KNL Performance Comparison: HMB

August 2017

G.N. Barakos and M.A. Woodgate
George.Barakos@Glasgow.ac.uk   Mark.Woodgate@Glasgow.ac.uk

CFD Laboratory
School of Engineering
University of Glasgow
Glasgow, G128QQ, Scotland, Uk

1. Compilation, Setup and Input

Compilation

The code was compiled using the Intel C++ compiler on both KNL and Xeon nodes. Version 17.0.0.098 was used on the KNL while the default version 15.0.2.164 was used on the Xeon. Currently we have no resources left on the Xeon to compare the performance of the difference between the two versions of the Intel compiler but it is not expected to have a large effect on the runtime of the code. The compiler optimization option used was -O3 and the compiler directives for the source code where -D_NEW_ALLOC -D_METHOD2 which is the most efficient memory layer currently implemented in the parallel code.

Setup

The code uses MPI for parallel communication and was run with up to 24 processes per node on the Xeon and 64 processes per node on the KNL. The performance across nodes shows linear speedup from the Xeon but, as it is currently only possible to run on two KNL nodes, there are not enough data points to draw any conclusions on the performance across nodes for the KNL system. The memory option used on the KNL system was quad_100 where all the MCDRAM is used to cache the main memory.

Input

Two three-dimensional benchmarks were used. One was used to test the parallel performance on a small number of cores (less than 576) and the other was to test scaling on a much larger number of cores (up to 100,000). The first test problem is turbulent flow around a cube with 576 blocks each of size 21x21x21 while the second is a cavity calculation with 65536 Blocks each of size 21x21x21. Both problems were fully turbulent and solved using Osher's approximate Riemann solver with MUSCL extrapolation and the k-ω two equation turbulence model.

2. Performance Data

Table 1 shows the effect of block size on the performance of HMB on 6 cores on a local compute node (Glasgow K-Node system) for one implicit iteration. For the 6-block case, the boundary conditions and the halo exchange were substantially fast. However, this is a minor part of the cost of an iteration, and even though the amount of work for each case is the same, the iteration is 10% faster on smaller blocks. This is due to the block better fitting into the cache on the processors and can be thought of adding tiling within the bigger blocks.

Computational subroutines	576 21x21x21 blocks	6 81x81x121 blocks
Boundary conditions and Halo	0.038357s	0.008444s
Initialize and time-step	0.221877s	0.217172s
Jacobian and Residual	3.548065s	4.003661s
Preconditioner	0.716755s	0.781016s
Linear solver	1.893830s	2.116967s
Total time	6.448413s	7.155373s

Table 1: Performance of different parts of HMB using the same size mesh split into different block sizes running on 6 cores.

Table 2 uses the same local cluster and shows the good scaling of the code for up to 16 cores for both the implicit and explicit iteration schemes.

Glasgow K Nodes (Cores)		CPU time per explicit iteration	CPU time per implicit iteration
1		9.663s	33.18s
2		4.849s	16.10s
4		2.548s	8.959s
6		1.897s	6.487s
8		1.611s	5.469s
16		0.8975s	3.089s

Table 2 Performance data for running HMB on different numbers of cores on a Glasgow K Node.

Table 3 shows the results of the second test case running on ARCHER using up to 16384 cores. The code scales very well with a 16 fold increase in cores producing a 15.32 reduction in iteration time.

Archer Cores (Nodes used)	CPU per explicit iteration	CPU per Implicit Iteration
Half Cavity Calculation		
1024	1.3995s	5.2401s
2048	0.7414s	2.7589s
4096	0.3681s	1.3898s
8192	0.1855s	0.6773s
16384	0.0989s	0.3419s

Table 3 Performance data for running HMB on testcase 2 on ARCHER

Table 4 shows the performance of HMB on the ARCHER KNL nodes. The single core performance is very weak compared to an Intel Xeon core even considering the lower clock frequency. For example, it is over 4 times slower than a single Cirrus core. However, the parallel efficiency scales well for up to 64 cores making good use of all the extra cores. Hence when comparing a fully loaded KNL node vs a fully load Cirrus node the performance gap drops to around 30%. The scaling for KNL is similar to the scaling shown for the other HPC systems.

