very close when using more than one node, hence the reason the KNL MCDRAM points are not visible on 2, 3, and 4 nodes. We only ran on up to 4 KNL nodes due to time constraints compiling this report. The code should scale to a larger number of nodes (MPI scaling is not a significant constraint on COSA with test case, as demonstrated by the ARCHER scaling) but we have not investigate that yet.

Number of Nodes	DRAM	Flat mode: numactl –p 1	Cache mode
1	2933	2054	1658
2	1496	704	733

Table 1: Performance of the different memory configurations of the KNL system. Runtime in seconds.

4. Summary and Conclusions

We can see from Figures 1 and 2 that the IvyBridge benchmarks show very good parallel scaling, with the performance of COSA being almost identical to the ideal performance.

When comparing KNL performance to the ARCHER IvyBridge processors, we can see that on a process to process comparison the IvyBridge is faster (as shown in Figure 1). We can complete this simulation in 1137 seconds on the ARCHER compute nodes using 50 MPI processes, compared to the best KNL node time of 2054 seconds. An initial analysis would show the ARCHER IvyBridge nodes around 1.8x faster than the KNL.

However, as is demonstrated in Figure 2, if we compare on node to node basis, rather than MPI process to MPI process, we can see that the KNL can complete the simulation quicker than the IvyBridge processes. 50 MPI processes, which fit in one KNL node, complete in 2054 seconds, whereas the fastest simulation on one node of the main ARCHER system (which has 24 cores) is 2327 seconds. Furthermore, we can see that 2 KNL nodes can complete the simulation in 704 seconds, compared to 1137 seconds for 3 ARCHER IvyBridge nodes. The KNL system is 1.6x faster when using 2/3 of the nodes of the ARCHER system.

The scaling from one to two nodes on the KNL system is superlinear, with the simulation completing ~3x faster on two nodes than it does on one KNL node. However, this is likely to be the effect of more of the simulation data fitting into the MCDRAM on the KNL, providing a significant performance advantage compared to the single node case.

Indeed, if we look at the performance of the KNL system in Figures 1 and 2 and Table 1 we can see that the MCDRAM gives significant performance benefits for this application. For the single node case using the MCDRAM in flat mode and numactl –p 1 gives a performance benefit of ~1.4x, whereas the MCDRAM used in cache mode gives an even bigger performance benefit of ~1.8x. Moving to two nodes, where more of the simulation data will fit in MCDRAM, we see a different outcome with the manual use of MCDRAM performing better than the cache mode (albeit both give much better performance than not using the MCDRAM and only a small difference in the performance between the different methods, 2.125x faster vs 2.04x faster).

Therefore, we can see that the per process performance of COSA on the KNL is slower than on ARCHER, but on a node for node comparison the KNL is significantly faster, primarily due to the high bandwidth memory. It is possible that for different test cases/benchmarks, where less of the data fits in the MCDRAM, this performance benefit would not be as pronounced.

5. Acknowledgements

This work was supported by Intel through EPCC's IPCC project (Intel Parallel Computing Centre). Fiona Reid (EPCC) kindly provided proof-reading and corrections.

Appendix A

```
Input file for 3D Euler/NS code
debug       model      flow-type  id         nblade
n           sst        external   shawt      3
gamma       reyno      pranl      machfs     alpha     beta
1.4d0       7.7d+5     0.71d0     0.0335     0.00      20.00
prant       tkefar     mutfar     wall       roughk
0.9d0       1.d-4      0.1        menter     0.d0
posprd      lim_ptke   prdlim     lim_pome   pr.type   turb.ord.
n           y          10         y          minimum   second
flow-mode   solver     rk option  nharms
unsteady    hb         rkex       4
move        freq.      xrotc      yrotc      frame
rotating    -0.000593  0.d0       0.d0       relative
irest       srest      cfl        cdff       lmax      iupdt     toler
0           5000       1.5        4          20        1         1.d-12
rkap        irs-typ    cfli       cutcirs    psi
-1.         cirs_v1    3.0        0          0.0625
cflt        cflit      ramp-opt   n(3)       n(2)      n(1)
2.0         4.0        ramping1   2000       1000      500
cfli(2)     cflit(2)   cfli(1)    cflit(1)
1.5         2.0        0.1        0.1
lim         epslim     cntrpy     etpfxtyp   entfxctf
4           1.d-6      1.d0       0          0.3d0
nlevel      nl_crs     nl_fmg     nstart     npre      npost     ncrs
1           4          1          2          3         2         2
flow-speed
```

```
nolomach
tref
288.2
functional
default 0.0 0.0 0.0
lref1       lref2       lref3
1.0         1.0         1.0
1
76  33 37 41 42 49 50 57 58 125 126 161 165 169 170 197 198 233 234 241 242
341 342 351 356 357 362 371 372 385 390 395 396 405 406 413 418 455 460 46
5 470 475 480 481 486 499 504 505 510 519 524 529 534 596 601 606 611 616
621 626 631 636 640 645 650 662 667 672 676 681 686 700 705 710 715 720 725
800
```

Appendix B

Timings data that the graphs in this report are based on.

ARCHER IvyBridge:

Number of Nodes Used	Number of Cores Used	Runtime (seconds)	Ideal runtime (seconds)
1	20		
1	24	2327	2327
2	25	1591	
2	25	2212	2233.92
2	25	2225	2233.92
3	50	1137	1116.96
3	50	992	1116.96
5	100	583	558.48
7	160	356	349.05
9	200	287	279.24
17	400	147	139.62
34	800	73	69.81

ARCHER KNL:

Number of Nodes Used	Number of Cores Used	KNL Performance	KNL Performance using MCDRAM directly	KNL Performance using Cache mode
1	20			
1	24			
1	25			
1	25			
1	25			2799
1	50	2933	2054	1658
1	50			
2	100	1496	704	733
3	160			445
4	200			353