

## Early Multi-Node Performance Evaluation of a Knights Corner (KNC) Based NASA Supercomputer

Subhash Saini, Haoqiang Jin, Dennis Jespersen, Samson Cheung\*, Jahed Djomehri\*, Johnny Chang\*, and Robert Hood\*

NASA Advanced Supercomputing (NAS) Division

NASA Ames Research Center

Moffett Field, California, USA

\*Computer Sciences Corporation (CSC)

{subhash.saini, haoqiang.jin, dennis.jespersen, samson.h.cheung, jahed.djomehri, johnny.chang, robert.hood}@nasa.gov

**Abstract**—We have conducted performance evaluation of a dual-rail *Fourteen Data Rate (FDR) InfiniBand (IB) connected cluster*, where each node has two Intel Xeon E5-2670 (Sandy Bridge) processors and two Intel Xeon Phi coprocessors. The Xeon Phi, based on the Many Integrated Core (MIC) architecture, is of the Knights Corner (KNC) generation. We used several types of benchmarks for the study. We ran the MPI and multi-zone versions of the NAS Parallel Benchmarks (NPB)—both original and optimized for the Xeon Phi. Among the full-scale benchmarks, we ran two versions of WRF, including one optimized for the MIC, and used a 12 Km Continental U.S (CONUS) data set. We also used original and optimized versions of OVERFLOW and ran with four different datasets to understand scaling in symmetric mode and related load-balancing issues. We present performance for the four different modes of using the host + MIC combination: native host, native MIC, offload, and symmetric. We also discuss the various optimization techniques used in optimizing two of the NPBs for offload mode as well as WRF and OVERFLOW. WRF 3.4 optimized for MIC runs 47% faster than the original NCAR WRF 3.4. The optimized version of OVERFLOW runs 18% faster on the host and the load-balancing strategy used improves the performance on MIC by 5% to 36% depending on the data size. In addition, we discuss the issues related to offload mode and load balancing in symmetric mode.

**Keywords:** Benchmarking; Performance; WRF, OVERFLOW

### I. INTRODUCTION

Recently, the US Congress passed a law directing the Department of Energy (DOE) to develop an exascale class supercomputing capability within the next decade in order to meet the objectives in this endeavor of the nuclear stockpile stewardship program [1]. One of the key challenges is to develop energy efficient circuits and improved power and cooling technologies. The stagnancy of processor frequency due to the constraints of power and current leakage has led computer hardware vendors to increase parallelism in their processor designs. NVIDIA and Intel have been proactive in developing low-powered processors for use in hybrid heterogeneous systems, and such systems currently occupy many of the top spots in the Top500 list [2]. Since 2011, the most powerful supercomputer systems ranked in the Top500 list are hybrid systems of two different architectures composed of thousands of nodes that include both processors and accelerators, i.e. accelerators such as the NVIDIA General-Purpose Graphical Processing Unit (GPGPU) and the Intel Xeon Phi coprocessor based on the Intel Many

Integrated Core (MIC) architecture [3, 4]. In the 2014 June Top500 list, a total of 62 systems on the list use accelerator/coprocessor technology — 44 of these use NVIDIA GPGPU chips and 17 systems use Intel Xeon Phi [2].

The Intel Xeon Phi based Tianhe-2 (MilkyWay-2) system at the National Supercomputer Center in Guangzhou, China and Stampede system at Texas Advanced Computing Center (TACC) are currently ranked 1 and 7, respectively, on June 2014 Top500 list [2, 5, 6]. They both use the current generation of Xeon Phi called Knights Corner (KNC). In April 2014, the National Energy Research Scientific Computing Center (NERSC) and Cray Inc. signed a \$70+ million contract for a next-generation supercomputer based on a future Intel Xeon Phi processor code-named “Knights Landing” (KNL), which can provide 3 teraflops of peak performance per processor. It is scheduled for delivery in mid-2016 [7,8]. Also the U.S. DOE National Nuclear Security Administration (NNSA) has awarded Cray \$174 million to develop Trinity, a multi-petaflop supercomputer based on KNL [9]. The KNL processor is not a coprocessor like KNC, but is “self-hosted,” meaning that it is neither an accelerator nor dependent on a host processor.

Many researchers investigating KNC performance have examined single nodes, where one or two Xeon processors on a single host are combined with one or two KNC coprocessors [7-15]. There are not, however, many publications on programming multiple nodes with MIC coprocessors. Park et al. achieved 1 Tflop/s on 64 nodes of Xeon Phi (KNC) and 6.7 Tflop/s with 512 nodes for a 1-D FFT kernel, which is 1.5 times higher than on 512 nodes of Xeon processors [10]. Joo et al. demonstrated a fully ‘native’ multi-node LQCD implementation running entirely on KNC nodes with minimum involvement of the host processor with strong scaling to 3.6 Tflop/s on 64 KNC [11]. Saini et al. did performance evaluation of a single MIC using several low level benchmarks and two applications [13]. They measured STREAM bandwidth, load latency, load read and write bandwidth for L1/L2/L3 cache and main memory, MPI latency and bandwidth, and offload bandwidth between host and MIC0/MIC1. They also reported the intra-node performance of several MPI functions along with the overhead of various OpenMP directives and constructs with data privatization, loop scheduling, and synchronization. They also measured the read and write bandwidth for MIC. In addition, they reported the performance of NAS Parallel

Benchmarks (NPB) MPI and NPB OpenMP, and two CFD applications for native host mode, native MIC mode and symmetric mode for a single node. It may be noted that Saini et al. conducted the performance evaluation of only a single node i.e. one node (one host + MIC0 + MIC1). In the present paper we evaluate the performance of multiple nodes [13], optimize two large-scale production quality codes and implement a new technique to load balance for symmetric mode, which enhanced their performance significantly.

In the present paper, we study the multi-node performance of an InfiniBand (IB) connected cluster called “Maia”, where each node has two Intel Xeon E5-2670 (Sandy Bridge) processors and two Xeon Phi 5110P coprocessors (KNC). We will refer to the two Sandy Bridge processors collectively as the “host” and the KNC coprocessors as the “MIC”, using “MIC0” and “MIC1” whenever we need to distinguish between the two coprocessors.

To the best of our knowledge the following are our original contributions:

- We optimized a production quality computational fluid dynamics (CFD) code OVERFLOW, which enhanced the performance on the host by 18%. In addition, we implemented a new load-balancing strategy among hosts and MICs for multiple nodes that increased in performance 5% to 36%, depending on the size of data used. We compared the performance of OVERFLOW in host-native, MIC-native and symmetric modes for four different data sets with grid sizes of 10.6, 36, 83, and 91 million on up to 48 host nodes and 96 MIC coprocessors [17].
- We used Intel optimized climate code WRF 3.4 on MIC, which resulted in a 47% performance increase for the symmetric mode using a hybrid-programming paradigm (MPI + OpenMP). Various strategies were used to load balance MPI threads and OpenMP threads for optimal performance. We evaluated the performance of the optimized and original weather code WRF 3.4 with up to 4 host nodes and 8 MIC coprocessors for a 12 Km CONUS data set [18].
- We evaluated the performance of NAS Parallel Benchmarks (NPB) MPI version, and the NPB multi-zone version (NPB-MZ) up to 64 host nodes and 128 MIC coprocessors [16].
- We implemented three offload versions of NPB SP and BT compact applications from the NPB 3.3 suite and evaluated to examine data transfer at different granularities.

The remainder of the paper is organized as follows. Section II provides details of the MIC-based heterogeneous computing system called Maia. In Section III we give a brief overview of the experimental setup used in the study. Section IV describes the four programming modes available on Maia. Section V gives the description of benchmarks and applications. In Section VI we discuss the multi-node

performance results obtained in our evaluation of the Sandy Bridge hosts and the MIC coprocessors. In Section VII we present our conclusions.

