Performance and Scalability of Unstructured Mesh CFD Workflow on Emerging Architectures

Cameron W. Smith   Ben Matthews   Michel Rasquin
Kenneth E. Jansen

June 1, 2015

Abstract

The performance and scalability of an unstructured mesh based CFD workflow on Intel Xeon Phi based system is discussed.

1 Introduction

PHASTA is a parallel, hierarchic (2nd to 5th order accurate), adaptive, stabilized (finite-element [4]) transient analysis solver for compressible or incompressible flows. It solves PDE's typical of physical problems in fluid mechanics, electromagnetics, biomechanics, etc. PHASTA and it's predecessor ENSA were the first massively parallel unstructured grid LES/DNS code [7, 8, 10] and have been applied to flows ranging from validation benchmarks to complex cases.

The PUMI, parallel unstructured mesh infrastructure, adaptive meshing and scalable partitioning tools support the creation of meshes with tens of billions of elements for use by PHASTA. The largest PHASTA scaling run to date used over three million processors on the Argonne Mira IBM BlueGene /Q (BGQ) with a 92 billion element PUMI mesh [13]. PUMI supports these operations through the use of a component based design. At PUMI's core is an array based mesh representation component that provides efficient mechanisms to query and modify the mesh while maintaining a small memory footprint [6, 14]. Parallel mesh operations, such as the definition of the partition graph, the migration of elements, and synchronization of off-process boundary data, is provided by the APF component. These parallel mesh operations provide the supporting functionality to implement mesh adaptation and fast dynamic load balancing components, MeshAdapt [3, 12] and ParMA [16, 21], respectively.

ParMA APIs are used to (1) predicatively balance mesh elements during mesh adaptation to avoid memory exhaustion, and after adaptation operations are completed, (2) ensure that the applications mesh entity balance requirements are met. For a PHASTA analysis with linear shape functions that associated degrees of freedom with mesh vertices ParMA first targets the reduction of mesh vertex imbalance to ensure the scalability of the dominant equation solution analysis stage, and then balances elements, without disturbing the vertex imbalance, to scale the equation formation stage (forming the LHS $\mathbf{A}$ and the RHS $\mathbf{b}$). PHASTA's strong scalability on Mira was improved by over 35% using ParMA meshes relative to meshes prepared with only graph and geometric based partitioning methods [17].

The performance and scalability of the PUMI and PHASTA tools on Intel Xeon Phi based systems are examined in this paper and compared against BGQ system results. Tests were performed on the TACC Stampede and NERSC Babbage Intel Xeon Phi based systems, and the Argonne BGQ. The characteristics of the Intel Xeon Phi and the BGQ A2 processor are compared in Table 1.

processor	IBM PowerPC A2	Intel Xeon Phi SE10P	A2/Phi
cores	16	61	0.26
hwthreads/core	4	4	1.00
cpu clock freq (GHz)	1.6	1.09	1.47
L1 cache/core (KB)	16	32	0.50
l2 cache/core (KB)	2000	512	3.91
total l2 cache (KB)	32000	31232	1.02
in-order	yes	yes	-
theoretical peak MFLOPS	204.8	1073.6	0.19
STREAM benchmark GB/s /core [18, 20]	1.625	2.623	0.62

Table 1: Hardware comparison.

2 PHASTA Scaling

PHASTA was run on up to 3840 Stampede Phi and BGQ cores as shown in Table 2. One MPI rank was assigned to each of 60 cores on the Phi; the 61st core running the OS was avoided. Likewise, one MPI rank was run per core on BGQ. On Stampede Intel MPI core pinning (I_MPI_PIN_DOMAIN=core) was used to ensure processes did not move between cores and introduce jitter. Processes are pinned by default on the BGQ. To avoid excessive MPI memory usage on the Phi the rendezvous [1] and connectionless DAPL protocols [2] are used. Unlike the eager protocol which copies data with the assumption that the reciever is ready, rendezvous uses a more memory efficient handshaking model where data is not copied until the receiver declares itself as ready. The connectionless DAPL protocol, UD, reduces memory by using a fixed number of connection pairs as opposed to the default protocol which requires each pair of proceses to setup a one-to-one connection. Of the test cases ran on Babbage (Table 3) only the eight Phi, two Phi per node, 240 process case was slower (by 7%) with UD enabled.