KNL quad_100 Cores	CPU time per explicit iteration	CPU time per implicit iteration
1	35.47s	96.03s
2	18.80s	49.46s
4	9.464s	24.87s
8	4.762s	12.35s
16	2.408s	6.298s
32	1.206s	3.176s
64	0.6375s	1.770s
192 (3 Nodes)	0.2198s	0.5945s

Table 4: Performance data for running HMB on different numbers of cores on an ARCHER KNL compute node in quad_100 configuration using testcase 1.

Table 5 shows the performance of HMB on Cirrus. It should be noted here that the GCC 6.2 compiled code produced very inconsistent results with the times of both the explicit and implicit iteration varying by up to a factor of 4 per iteration. It was found that this effect was worst when running on just a few cores within a compute node. However, even the case of a fully loaded computer node produced results where the CPU time per iteration varied more than normal – there are minor differences in iterations times depending on which code branch is followed in Osher's Riemann solver. Hence the results present here are for the intel compiler 17.0.2. The single CPU scaling is reasonable up until about 12 cores, as has been seen on ARCHER nodes, but increasing the number of cores above this has very little effect on the CPU time of an iteration. The external node scaling is good with a sixteen-fold increase in cores producing a 14.85 reduction in CPU time per iteration.

Cirrus Cores (Nodes used)	CPU per explicit iteration	CPU per Implicit Iteration
1 (1)	6.578s	21.926s
2 (1)	3.337s	11.270s
4 (1)	1.763s	6.453s
8 (1)	1.005s	3.768s
16 (1)	0.6576s	2.754s
18 (1)	0.6373s	2.699s
36 (1)	0.3203s	1.364s
Half Cavity Calculation		
256	5.248s	26.888s
512	2.769s	13.668s
1024	1.438s	6.941s
2048	0.7271s	3.5131s
4096	0.3792s	1.8183s

Table 5: Performance data for running HMB on different number of cores on Cirrus computer nodes using both test cases.

Table 6 has the performance numbers from ARCHIE-WeST and it should be noted that the ratio between the cost of the explicit and Implicit iterations is very different than all the other systems. Here the ratio is 1.5 where all the other nodes the ratio is closer to 3.5. In fact, the time spent calculating the just the residual in the explicit scheme is the same as the time taken to calculate both the residual and form the Jacobian matrix where the extra computational work should mean it takes 2.5 times longer. The scaling is much better within the node and this is probably due to the slower than expected single core performance.

ARCHIE-WeSt Cores (Nodes used)	CPU per explicit iteration	CPU per Implicit Iteration
1 (1)	35.314s	50.560s
2 (1)	19.476s	25.278s

4 (1)	9.0927s	14.878s
6 (1)	6.2429s	10.541s
8 (1)	4.7257s	7.7089s
12 (1)	2.8791s	4.8169s
24 (2)	1.485s	2.4544s
36 (3)	1.0296s	1.6467s
Half Cavity Calculation		
512		
1024		

Table 6: Performance data for running HMB on different number of cores on ARCHIE-WeST compute nodes using both test cases.

3. Summary and Conclusions

It seems that the GCC v4.8.2 compiler does not do a good job at producing an efficient executable when using on ARCHIE-WeST and that the some more testing of different available compilers should be undertaken to build a better executable.

HMB scaling within a node is effective up until about 12 cores after which there is very little reduction in wall clock time per iteration with the addition of more cores. The scaling between nodes is excellent with great and 90% efficiency seen on both ARCHER and Cirrus.

Although the single core performance of a KNL node was around 450% slower than a Cirrus node when they were both fully load the KNL node had reduced this gap to around 30%. From the speedup curves it also appears that HMB could be able to make use of extra cores with a KNL node while it is already at the limit of effective use of the Intel Xeon cores.