## II. MAIA COMPUTING PLATFORM

The NASA Advanced Supercomputing (NAS) Division recently installed a 128-node heterogeneous SGI Rackable computing system. Each node has 2 Sandy Bridge (Intel Xeon E5-2670) processors and 2 KNC coprocessors (Intel Xeon Phi). Each Sandy Bridge processor has 8 cores with a clock of 2.6 GHz, a 20MB L3 cache, and two hyper-threads (HT) per core. Each Sandy Bridge node has two Intel Xeon Phi 5110p (Knights Corner - KNC) coprocessors. Each MIC is a 60-core Symmetric Multi Processor (SMP) on a single die using 22-nm process technology and running at 1.053 GHz. It has four hardware threads per core, 8 GB of GDDR5 memory, and a peak performance of 1010.5 Gflop/s. Overall the system has a theoretical peak performance of 301.3 Tflop/s. Of that peak performance, 42.6 Tflop/s come from the 2,048 Sandy Bridge cores and 258.7 Tflop/s come from the 15,360 MIC cores. The system has 4 TB of memory available to the Sandy Bridge processors and 2 TB for the MIC coprocessors for a total memory of 6 TB. The four hardware threads on each MIC core can hide memory and multi-cycle instruction latency. Because instructions from a thread can be issued only every other cycle, it is absolutely necessary to use a minimum of two threads per core. Each core has 512-bit wide vector units that can execute 8 double-precision or 16 single-precision, single-instruction multiple-data (SIMD) instructions in a single clock. Each core has two levels of cache: 32 KB L1 data cache and a globally cache coherent 30 MB L2 cache partitioned among the 60 cores, i.e. each core has a 512 KB partition. The memory bandwidth in streaming can reach 165 GB/s [13].

Each of the 128 nodes has two different memory systems. The host memory is 32 GB shared cache-coherently by the 16 cores of the two Sandy Bridge processors. The cores of each MIC share an 8 GB GDDR5 cache-coherent memory system. Each MIC is connected to the host via a separate 16-lane PCI Express (PCIe) bus. An FDR IB Host Channel Adapter (HCA) plugged into the first PCIe bus connects to other nodes.

## III. EXPERIMENTAL SETUP

All benchmarks and applications were run on the NAS Maia cluster with a Network File System (NFS) home and Lustre scratch file systems re-exported to MIC via a user space implementation of NFS (UNFS). The system software consisted of SUSE Linux Enterprise Server SLES11SP2, the Intel Manycore Platform Software Stack (MPSS) version 3.1.2-1, Intel compiler version 15.0, and Intel MPI version 4.1. Cross-compilation was done via the Intel compiler (-mmic) and an in-house “as-if-native” compilation environment. The “as-if-native” compilation environment was created for Maia to enable the build of any open-source package as if it was compiled natively on the MIC without modification. This allowed the MIC “busy box” environment to be supplanted by a set of software that is fully featured with executable such as the bash shell, tesh shell, and modules, at

the user’s choice. The result makes the Xeon Phi environment close to a normal Xeon system. We used this “as-if-native” compilation to build required libraries, such as Network Common Data Form (NetCDF). Job submission and monitoring were done using PBS 11.2, with a MIC reset at the end of each job. Common environment variables used were:

```
I_MPI_MIC=1
I_MPI_TIMER_KIND=rdtsc
MIC_KMP_AFFINITY=balanced
```

In addition, we set the following two environment variables to specify which DAPL providers are used for various message sizes.

```
I_MPI_DAPL_DIRECT_COPY_THRESHOLD=8192,262144
I_MPI_DAPL_PROVIDER_LIST=ofa-v2-m1x4_0-1,ofa-v2-scif0
```

These last two variables result in three possibilities (small < 8KB, medium > 8KB and < 256 KB, and large messages > 256 KB) for data transfers between the host and the MICs.

#### IV. PROGRAMMING MODES

In this paper, we evaluated the performance of four programming modes.

**Offload Mode:** In offload mode, an application is launched on the host, and then parallel compute-intensive subroutines/functions are offloaded to the MIC using special directives that take care of code execution and data transfer seamlessly. An OpenMP parallel region is used to distribute work over MIC threads.

**Native Host Mode:** In native mode, the entire application is run exclusively on the host Sandy Bridge processors; MIC coprocessors are not used.

**Native MIC Mode:** In this mode, the entire application runs only on the MIC coprocessors. MIC-to-MIC communication is via host. Existing code running on the host can be compiled with `-mmic` option without any changes.

**Symmetric Mode:** In symmetric mode, an application is run using both the host processors and the MIC coprocessors; it needs to be compiled for host and MIC separately. A major challenge is to optimally balance the work between the hosts and coprocessors.

#### V. BENCHMARKS AND APPLICATIONS

##### A. NPB benchmarks:

The NPB suite developed at NASA has eight benchmarks: five kernels (CG, FT, EP, MG, and IS) and three compact applications (BT, LU, and SP). We used several versions of the Class C benchmarks in our study.

**NPB MPI:** We used MPI version 3.3 Class C of the NPB to measure multi-node scaling.

**NPB-MZ:** Multi-zone versions of NPB (NPB-MZ) are designed to exploit multiple levels of parallelism in applications and to test the effectiveness of multi-level and hybrid parallelization paradigms and tools. In this study we used BT-MZ and SP-MZ to measure single node and multi-node scaling.

**Offload codes:** We created offload versions of the NPBs SP and BT to examine data transfer at different granularities: at the loop or multi-loop level, at the subroutine level, and the whole computation.

##### B. Science and Engineering Applications

We used two full, production-quality applications representative of NASA’s workload. A brief description of these applications follows

###### 1) OVERFLOW

OVERFLOW is a general-purpose Navier-Stokes solver for computational fluid dynamics (CFD) problems [14]. The MPI version, a Fortran90 application, has 130,000 lines of code. The code uses an overset grid methodology to perform high-fidelity viscous simulations around realistic aerospace configurations. The main computational logic of the sequential code consists of a time loop and a nested grid loop. The code uses finite differences in space with implicit time stepping. Parallelism in OVERFLOW is at two levels — at the high level there is explicit-message passing using MPI and at the low level it uses OpenMP. We run Overflow for 100 time steps and each time step has two stages.

**Communication:** Each MPI rank sends all the inter-grid data needed by other MPI ranks, and receives all necessary data.

**Computation:** Each MPI rank computes on its grids, with no message passing involved.

Finally, all MPI ranks send a small amount of data (residuals, minimum pressure and density, etc.) to the MPI rank 0. All input/output (reading the grid and writing the results and restart file) is via the MPI rank 0. In this paper, we used hybrid mode (MPI + OpenMP) with MPI for parallelism at the outer level and OpenMP for parallelism at the inner level. We used the following four data sets.

**DLRF6-Large:** The dataset used is a wing-body-nacelle-pylon geometry (DLRF6-Large), with 23 zones and 36 million grid points. The input data set is 1.6 GB in size, and the solution file is 2 GB.

**DLRF6-Medium:** We also used a smaller data set (DLRF6-Medium) with 10.8 million grid points, as the DLRF6-Large case is too large to run on a single MIC coprocessor.

**DPW3:** DPW3 is a finer-grid version of the DLRF6-Large case. It’s wing-body geometry with 83 million grid points before grid splitting.

**Rotor:** Rotor is the NAS rotor test case. There are 91 million grid points before grid splitting.

###### 2) WRF

The Weather Research and Forecasting (WRF) model is a next-generation, mesoscale numerical weather prediction system designed to serve both atmospheric research and operational forecasting needs. The model serves a wide range of meteorological applications across scales ranging from meters to thousands of kilometers. WRF allows researchers the ability to produce simulations reflecting either real data (observations and analyses) or idealized atmospheric

conditions. We used two versions of the code. The first version is an original version 3.4 from National Center for Atmospheric Research (NCAR); the second is the Intel optimized WRF 3.4 for MIC. We used a benchmark case of 12-km Continental U.S (CONUS), simulating 48 hours in October 2001 with a time step of 72 seconds.