PHASTA maintains strong scaling (relative to base computed on 480 cores) out to 3840 (three processor doublings) on Stampede; Column 3 of Table 2. Attempts to run with 240 cores for this case failed due to lack of memory but this is not a great concern given that typical PHASTA production runs are in this lightly loaded range (lower element and node counts) to exploit the strong scaling and compress the time-to-solution. Column 4 indicates the number of elements per part and indeed $O(5k)$ elements per part is typical for PHASTA production runs. While the dynamic range has been shown higher on BG (90% scaling through 9 processor doublings) this is primarily due to the larger "per core" memory. The proposed future path to exascale machines (Theta and Aurora) for Phi architecture double the fast memory (8GB to 16GB) together with lighter weight MPI implementations which will significantly extend the strong scaling in the lower core count direction while faster interconnects will extend it in the higher core count direction. That said, the 91% scaling at the typical production run level already shows great promise for CFD on this architecture. Column 4 in Table 2 shows the timing for runs on the same core count on BGQ. Comparing to Column 2 (with the ratio taken in column 5) it is clear that the Phi is approaching the same per-core performance as BGQ with no hardware-specific performance tuning. Work is underway using various tools provided by Intel, but early results suggest that almost all of the hotspots are currently vectorizing to a high percentage.

cores	avg. time (sec.)	phi-S	Elements/part	BGQ	BGQ-S	phi/BGQ
480	84.60	1.00	45,833	73.7	1.00	1.15
960	37.87	1.12	22,917	37.18	0.99	1.02
1920	26.12	0.81	11,458	18.8	0.98	1.39
3840	11.60	0.91	5,729	9.60	0.96	1.21

Table 2: PHASTA scaling results on Stampede.

Table 3 shows results from recent runs on Babbage. This table explores the influence of a variety of configurations changes on PHASTA performance. It sets up two normalized "base" configuration; Column 5 shows strong scaling on a per-core basis (as was done on Stampede) whereas Column 6 shows strong scaling on a per-Phi basis. Note that these base configurations are not the same as the 8 node, 60 process per node configuration studied in Table 2 for Stampede, rather, here the base is a 240 process run 8 nodes. Effectively, this base places twice as large of parts on the same number of nodes running half as many processors (30 instead of 60). This substantially reduces the communication pressure by decreasing the number of messages that must be passed (though the actual size of each message may rise, net traffic is reduced) and increasing the work load on each process (while holding half of them idle). Comparing this base to the 60 process per node runs shows substantial acceleration on a per-Phi basis (1.77 in column 7) which indicates that PHASTA is not network bound on this hardware and more processes are beneficial.

Table 3 also includes (in the lower half) results from running with 2 Phi's per node. Comparing the scaling numbers, it is clear that scaling does degrade somewhat faster when employing 2 Phi's per node. This is not of great concern since Theta and Aurora are proposed to be hostless Phi nodes and substantially higher network performance.

