PARALLELIZATION AND PERFORMANCE OF THE NIM WEATHER MODEL ON CPU, GPU, AND MIC PROCESSORS

Mark Govett, Jim Rosinski, Jacques Middlecoff, Tom Henderson, Jin Lee, Alexander MacDonald, Ning Wang, Paul Madden, Julie Schramm, and Antonio Duarte

Next-generation supercomputers containing millions of processors will require weather prediction models to be designed and developed by scientists and software experts to ensure portability and efficiency on increasingly diverse HPC systems.

A new generation of high-performance computing (HPC) has emerged called fine grain or massively parallel fine grain (MPFG). The term “massively parallel” refers to large-scale HPC systems containing tens of thousands to millions of processing cores. “Fine grain” refers to loop-level parallelism that must be exposed in applications to permit thousands to millions of arithmetic operations to be executed every clock cycle. Two general classes of MPFG chips are available: Many Integrated Core (MIC) from Intel and graphics processing units (GPUs) from NVIDIA and Advanced Micro Devices (AMD) (see “Many-core and GPU computing explained” sidebar). In contrast to up to 18 cores on the latest-generation Intel Broadwell CPUs, these MPFG chips contain hundreds to thousands of processing cores. They provide 10–20 times greater peak performance than CPUs, and they appear in systems that increasingly dominate the list of top supercomputers in the world (Strohmaier et al. 2016). Peak performance does not translate to real application performance, however. Good performance can only be achieved if fine-grain parallelism can be found and exploited in the applications. Fortunately, most weather and climate codes contain a high degree of parallelism, making them good candidates for MPFG computing.

As a result, research groups worldwide have begun parallelizing their weather and climate prediction models for MPFG processors. The Swiss National Supercomputing Center (CSCS) has done the most comprehensive work so far. They parallelized the dynamical core of the Consortium for Small-Scale Modeling (COSMO) model for GPUs in 2013 (Fuhrer et al. 2014). At that time, no viable commercial FORTRAN GPU compilers were available, so the code was rewritten in C++ to enhance performance and portability. They reported the C++ version gave a 2.9-times speedup over the original FORTRAN code using same-generation dual-socket Intel Sandy Bridge CPU and Kepler K20x GPU chips. Parallelization of model physics in 2014 preserved the original FORTRAN code using industry standard open accelerator (OpenACC) compiler directives for parallelization (Lapillonne and Fuhrer 2014). The entire model, including data assimilation, is now running...
operationally on GPUs at the Swiss Federal Office of Meteorology and Climatology (MeteoSwiss).

Most atmospheric modeling groups exploring MPFG focused on the parallelization of model dynami-
cs. The German Weather Service (DWD) and Max Planck Institute for Meteorology developed the Icosa-
hedral Nonhydrostatic (ICON) dynamical core, which has been parallelized for GPUs. Early work by Sawyer et al. (2011) converted the FORTRAN using NVIDIA-specific Compute Unified Device Archi-
tecture (CUDA)-FORTRAN and open computing language (OpenCL) demonstrated a 2-times speedup
over dual-socket CPU nodes. The invasive, platform-
specific code changes were unacceptable to domain
scientists, so current efforts are focused on minimal changes to the original code using OpenACC for par-
allelization (Sawyer et al. 2014). Another dynamical core, the Nonhydrostatic Icosahedral Atmospheric
Model (NICAM), has been parallelized for GPUs, with a reported 7–8-times performance speedup
comparing two 2013-generation Kepler GPUs to one 2011-generation dual-socket Intel Westmere CPU
(Yashiro et al. 2014). Other dynamical cores parallel-
ized for the GPU, including the finite volume cubed
(FV3) model (Lin 2004) used in Goddard Earth
Observing System Model, version 5 (GEOS-5) (Put-
nam et al. 2011), and the Met Office High Resolution
Environment (HOMME) (Carpenter et al. 2013), have shown some speedups versus the CPU.