The WRF 3.4 code has parallelism at two levels. The outer loops are parallelized with MPI and the inner loops with OpenMP. This makes it simple to run WRF in symmetric mode because part of WRF runs on the host and part on the MIC.

## VI. RESULTS

In this section we present results of our tests of MPI and offload bandwidth as well as performance of the various NPBs and the two full applications (OVERFLOW and WRF).

### A. NAS Parallel Benchmarks

#### 1) MPI VERSION

Figure 1 shows the scaling performance of compact applications BT, SP, and LU from the NPB Class C suite on native host and native MIC. In this figure, bars show the results for native MIC mode whereas lines in the graph correspond to native host mode using Sandy Bridge (SB) processors. Each host node contains two SB processors and two MICs (MIC0 and MIC1). Each SB processor has 8 cores and each MIC has 60 cores. When we indicate 128 SB processors it means we used 64 host nodes.

Each MIC can have a maximum of 4 threads per core. For optimal performance one must use a minimum of 2 threads on a core as it issues instructions every other cycle. In addition, the number of threads per core is a tuning parameter that one needs to determine by experimentation for a given application.

Fig. 1 shows the best total time (computation and communication time) for a given number of MICs or SB processors ranging from 1 to 128. For SB processors we used one MPI process per core, e.g., 32 SBs mean 32x8 MPI processes. However, for the MICs, the best performance often involved leaving cores idle. The number of MPI processes used for optimum performance on a MIC varies with the number of MICs used. The total number of MPI processes used for each MIC count is indicated within the bars. In addition, for BT and SP, there is a restriction of running only a square grid of MPI processes and for LU there is a restriction of running on power-of-two MPI processes. For a given number of MICs we ran the benchmarks by varying the number of MPI processes per MIC and used the run with the minimum time. This was repeated for all MIC counts. For example, the best result for BT using 32 MICs was from 484 MPI processes and only about 15 cores were used per MIC. For a small number of processors (< 4) one MIC is about one SB processor. While scaling is reasonably good on SB processors, it is much worse on MICs, partly due to communication resulting from many more MPI processes used, inefficient resource utilization on MICs, load balancing, and poor performance of MPI functions when using 2 or 3 or 4 MPI processes per core.

It should be noted that performance of MPI functions in native MIC mode is 3 to 20 times worse than in native host mode as reported by Saini et al. [13]. Poor scalability for BT and SP on MIC is because of load imbalance using the pure MPI paradigm. It clearly shows that pure MPI is not appropriate for MIC, as one can't load balance the workload. We will see later on that a hybrid-programming model (MPI + OpenMP) resolves the scaling issue for BT and SP (see Figure 2). For the same reason, all the published literature on applications running on MIC use a hybrid-programming model (MPI + OpenMP).

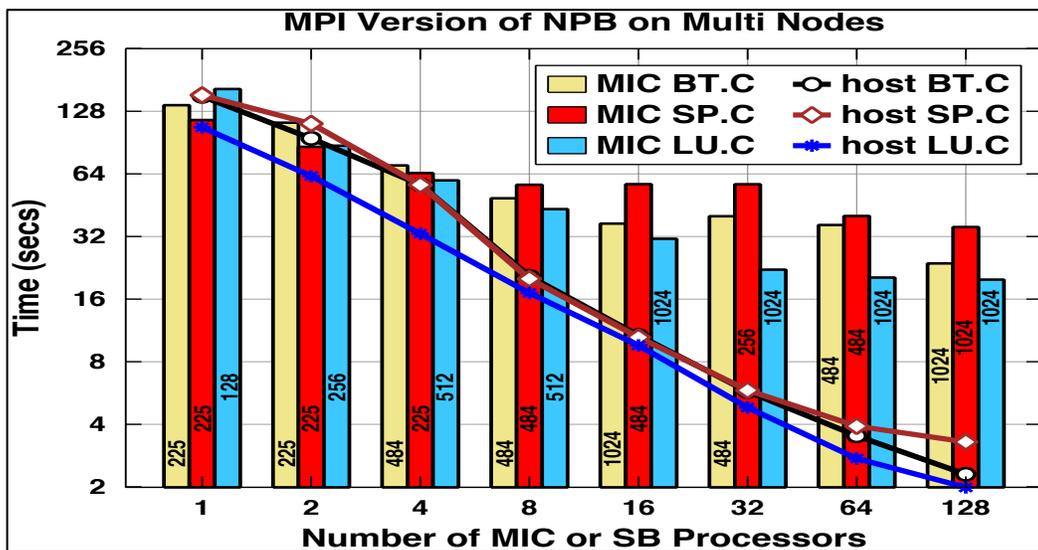


Figure 1. Performance of MPI version of Class C BT, SP, and LU benchmarks for host and MIC.

Figure 2 shows the scaling performance of compact applications CG, MG, and IS from the NPB Class C suite on native host and native MIC. Scaling of CG and IS is worse on the host for large number of processors. The reason for this poor scaling of CG is that this benchmark uses indirect addressing and as such cannot reuse the cache efficiently. In addition, it sends lots of messages with average message length of 4 KB and thus is network latency bound. Scaling of CG on MIC is worse than that on the host. We did extensive testing to investigate this and found that the compiler vectorized the most time-consuming loop using the gather-scatter vector instructions, but performance of this version was only 10% better than the version without vectorization. This indicates that the gather-scatter instruction is not efficient on MIC as it is done in software and not in hardware. The other reasons are that MIC has relatively lower memory bandwidth per process and lower network bandwidth between MIC of one node to MIC of another node. We measured this bandwidth to be only 950 MB/s compared to 6 GB/s of MIC0 to MIC1 of the same node.

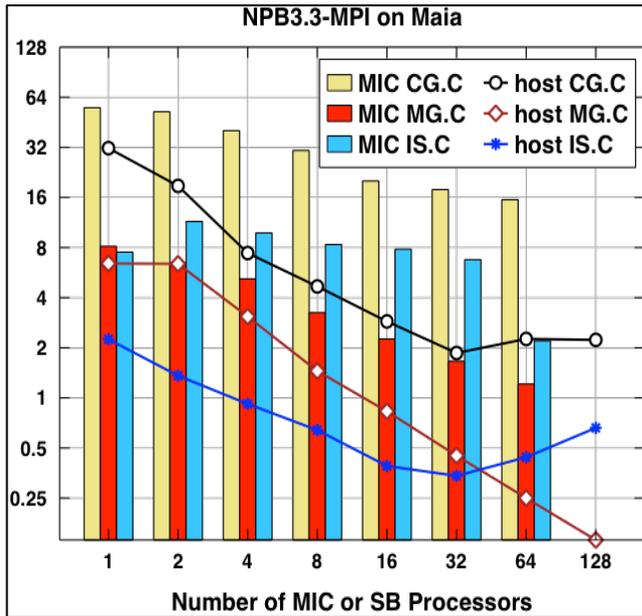


Figure 2. Performance of Class C CG, MG, and IS benchmarks on Maia.

### 2) NPB-MZ:

Figure 3 shows the performance of BT-MZ and SP-MZ for Class C. The notation  $r \times t$  is used within the bars to indicate the number of MPI ranks ( $r$ ) and the number of OpenMP threads ( $t$ ) per MPI rank on each MIC; e.g.,  $4 \times 30$  denotes 4 MPI ranks and 30 OpenMP threads per MPI rank. We present the best result for a given number of MIC or SB processors. We found that one MIC is about one SB processor for SP-MZ, but close to two SB processors for BT-MZ. Hybrid MPI-OpenMP codes scale better than pure MPI on MICs as shown in Figure 1 because it is easier to adjust the number of MPI ranks and OpenMP threads to get better resource utilization.