3 PHASTA Block Size Study

The equation formation work in PHASTA is dominated by the computation of integrals (intensive loads, stores, multiplies and adds) appearing in the weak form using quadrature which after implicit time integration [9] yields a system of non-linear algebraic equations. Equation formation work can be

cores	Phis	Phi/node	ppn	Per-Core Scaling	Per-Phi Scaling
240	8	1	30	1.00	1.00
480	8	1	60	0.88	1.77
960	32	1	30	0.95	0.95
1920	32	1	60	0.85	1.69
240	8	2	30	1.00	1.00
480	16	2	30	0.98	0.98
480	8	2	60	0.81	1.63
960	32	2	30	0.97	0.97
960	16	2	60	0.80	1.59
1920	64	2	30	0.90	0.90
1920	32	2	60	0.69	1.39
3840	64	2	60	0.55	1.11

Table 3: PHASTA scaling results on Babbage.

tuned to specific architectures by varying the number of elements in a block (block size) . Table 4 shows the results of a sweep on block size from a significantly longer run (to gather stable statistics). Column 3 divides a given block size compute time by the best performing block size to illustrate a rather broad sweet spot in the 64-96 elements per block range which tails up rather strongly (50% slow down) for block sizes smaller than 16 or greater than 256. There is clearly a tradeoff between larger block sizes more efficiently filling the vector loops and smaller block sizes reducing cache misses. Future work will determine if code modifications can alter this tradeoff to greater benefit.

4 ParMA Scaling

Running large numbers of MPI process counts on the Phi requires a non-trivial amount of memory. To test memory usage the same partitioning followed by ParMA improvement job was done using MPI only and then with MPI + pthreads, which requires MPI_THREAD_MULTIPLE. Partitioning from two parts to 128 parts, one part per thread or process, the memory usage increase for an MPI only run is roughly 30x (51Mb to 1500Mb) while the MPI+pthread run only has a 2x memory only increased roughly 2x (140Mb

Elements per block	time (avg)	tbest/t
16	299.93	0.67
32	236.8	0.85
48	213.45	0.95
64	206.49	0.98
72	202.77	1.00
80	202.56	1.00
96	202.35	1.00
128	228.9	0.88
256	307.39	0.66

Table 4: The effect of block size on Phi PHASTA performance.

elements (M)	parts	sel (s)	migr (s)	reduction	reduction rate
15.5	2048	55.61	18.15	6	5.25E-03
25.3	4096	16.04	4.7	8	7.64E-03
202.1	8192	50.91	6.71	5	1.07E-04
123.9	16384	250.91	108.27	11	3.09E-05

Table 5: Stampede native Phi run times ParMA Vertex > Element partition improvement following local ParMETIS partitioning. 15 MPI ranks were ran on each Phi and each rank spawns four posix threads.

vs 65Mb). Given the slow memory increase for MPI+pthreads the initial set of scaling tests were run on up to 16,384 Phi cores as shown in Table 5. Note, the eager communication protocol was used to avoid an Intel MPI 14 failure seemingly associated with dynamic switching between the eager and rendezvous protocol based on message length. Investigations into this issue are ongoing with TACC HPC specialists.

The 'sel' times the execution of ParMA element selection [16] while 'migr' times the subsequent migrations. The 'reduction' column lists the percentage point reduction in the vertex imbalance after ParMA execution. Imbalance reduction is normalized in the 'reduction rate' column by the number of elements in the mesh, the number of parts, and the time spent in selection and migration. From 2048 and 4096 the reduction rate improves, but from 4096 to 8192, and again from 8192 to 16384 a decline in the rate is observed.

Further tests to explore the reduction rate declines will use the latest Intel 15 compilers and Intel MPI 5 (at the time the tests were run only Intel 14 and