Collectively, these experiences show that porting
codes to GPUs can be challenging, but most users
have reported speedups over CPUs. Over time, more
mature GPU compilers have simplified parallelization
and improved application performance. However, reporting of results has not been uniform and can be
misleading. Ideally, comparisons should be made us-
ing the same source code, with optimizations applied
faithfully to the CPU and GPU, and run on same-
generation processors. When codes are rewritten, it
becomes harder to make fair comparisons between
multiple versions must be maintained and optimized. When different-generation hardware is used (e.g.,
2010 GPUs vs 2013 GPUs), adjustments should be made to nor-
malize reported speedups. Similarly, comparisons are made with multiple GPUs attached to a single
node, further adjustments should be made. Finally,
comparisons between a GPU and a single CPU core
give impressive speedups of 50–100 times, but such
results are not useful or fair and require adjustment
to factor in use of all cores available on the CPU.

When Intel released its MIC processor, called
Knights Corner (KNC), in 2013, a new influx of re-
searchers began exploring fine-grain computing. Re-
search teams from National Center for Atmospheric
Research (NCAR)’s Community Earth System Model
(CESM) (Kim et al. 2013), Weather Research and
Forecasting (WRF) Model (Michalakes et al. 2016),
and the FV3 (Nguyen et al. 2013) reported little to no
performance gain compared to the CPU. A more pre-
hensive parallelization for the MIC with National
Oceanic and Atmospheric Administration (NOAA’s)
Flow-Following Finite Volume Icosahedral Model
(FIM) (Bleck et al. 2013) included dynamics and phys-
ics running on the MIC (Rosinski 2015). Execution of the entire model on the KNC gave no performance
benefit compared to the CPU. A common sentiment in
these efforts is that porting applications to run on
the MIC is easy, but getting good performance with
MIC was difficult. This is changing with the release
of Intel’s Knights Landing (KNL) processor in early
2016. Research groups are now reporting 2-times or
even more improvement in application performance
for KNL versus the CPU.

This paper describes the development of the non-
hydrostatic icosahedral model (NIM), a dynamical
core that was designed to exploit MPFG processors.
The NIM was initially designed for NVIDIA GPUs in
2009. Since commercial FORTRAN GPU compilers
were not available at that time, the FORTRAN-to-
CUDA accelerator (F2C-ACC) (Govett et al. 2010)
was co-developed with NIM to convert FORTRAN
code into CUDA, a high-level programming language
used on NVIDIA GPUs (NVIDIA 2015). The F2C-
ACC compiler has been the primary compiler used
for execution of NIM on NVIDIA GPUs and has

Fig. 1. An illustration contrasting the converging grid points of a lat–lon grid vs the nearly uniform grid spacing
of an icosahedral–hexagonal grid.

served as a benchmark for evaluation of commercial
OpenACC compilers from Cray and The Portland Group International (PGI). Using the same source
code, the NIM was ported to Intel MIC in 2013 when
these processors became available.

We believe NIM is currently the only weather
model that runs on CPU, GPU, and MIC processors
with a single-source code. The dynamics portion of
NIM uses open multiprocessing (OpenMP) (CPU and
MIC), OpenACC (GPU), and F2C-ACC (GPU) direc-
tives for parallelization. Scalable Modeling System
(SMS) directives and run-time library support MPI-
based distributed-memory parallelism, including do-
main decomposition, interprocess communications,
and input/output (I/O) operations (Govett et al. 2003).
Collectively, these directives allow a single-source
code to be maintained capable of running on CPU,
GPU, and MIC processors for serial or parallel execu-
tion. Further, the NIM demonstrates efficient parallel
performance and scalability to tens of thousands of
compute nodes and has been useful for comparisons
between CPU, GPU, and MIC processors (Govett et al.

MODEL DESIGN. NIM is a multiscale model,
which has been designed, developed, and run
globally at ~3-km resolution with a goal to improve
medium-range weather forecasts. The model was
designed to explicitly permit convective cloud systems
without cumulus parameterizations typically used
in models run at coarser scales. In addition, NIM has
extended the conventional two-dimensional
finite-volume approach into three-dimensional
finite-volume solvers designed to improve pressure
calculation and orographic precipitation over
complex terrain.

NIM uses the following innovations in the model
formulation:

• a local coordinate system that remaps a spherical
  surface to a plane (Lee and MacDonald 2009);
• indirect addressing of grid cells to simplify the
code and improve performance (MacDonald et al.
2011);
• flux-corrected transport formulated on finite-
  volume operators to maintain conservative and
  monotonic transport (Lee et al. 2010);
• all differentials evaluated as finite-volume inte-
  grals around the cells, and
• icosahedral–hexagonal grid optimization (Wang
  and Lee 2011).