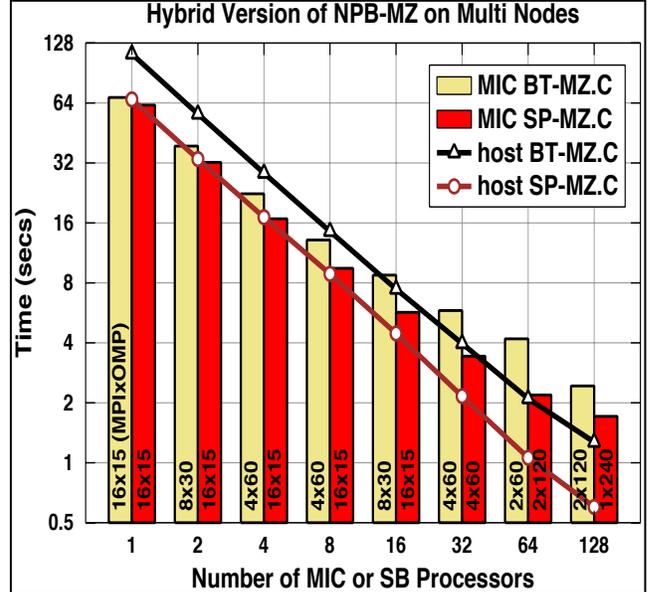


Figure 3. Performance of Class C BT-MZ and SP-MZ benchmarks on MICs and SBs

### 3) Offload Mode:

In this section we present the results for the OpenMP version of the BT and SP benchmarks and compare performance for offload mode with native host and native MIC modes. We created three versions of BT and SP for offload testing on Maia: offload multiple OpenMP loops, offload the iteration loop in the main program, or offload the whole computation. Figures 4 and 5 show the performance of BT and SP, respectively, in native host, native MIC, and offload modes on a single MIC. We see that the performance of all the offload versions is much lower than both native host and native MIC modes except for offloading the whole computation. The main reason for this is the high overhead for data transfer between the host and the MIC. The impact of the Coprocessor Offload Infrastructure (COI) daemon is visible in the offload mode when compared to the native MIC mode for 240 threads, which corresponds to 4 threads per core. Also, with Intel’s MPSS software, many of the kernel services and daemons are affinitive to the “Boot Strap Processor” (BSP), which is the last physical core. This core is also where the offload daemon runs the services required to support data transfer for offload. It is therefore generally beneficial to avoid using this core for user code, i.e., one should use only 59 cores. For this reason we saw performance drop at 60, 119, 179 and 237 threads. For our study we used only 59 cores of the MIC. Therefore, we show results only for 118, 178 and 236 threads corresponding to 1, 2, 3 and 4 hardware threads per core.

The amount of data transferred between host and MIC, and the number of offload invocations are different for our three offload versions BT and SP. By simply offloading the multiple “do loops” in both BT and SP. Here the amount of data transferred is the least for each offload invocation. But the aggregate amount of data transferred and the number of

offload invocations are the most among the three versions. Thus the performance of this version is the worst. The performance is improved when offloading the iteration loop with fewer offload occurrences and data. The efficient method is to offload the whole computation to the MIC. In this version the data transferred is the least because input data is generated on the host and transferred to the MIC only once. The guideline to evaluate whether an application is appropriate for offload mode is the incurred cost of data transfers and offloads overhead. It is obvious from our experiment that NPB compact applications BT and SP are not suitable for offload mode.

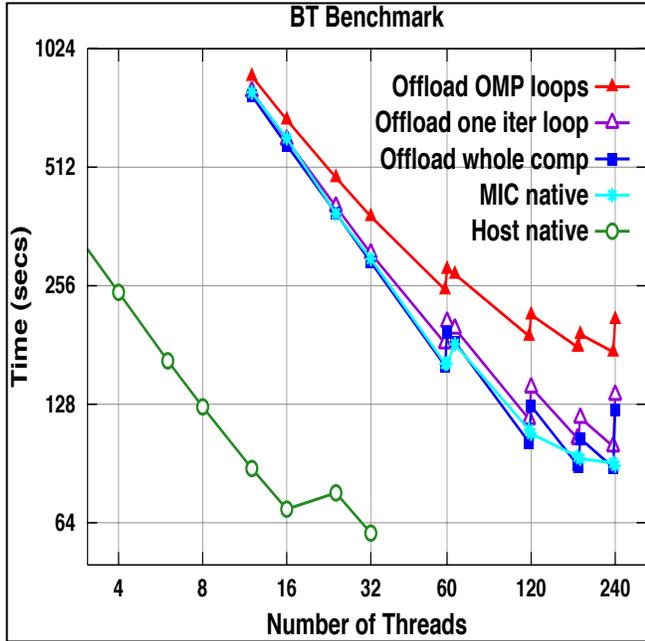


Figure 4. Performance of three offload versions of BT compared to host-native and MIC-native versions.

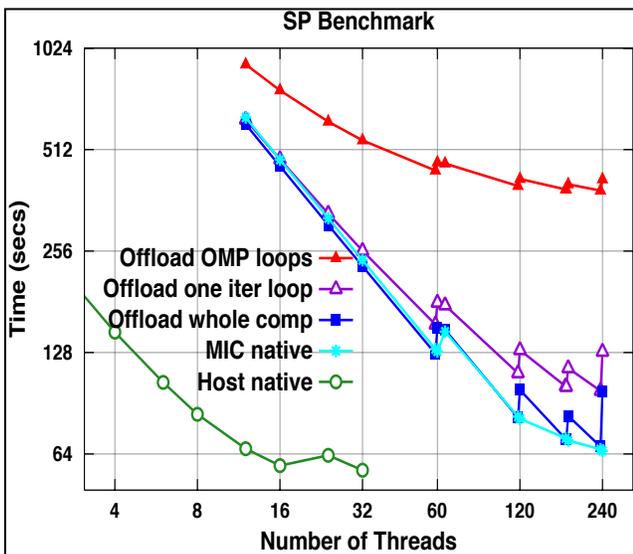


Figure 5. Performance three offload version of SP compared to host-native and MIC-native versions.

## B. Scientific and Engineering Applications

In this subsection we present the results of two large-scale applications – one from computational fluid dynamics (OVERFLOW) and the other from weather modeling (WRF).

### 1) OVERFLOW

Figure 6 shows the wallclock time per step of the MPI+OpenMP hybrid OVERFLOW in native host and symmetric modes for the DLRF6-Large data set. The notation  $m \times n + p \times q$  used here means  $m \times n$  hybrid on the hosts where  $m$  is number of MPI processes and  $n$  is number of OpenMP threads per MPI process;  $p \times q$  hybrid on the MIC where  $p$  is number of MPI processes on a MIC and  $q$  is number of OpenMP threads per MPI process on a MIC. We used up to 48 hosts (96 Sandy Bridge processors and 96 MICs). Figure 5 shows results for 1 host and 2 hosts as well as for 1 host plus 2 MICs. In this figure we show four times — total time, flow right-hand-side time, flow left-hand-side time, and boundary exchange time (CBCXCH). CBCXCH is basically a communication time. In host-native mode, CBCXCH is less than 3% of the total time whereas in symmetric mode (1 host + MIC0 + MIC1) it is about 20%. The reason for this is that latency from host to MIC0, host to MIC1, and MIC0 to MIC1 is very high.