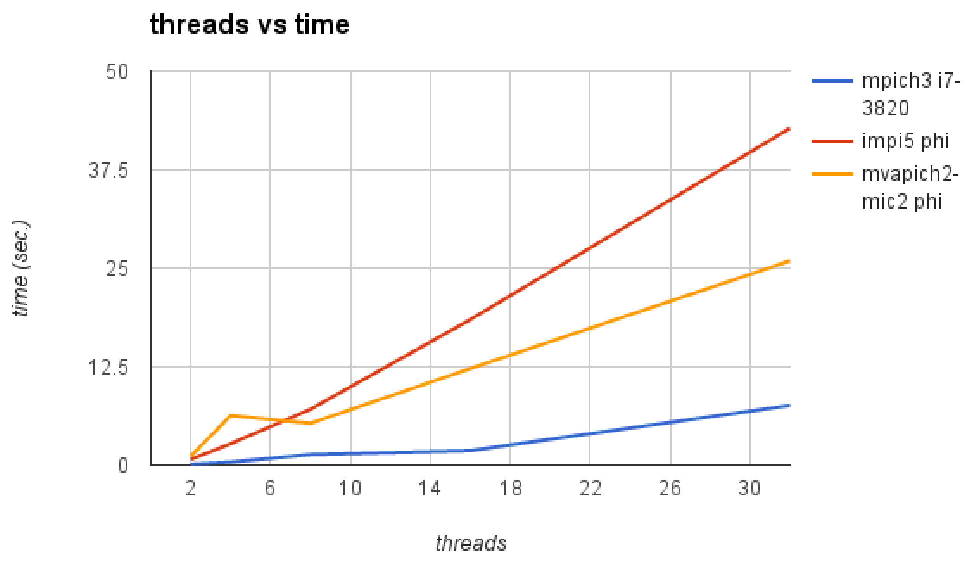
Intel MPI 4 were available) using the Phi specific Intel recommended compiler flags that were successfully applied to PHASTA. In PHASTA performance gains between 10% and 36% were observed using the '-O3 -opt-assume-safe-padding -opt-streaming-stores always -opt-streaming-cache-evict=0' flags vs just '-O3'.

In addition to this effort is an ongoing pursuit of performance with increasing thread counts per MPI rank. Testing has shown that in the presence of MPI_THREAD_MULTIPLE increasing the number of threads per MPI process has a significant negative affect on performance [15]. Table 6 lists the left-right communication kernel test results using one MPI process and increasing numbers of threads. MPI implementation tests on Phi pinned one, two and four threads per core. The growth in Phi run times exceeds the growth rate observed with MPICH3 on a four core Intel i7-3820 with hyperthreading enabled. With 32 threads, 8x oversubscribed on the i7, the i7 processor runs 3x and 6x quicker than the Phi with MVAPICH2-MIC v2.0 and Intel MPI 5, respectively. Figure 1 depicts the much slower growth in time as threads are increased on the i7 vs the Phi. This issue is also being investigated with help from TACC HPC specialists. The degraded MPI_THREAD_MULTIPLE performance on the Phi may be avoided using the Intel implementation of the MPI 3.1 endpoints APIs, EPLib [19], the conceptually similar FineGrain MPI [11], or modification of the PCU [5] communication library used by ParMA.

		threads	threads				
system	mpi	per core	2	4	8	16	32
i7-3820	mpich3	N/A	0.12	0.398	1.344	1.856	7.578
phi	impi5	1	0.7422	2.7226	7.1064	18.4822	42.765
phi	impi5	2	0.9698	2.3846	6.4668	15.289	41.4386
phi	impi5	4	0.9668	2.0686	6.0412	13.7852	38.2496
phi	mvapich2-mic2	1	1.1414	6.3	5.3218	12.2706	25.9254
phi	mvapich2-mic2	2	0.6936	2.2848	4.9388	10.5948	22.7568
phi	mvapich2-mic2	4	0.6854	2.7284	5.356	10.8154	23.2128

Table 6: Left-right communication kernel [15] testing on a Stampede Phi vs an Intel i7-3820 with four cores and hyper threads on with MPI_THREAD_MULTIPLE.

Figure 1: Threads vs time on a Stampede Phi vs an Intel i7-3820 with four cores and hyper threads on.



References

[1] Communication Fabrics Control, Intel MPI Library Reference Manual for Linux OS, 2015.

[2] Enabling Connectionless DAPL UD in the Intel MPI Library, 2015.