The icosahedral–hexagonal grid is a key part of the
model. This formulation approximates a sphere
with a varying number of hexagons but always includes
12 pentagons. (Sadourny et al. 1968; Williamson
1971). The key advantage of this formulation is the evenly
uniform grid areas that are possible over a sphere as
illustrated in Fig. 1. This is in contrast to the latitude-
longitude models that have dominated global weather
and climate prediction for 30 years. The nearly uni-
form grid represents the poles. Yet the notorious “pole problem” inherent in latitude–longitude grids,
where meridians converge toward the poles.
MANY-CORE AND GPU COMPUTING EXPLAINED

Many core and GPUs represent a new class of computing called MPFG. In contrast to CPU chips with up to 18 cores, these fine-grain processors contain hundreds to thousands of computational cores. Each individual core is slower than a traditional CPU core, but there are many of them available to execute instructions simultaneously. This has required model calculations to become increasingly fine grained. GPUs are designed for compute-intensive, highly parallel execution. GPUs contain up to 5,000 compute cores that execute instructions simultaneously. As a coprocessor to the CPU, work is given to the GPU in routines or blocks on the GPU. Gang parallelism is for single instruction multiple data (SIMD) or vector parallelism that is executed on the hardware simultaneously. MIC hardware from Intel also provides the opportunity to exploit more parallelism than traditional CPU architectures. Like GPUs, the clock rate of the chips is 2–3 times slower than current-generation CPUs, with higher peak performance provided by additional processing cores, wider vector processing units, and a fused multiply–add (FMA) instruction. Programming models used to express parallelism on MIC hardware is OpenMP threading along with vectorization. Users can write to offload computationally intensive calculations from the CPU to the MIC (similar to GPU), run in MIC-only mode, or share between MIC and CPU host.

NIM uses a fully three-dimensional finite-volume discretization scheme designed to improve pressure gradient over complex terrain. Three-dimensional finite-volume operators also provide accurate and efficient tracer transport essential for next-generation global atmospheric models. Prognostic variables are allocated at horizontal cell centers (Arakawa and Lamb 1977). This simplifies looping constructs and reduces data dependencies in the code. The numerical scheme uses a local coordinate system remapped from the spherical surface to a plane at each grid cell. All differentials are organized with the vertical dimension innermost in dynamics routines. This organization effectively amortizes the cost of the indirect access of grid cells over the 96 vertical levels. Testing during model development verified there was less than 1% performance penalty using this approach (MacDonald et al. 2011). NIM dynamics executes completely on the GPU. Model state remains resident in GPU global memory. Data are only copied between CPU and GPU for coarse-grain parallelism. 2) minimize data movement between the GPU and CPU. Grid cells can be stored in any order because a lookup table is used to indirectly access neighboring grid cells and edges on the icosahedral—hexagonal grid. The model’s loop and array structures are optimized for the vertical dimension in memory. These optimizations improve performance and improve parallelism. To minimize data movement and improve parallelism, NIM is a FORTRAN code containing a mix of FORTRAN 77 and FORTRAN 90. The FORTRAN 77 contains 512-bit vector registers. As a result, vectorization has been done to accelerate both the CPU and GPU. Short loops are not used to derive types, pointers, or other constructs that can be challenging for compilers to support or run efficiently. The SMS library used by NIM for coarse-grain parallelism employs the message passing interface (MPI) library to implement thread-level parallelism and interprocess communications, reductions, and other MPI operations. NIM was designed from the outset to maximize fine grain or loop-level parallel computational capability of both NVIDIA GPU and Intel MIC architectures. Primary model computations are organized as simple dot products or vector operations and loops with no data-dependent conditionals or branching. The NIM dynamical core requires only single-precision floating-point computations and runs well on the CPU, achieving 10% of peak performance on an Intel Haswell CPU.

Data movement between the CPU and GPU are handled automatically by the run-time system. However, copying data between the host (CPU) and device (GPU) is slow, so minimizing data movement is an important optimization needed to improve performance. The data directive can be used to manage data movement between the CPU and GPU explicitly. Managing data movement explicitly is expected to diminish with the introduction of unified memory in Pascal-generation chips. Unified memory is a way to programmatically treat CPU and GPU memory as a single large memory on NVIDIA hardware. Using NVIDIA’s proprietary hardware called NVLink, the GPU can access CPU memory at the same speed as the CPU would, further reducing the requirement to manage data movement explicitly.