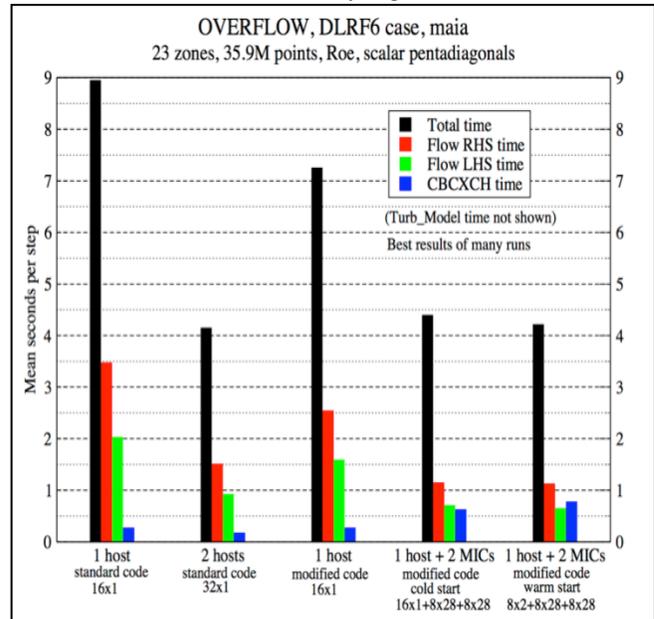


Figure 6. Performance of OVERFLOW on a host + MIC0 + MIC1.

Figure 6 shows results both for standard OVERFLOW as well as a modified version. We did several transformations to enhance the performance of OVERFLOW. The performance modification to OVERFLOW consisted of an attempt to expose more parallelism for OpenMP and to allow thread-level vectorization. The original OpenMP parallelism in OVERFLOW consists of directives to parallelize loops over planes of a 3-dimensional (3-D) grid. The resulting number of OpenMP threads is typically on the order of 40. Inside the OpenMP loop, computation is done on a 2-dimensional (2-D)

plane of data. The optimization consisted of recoding to have OpenMP parallelism over strips of a plane instead of over a full plane. This typically increases the number of OpenMP threads to something on the order of a few hundred. In addition, the recoding has each thread compute over a smaller data set, a strip of a 2-D plane instead of a full plane, which may have the effect of decreasing cache traffic. These optimizations resulted in 18% improvement in total time on one host. We show the breakdown time of host + MIC runs compared to host only. The best performance on a single host is for  $16 \times 1$  and the scaling from one to two hosts is excellent: 9 seconds for one host to 4.1 seconds for two hosts. The reason for this is that on two hosts more data fits into cache. It is clear from this figure that the performance on 2 hosts is almost the same as on 1 host plus 2 MICs. The reason for the lower performance of OVERFLOW on the MIC is the much larger MPI communication time for boundary exchange.

In addition to the previously mentioned optimizations, we also developed and implemented a strategy for balancing the workload between host and MIC depending on their compute power. OVERFLOW has an internal load balancing mechanism that assumes the processors are equally powerful. This mechanism was modified to account for processors of different strengths. The modification consisted of writing a file containing timing data for each processor. If nothing is known a priori, a cold start can be made, running a small number of steps, with load balancing assuming the processors have the same strength. Then a warm start can be made and the file containing timing data is used in the load-balancing algorithm to allow for processors of different strengths. If a priori information is available, then a file containing mock timing data can be constructed by hand and it will be used by the code. In Figure 5 and in rest of the paper, “cold start” means without reading timing data file, and “warm start” means with reading timing data file.

a) *DLRF6-Medium*

Figure 7 shows cold vs. warm start performances using the DLRF6-Medium data set for various MPI+OpenMP combinations in symmetric mode:  $2 \times 8 + 2 \times 116$  (232 threads—close to 4 threads per core),  $2 \times 8 + 4 \times 56$  (224),  $2 \times 8 + 6 \times 36$  (216), and  $2 \times 8 + 8 \times 28$  (224). The best result is for  $2 \times 8 + 6 \times 36$  (216) and it is 38% better than the worst result; so for each application some experimentation is required to find the optimal combination of MPI and OpenMP threads and load balancing.

b) *DLRF6-Large*

Figure 8 shows the performance using the DLRF6-Large data set. Here we notice that performance on MIC for warm start improves as the number of OpenMP threads decreases; the worst performance is for 116 OpenMP threads and the best is for 56 OpenMP threads. Performance gain due to load balancing is 10%.

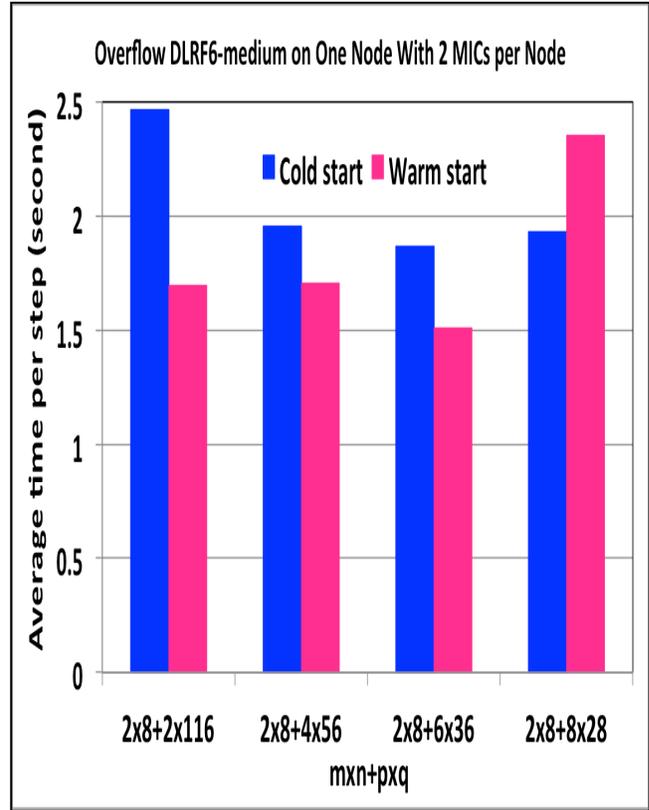


Figure 7. Performance of OVERFLOW on a host + MIC0 + MIC1.

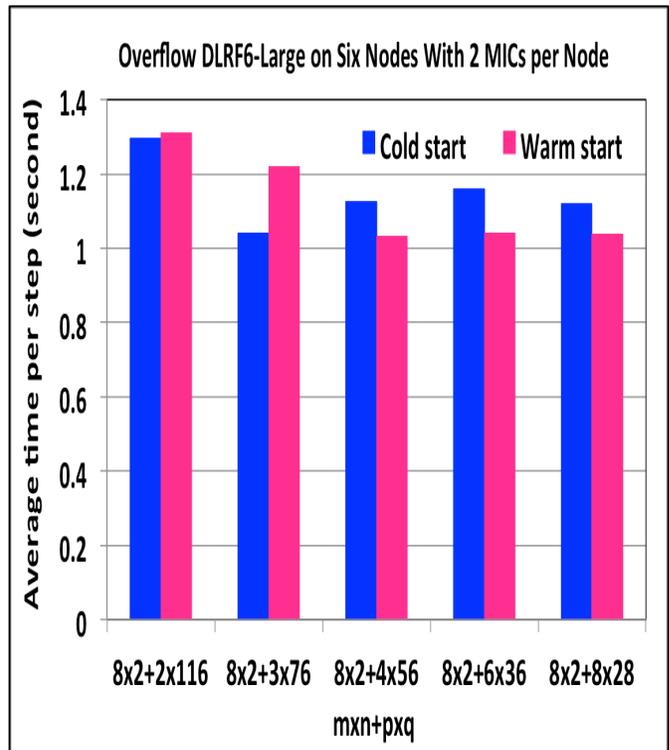


Figure 8. Performance of OVERFLOW DLRF6-Large on 6 nodes.

c) DPW3:

Figure 9 shows the performance using the DPW3 data set. DPW3 is a finer-grid version of the DLRF6-Large case. It is a wing-body geometry with 83 million grid points before grid splitting, i.e., the number of grid points is 2.3 times larger than the DLRF6-Large case. In this case, performance increases as the number of OpenMP threads increases because the number of grid points is large enough to keep all the threads busy. The best performance is for 2 MPI processes and 116 OpenMP threads per MPI process, and the performance gain due to balancing the number of MPI processes vs. number of OpenMP threads is more significant than for the DLRF-Large case.