[3] Frédérik Alauzet, Xiangrong Li, E. Seegyoung Seol, and Mark S. Shephard. Parallel anisotropic 3d mesh adaptation by mesh modification. *Engineering with Computers*, 21(3):247–258, jan 2006.

[4] A. N. Brooks and T. J. R. Hughes. Streamline upwind / Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 32:199–259, 1982.

[5] Daniel A Ibanez, Ian Dunn, and Mark S Shephard. Hybrid mpi-thread parallelization of adaptive mesh operations. *Parallel Computing*, 2014.

[6] Daniel A. Ibanez, E. Seegyoung Seol, Cameron W. Smith, and Mark S. Shephard. Pumi: Parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software*, under review, 2015.

[7] K. E. Jansen. Unstructured grid large eddy simulation of flow over an airfoil. In *Annual Research Briefs*, pages 161–173, NASA Ames / Stanford University, 1994. Center for Turbulence Research.

[8] K. E. Jansen. A stabilized finite element method for computing turbulence. *Comp. Meth. Appl. Mech. Engng.*, 174:299–317, 1999.

[9] K. E. Jansen, C. H. Whiting, and G. M. Hulbert. A generalized-α method for integrating the filtered Navier-Stokes equations with a stabilized finite element method. *Comp. Meth. Appl. Mech. Engng.*, 190:305–319, 1999.

[10] K.E. Jansen. Unstructured grid large eddy simulation of wall bounded flow. In *Annual Research Briefs*, pages 151–156, NASA Ames / Stanford University, 1993. Center for Turbulence Research.

[11] Humaira Kamal and Alan Wagner. An integrated fine-grain runtime system for MPI. *Computing*, 96(4):293–309, 2014.

[12] Aleksandr Ovcharenko, Kedar C. Chitale, Onkar Sahni, Kenneth E. Jansen, and Mark S. Shephard. Parallel adaptive boundary layer meshing for cfd analysis. In Xiangmin Jiao and Jean-Christophe Weill, editors, *Proceedings of the 21st International Meshing Roundtable*, pages 437–455. Springer Berlin Heidelberg, 2013.

[13] M. Rasquin, C. Smith, K Chitale, S. Seol, B.A. Matthews, J.L. Martin, O. Sahni, R.M. Loy, M.S. Shephard, and K.E. Jansen. Scalable fully implicit finite element flow solver with application to high-fidelity flow control simulations on a realistic wing design. *Computing in Science and Engineering*, 16(6):13–21, 2014.

[14] E.S. Seol. *FMDB: Flexible Distributed Mesh Database for Parallel Automated Adaptive Analysis.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, May 20055. SCOREC Report 2005-9.

[15] Cameron W. Smith. Communication kernel for performance testing on stampede., 2015.

[16] Cameron W. Smith, Michel Rasquin, Dan Ibanez, Kenneth E. Jansen, and Mark S. Shephard. Application specific partition improvement. *SIAM Journal on Scientific Computing*, submitted, 2015.

[17] Cameron W. Smith, Michel Rasquin, Dan Ibanez, Mark S. Shephard, and Kenneth E. Jansen. Partition improvement to accelerate extreme scale cfd. *SIAM Journal on Scientific Computing*, in preparation, 2015.

[18] Konstantin S. Solnushkin. Memory Bandwidth for Intel Xeon Phi (And Friends), Feb. 2013.

[19] Srinivas Sridharan, James Dinan, and Dhiraj D Kalamkar. Enabling efficient multithreaded mpi communication through a library-based implementation of mpi endpoints. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 487–498. IEEE, 2014.

[20] Bob Walkup. Application Performance Characterization and Analysis on Blue Gene/Q, 2008.

[21] M. Zhou, O. Sahni, T. Xie, M.S. Shephard, and K.E. Jansen. Unstructured mesh partition improvement for implicit finite element at extreme scale. *Journal of Supercomputing*, 59(3):1218 – 28, 2012/03/.