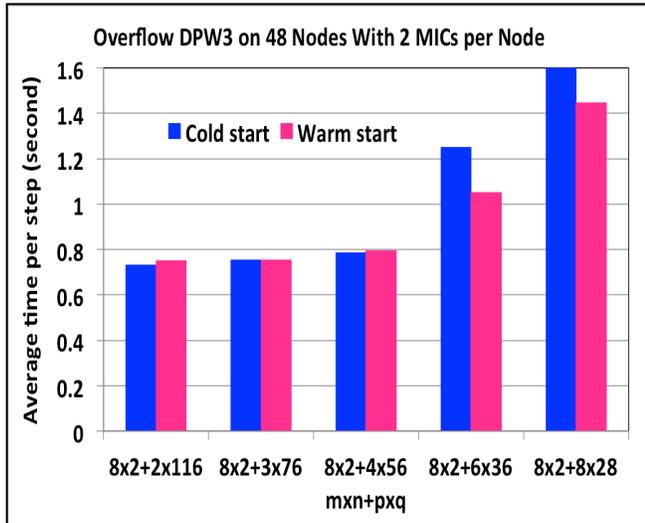


Figure 9. Performance of OVERFLOW DPW3 on 48 nodes each with two MICs per node.

d) Rotor

Figure 10 shows the performance using the NAS Rotor data set. There are 91 million grid points before grid splitting. Like DPW3, here the performance also increases as the number of OpenMP threads increases.

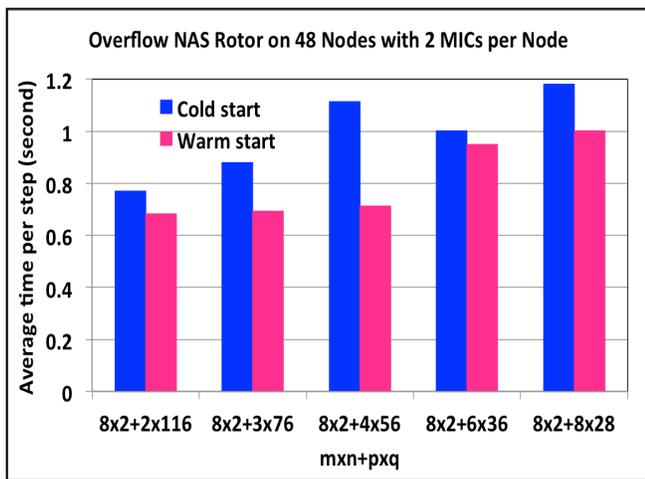


Figure 10. Performance of OVERFLOW Rotor on 48 nodes with two MICs per node.

Figure 11 shows the percentage improvements due to load balancing for three different data sets—DLRF6-large on 6 nodes, DPW3 on 48 nodes and NAS Rotor on 48 nodes. Largest gain (5% to 35%) is for NAS Rotor on 48 nodes, maximum being for 4 MPI and 56 OpenMP threads (224 threads) and lowest is 6 MPI and 36 OpenMP threads. Performance gain is from -1% to 17% for DPW3 on 48 nodes, maximum for 6 MPI and 36 OpenMP threads (216 threads). Among three data sets, DLRF6-Large on 6 nodes gains the least advantage from load balancing and in fact our load balancing effort has negative impact for small number of OpenMP threads.

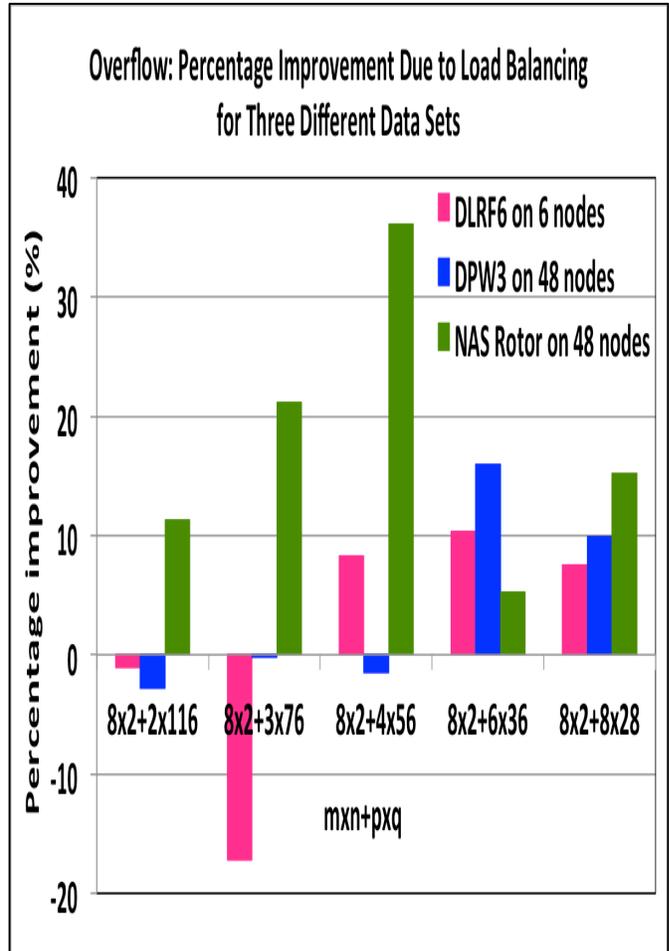


Figure 11. Percentage improvement of OVERFLOW by load balancing for three cases.

2) WRF

In this subsection we present results of WRF for both single node and multi-nodes.

a) Single Node

We present single node results for host-native, MIC-native and symmetric modes. Table 1 shows the performance of original and optimized WRF 3.4 on single node of Maia.

**Host-Native Mode:** Host-native results are shown for both original WRF 3.4 and optimized WRF 3.4 and used 16 MPI

processes with 1 OpenMP thread per process ( $16 \times 1$ ). Both versions were compiled with AVX instruction (256-bit vector width). The performance difference between the two is less than 3%.

**MIC-Native Mode:** MIC-native results are shown for WRF 3.4 with NCAR default compiler flags or with special flags for the MIC. The MIC special flags are:

```
-mP2OPT_hlo_fusion=F -mP2OPT_hpo_vec_check_dp_trip=F
-mGLOB_default_function_attrs="knc_stream_store_controls=2"
-fimf-precision=low -fimf-domain-exclusion=15
```

Rows 1 and 2 of Table 1 compare the results of original WRF 3.4 and optimized WRF3.4 on the host using 16 MPI threads and 1 OpenMP thread. The performance gain of the optimized version is only 2%.

Comparing the 3<sup>rd</sup> and 4<sup>th</sup> rows of Table 1, one can see that the MIC special flags provide a nice speed up of almost a factor of 2 for the 64-thread case with both MICs running the  $32 \times 1$  combination.

Rows 5 and 6 compare the results for 224 threads run with either all threads on MIC0 ( $8 \times 28$ ) or split evenly between MIC0 and MIC1 ( $4 \times 28$  on both MIC0 and MIC1). Using 2 MICs improves the performance by 18%. So, the benefit of

additional memory bandwidth using 2 MICs outweighs the increased communication costs across the MICs.

**Symmetric Mode:** Symmetric mode results are shown for host + 1 MIC for both original NCAR WRF 3.4 and optimized WRF 3.4 and for the optimized WRF 3.4 in the host +2 MICs configuration. Optimization transformed the code for intensive vectorization, improved OpenMP use, loop fusion, and used a modified shared memory tiling algorithm so that tiles are calculated only once per zone per domain including thread packing and unpacking of MPI messages. The optimized version has several “collapsed DO loops” as well. Most of the optimization is in subroutine WSM5 for vectorization and data alignment.

Rows 7 and 8 compare the performance of the optimized WRF 3.4 versus the original, both using MIC special flags and one host + one MIC in the ( $8 \times 2$ ) host + ( $7 \times 34$ ) MIC combination. The code optimization provided a 46.6% reduction in wallclock time. Finally, for the one host+2 MICs combination, the wallclock time for the optimized WRF3.4 in symmetric mode is reduced by a third compared to the wallclock time for the original WRF3.4 on a single host (row 1).

TABLE I. PERFORMANCE OF ORIGINAL WRF 3.4 AND OPTIMIZED WRF 3.4 ON SINGLE NODE OF MAIA.

Row #	Version	Flags	Processor	Threads	MPI $\times$ OpenMP	Time (sec.)
1	Original	AVX	Host	16	$16 \times 1$	147.77
2	Optimized	AVX	Host	16	$16 \times 1$	144.40
3	Original	Default	MIC0 + MIC1	64	$2 \times (32 \times 1)$	774.48
4	Original	MIC	MIC0 + MIC1	64	$2 \times (32 \times 1)$	404.15
5	Original	MIC	MIC0	224	$8 \times 28$	340.92
6	Original	MIC	MIC0 + MIC1	224	$2 \times (4 \times 28)$	281.15
7	Original	MIC	Host + MIC0	$16 + 238$	$8 \times 2 + 7 \times 34$	205.42
8	Optimized	MIC	Host + MIC0	$16 + 238$	$8 \times 2 + 7 \times 34$	109.76
9	Optimized	MIC	Host + MIC0 + MIC1	$16 + 400$	$8 \times 2 + 2 \times (4 \times 50)$	98.09

*b) Multiple nodes*

Figure 12 shows the performance of optimized hybrid (MPI + OpenMP) WRF 3.4 for a number of nodes ranging from 1 to 3. The notation  $1 \times 16 \times 1$  means one host node with 16 MPI processes and 1 OpenMP thread per process. The performance is better if one uses 2 OpenMP threads instead of one per MPI process, e.g., for 2 hosts, the performance using  $2 \times 8 \times 2$  is higher than using  $2 \times 16 \times 1$  by 2.5%; for three hosts, the performance using  $3 \times 8 \times 2$  is higher than using  $3 \times 16 \times 1$  by 7%. Overall, scaling on the hosts is very good.

Figure 12 also shows the performance of WRF 3.4 run in symmetric mode. The performance using host+MIC0 is 24% better than using only one host (16 threads), but this is reversed in going to multiple nodes where the performance of two hosts ( $2 \times 8 \times 2$ ) is better than that of 2 hosts + 4 MICs run in the  $2 \times (8 \times 2 + 4 \times 50 + 4 \times 50)$  combination. Similarly, performance of 3 hosts is better than the performance of 3 hosts + 3 MIC0 + 3 MIC1. The main reason for the poor performance of symmetric mode for multiple nodes is the very low communication bandwidth for multiple nodes.

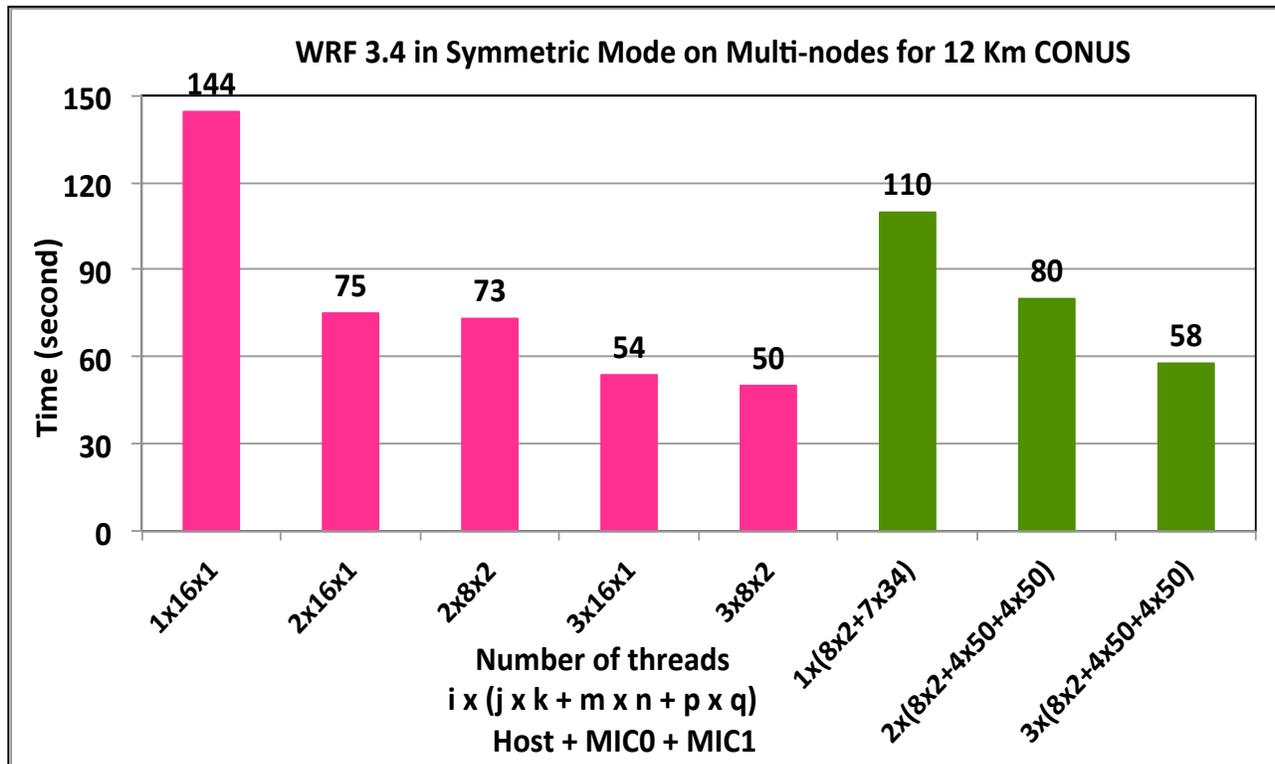


Figure 12. Performance of optimized WRF 3.4 in symmetric mode on multi-node of Maia. Red and green color bars denote HOST and HOST+MIC0+MIC1 respectively.

## VII. CONCLUSIONS

In this paper we studied the multi-node performance of Maia, an IB-connected cluster of nodes with Sandy Bridge hosts and KNC coprocessors. We ran a number of benchmarks ranging from many variations of the NPBs to real world applications. We optimized two full-scale production quality applications — WRF3.4 and OVERFLOW. We tested four programming modes: host-native, MIC-native, offload, and symmetric.

Optimization of an application on MIC is very challenging and time consuming and requires understanding of its architecture as well as that of the application. We did the following four types of optimizations: vectorization, algorithm, MPI communication, and load balancing.

If the application is not highly vectorized to use 512-bit wide vector units that can execute 8 double-precision SIMD instructions in a single clock, then the performance of the application is extremely poor on MIC. In view of this we spent a significant amount of time to vectorize WRF and OVERFLOW. We profiled the two applications to find out the most time-intensive subroutines. For WRF we found that the WSM3 subroutine uses the most compute time, so we concentrated all our optimization efforts in optimizing it. General optimizations we performed included “collapsed DO loops”, loop fusion and data alignment.

For WRF we modified the shared-memory tiling algorithm so that tiles are calculated only once per zone per

domain. For OVERFLOW we modified the code for OpenMP parallelism over strips of a plane instead of over a full plane.

Applications with significant amounts of MPI communication, especially collective communication, perform very poorly on MIC because the performance of MPI functions is 3 to 20 times slower for intra-MIC and 10 to 60 times slower for inter-MIC communication as compared to host [13]. To reduce MPI communication time, we performed optimization by packing and unpacking the MPI messages.

In symmetric mode, workload is proportionally distributed among hosts and MICs by taking into account the power of the processor, coprocessor and their memory. Hybrid (MPI + OpenMP) code is preferred for running in symmetric mode across coprocessors on multiple nodes to improve resource utilization and to reduce MPI communication, especially given that bandwidth from a MIC on one host to a MIC on another host may be limited to a maximum of only measured 950 MB/s as opposed to bandwidth of 6 GB/s for the same host. Our results for OVERFLOW and WRF 3.4 in symmetric mode indicate that performance to a large extent depends on the optimal number of MPI processes and OpenMP threads, and to determine the right combination one needs to experiment. Load balancing in symmetric mode is challenging and critical as is evident from our OVERFLOW and WRF 3.4 results. Compounding the problem is the fact that slow MPI communication across nodes can negate any gains in computational efficiency

through code optimization. This is especially clear in the OVERFLOW results where time spent in boundary exchange is separated out from some of the more compute-intensive parts of the code. Similarly for WRF 3.4., the Intel optimized version of the code improved performance on a single host + 2 MICs by a third compared to the original NCAR version running on a single host. However, when scaling beyond a single node, the advantage of running symmetrically across host and MICs is quickly lost compared to running natively on the host.

Intel's optimized WRF3.4 code for MIC runs 47% faster than the original NCAR WRF 3.4, and our optimized OVERFLOW runs 18% faster on the host and the load-balancing strategy used improved the performance on MIC by 5% to 36% depending on the data size.

Performance of MPI applications in MIC-native mode is much lower than in host-native mode. Getting good performance on the MIC in native mode is not an easy task. It requires careful design of data structures and memory layout together with enabling lots of parallelism as done in optimizing OVERFLOW and WRF3.4. Both pure MPI and OpenMP codes can run on one MIC card but only the former can run on more than one MIC card. Performance of MIC-native mode across two or more nodes degrades quickly due to the low inter-node, inter-MIC bandwidth especially if it involves MIC1, as in our cluster — this is a serious problem and needs to be addressed by Intel in the next generation of Xeon Phi.

Offload mode is an attractive solution for performing compute intensive tasks on the MIC while performing I/O and serial or less parallel computations on the host. However, there is a significant overhead to using the offload mode as seen by our offload versions of the BT and SP compact applications. One should very carefully select the granularity of the offloads to offset the overhead of the data transfer with the efficiency gained by execution on the MIC.

Recently, Intel announced the next generation of Xeon Phi called “Knights Landing” (KNL), whose processor will be based on “Atom” rather than on “Pentium” and will include out-of-order execution. The improvements such as gather/scatter in hardware instead of software, and of the compute cores—especially improved branch prediction and L1 hardware prefetching—should be very beneficial for performance. The most important architectural feature of the KNL is that it will be a bootable processor and not a coprocessor. As a result, it will no longer be subject to the bottleneck of a PCIe link between processor and coprocessor. In KNL, it will not be necessary to use a minimum of two hardware threads per MIC core, as instructions will be issued every cycle instead of every other cycle. The DOE has ordered a system based on 3-Tflop/s KNL that uses a Micron Hybrid Memory Cube (HMC) technology with 15 times more memory bandwidth than DDR3 [19, 20]. We look forward to benchmarking a KNL-based system and observing the performance improvements attributable to these changes.

## REFERENCES

- [1] Ten Year US Exascale Roadmap Crystalizes: <http://www.hpcwire.com/2014/01/17/ten-year-us-exascale-roadmap-crystalizes/>
- [2] Top500 List – November 2014, <http://www.top500.org/list/2014/06/>
- [3] WHAT IS GPU ACCELERATED COMPUTING?, <http://www.nvidia.com/object/what-is-gpu-computing.html>
- [4] Intel® Xeon Phi™ Product Family, <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>
- [5] NSCC-TJ *National Supercomputing Center* in Tianjin, <http://www.nscj-tj.gov.cn/en/>
- [6] STAMPEDE - Dell PowerEdge C8220 Cluster with Intel Xeon Phi coprocessors, <https://www.tacc.utexas.edu/resources/hpc;jsessionid=F87EF06DF1AD48342098B580E2B6430A.jvm1>
- [7] NERSC, Cray, Intel to Collaborate on Next-Generation Supercomputer, <http://cs.lbl.gov/news-media/news/2014/ner-sc-cray-intel-to-collaborate-on-next-generation-supercomputer/>
- [8] NERSC Supercomputer First to Use Intel's Next-Gen 'Knights Landing', <http://goparallel.sourceforge.net/ner-sc-supercomputer-first-use-intels-next-gen-knights-landing/>
- [9] US chases supercomputing crown with multi petaflop Trinity system, [http://www.techworld.com.au/article/549768/us\\_chases\\_supercomputing\\_crown\\_multipetaflop\\_trinity\\_system/](http://www.techworld.com.au/article/549768/us_chases_supercomputing_crown_multipetaflop_trinity_system/)
- [10] Jongsoo Park, Ganesh Bikshandi, Karthikeyan Vaidyanathan, Ping Tak Peter Tang, Pradeep Dubey, Daehyun Kim, Tera-Scale 1D FFT with Low-Communication Algorithm on Intel Xeon Phi, Denver, ACM SC13, CO, USA — November 17 - 21, 2013
- [11] Joo, D.D. Kalamkar. K. Vaidyanathan, M. Smelyanskiv, K. Pammany, V. W. Lee, P. Dubey, W. Watson III. “Lattice QCD on Intel Xeon Phi Coprocessors” 28th International Supercomputing Conference, ISC 2013, Leipzig, Germany, June 16-20, 2013. Proceedings Supercomputing Lecture Notes in Computer Science Vol. 7905, pp 40-54 2013. <https://software.intel.com/sites/default/files/article/401382/qcd-isc2013.pdf>.
- [12] Heinecke, A., Vaidyanathan K., M. Smelyanskiy, Kobotov A., Dubtsov R., Henry G., Chrysos G., and Dubey P., Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel Xeon Phi coprocessor, IPDPS, Boston, 2013,
- [13] Saini, S., Jin, H., Jespersen, D., Feng, H., Djomehri, J., Arasin, W., ... & Biswas, R. An early performance evaluation of many integrated core architecture based SGI rackable computing system. *SC13*, Denver, CO, USA — November 1 Denver, CO, USA — November 17 - 21, 2013.7 - 21, 2013.
- [14] Florian Wende and Thomas Steinke. 2013. Swendsen-Wang multi-cluster algorithm for the 2D/3D Ising model on Xeon Phi and GPU. *SC13*, Denver, CO, USA — November 1 Denver, CO, USA — November 17 - 21, 2013.7 - 21, 2013
- [15] Pennycook, S.J., Hughes, M. Smelyanskiy, C.J., and Jarvis, S.A., “Exploring SIMD for molecular dynamics, using Intel Xeon processors and Intel Xeon Phi coprocessors. IEEE IPDPS 2013, May Boston, USA
- [16] NAS Parallel Benchmarks, <http://www.nas.nasa.gov/publications/npb.html>
- [17] OVERFLOW, <http://aac.larc.nasa.gov/~buning/>
- [18] WRF Model Version 3.4: UPDATES, <http://www2.mmm.ucar.edu/wrf/users/wrfv3.4/updates-3.4.html>
- [19] Micron's revolutionary Hybrid Memory Cube tech is 15 times faster than today's RAM <http://www.pcworld.com/article/2366680/computer-memory-overhaul-due-with-microns-hmc-in-early-2015.html>
- [20] Intel's Next-Gen Xeon Phi (Knights Landing) to Use Silicon Photonics, <http://www.datacenterknowledge.com/archives/2014/06/24/next-gen-intel-phi-coprocessor-to-use-silicon-photonics-interconnect/>