PARALLELIZATION. NIM uses standards-compliant OpenMP (for CPU and MIC) and OpenACC (for GPU) directives for parallelization. OpenMP is the de facto standard for shared memory programming on the CPU and MIC processors, with recent extensions to support attached devices such as GPUs. OpenACC was developed initially to support GPUs, with more recent support for CPU (x86) and MIC processors. Both standards are striving toward performance portability, where a single set of directives is sufficient to run efficiently on CPU, GPU, MIC, and other processors. Until recently, FACC was the primary compiler being used to parallelize and run NIM on NVIDIA GPUs. FACC-ACC was an effective way to push for improvements in commercial FORTRAN GPU compilers. Prior evaluation of OpenACC compilers and their predecessors was done in 2011 (Compiler and Architecture for Embodied and Superscalar Processors (CAPS), PGI (Henderson et al. 2011), 2013 (PGI, Cray) (Govett 2013), and 2014 (PGI, Cray) (Govett et al. 2014). These evaluations exposed bugs and performance problems in the compilers. The problems identified have been corrected, making FACC-ACC no longer necessary.

OpenACC. GPU parallelization can be done in three phases: 1) define GPU kernels and identifying loop-level parallelism, 2) minimize data movement between the GPU and CPU, and 3) optimize performance. GPU kernels are regions of code, identified with the parallel or kernels directive, that are executed on the GPU. Loop-level parallelism is prescribed using the loop directive, with the optional key words gang,工作组, or vector, vectorize to specify the type of parallelism desired. These directives are generally sufficient to parallelize and run applications on GPUs. Further work involves optimization to minimize data movement and improve parallel performance.

Data movement between the CPU and GPU are handled automatically by the run-time system. However, copying data between the host (CPU) and device (GPU) is slow, so minimizing data movement is an important optimization needed to improve performance. The data directive can be used to manage data movement between the CPU and GPU explicitly. Managing data movement explicitly is expected to diminish with the introduction of unified memory in Pascal-generation chips. Unified memory is a way to programmatically treat CPU and GPU memory as a single large memory on NVIDIA hardware. Using NVIDIA’s proprietary hardware called NVLink, the GPU can access CPU memory at the same speed as the CPU would, further reducing the requirement to manage data movement explicitly.

OpenMP. Parallelization for the CPU and MIC involves two steps: 1) insert OpenMP directives to identify thread-level parallelism and 2) optimize performance. Loop optimizations are organized in NIM with thread- over the single horizontal dimension and vectorization over the generally independent vertical dimension. Threading of the horizontal loop is normally outside of the vertical loops and, if applicable, loops over cell edges. Most OpenMP loops in NIM contain sufficient work to amortize the overhead of assigning work to threads on loop start-up and thread synchroniza- tion at the end of the threaded region. These costs are generally higher on the MIC than on the CPU because there are more threads to manage. Vectorization is an optimization where independent calculations executed serially within a loop can be executed simultaneously in hardware by specially design- nated vector registers available to each processing core. Intel compilers automatically attempt vectorization, with compiler flags available for further optimization on specific hardware. The number of operations that can be executed simultaneously is based on the length of the vector registers. On the CPU, vector registers are currently 256 bits in length; the KNC MIC coprocessor contains 512-bit vector registers. As a result, vectoriza- tion provided some benefit on the host, but in most cases, it provided a greater improvement on the MIC.

PERFORMANCE. The NIM has demonstrated good performance and scaling on both CPUs and GPUs on Titan, where it has run more than 400,000 CPU core and 250,000 GPU core simulations. It has also been run on up to 320 Intel MIC (Xeon Phi) processors at the Texas Advanced Computing Center (TACC). Optimizations targeting Xeon Phi and GPU have also improved CPU performance. Currently NIM has been run on up to 17,000 GPUs, managed by the U.S. Department of Energy’s Oak Ridge National Laboratory (ORNL). Titan is an AMD-GPU-based system containing over 17,000 GPUs, managed by the U.S. Department of Energy’s Oak Ridge National Laboratory (ORNL). Runes were made on Stampede, an Intel CPU-MIC system supported by the National Science Foundation (NSF). GPU performance relied on the F-ACC compiler. Based on our evaluations, we believe openACC compilers would yield similar results.
Device performance. Single-device performance is the simplest and most direct comparison of chip technologies. Figure 2 shows performance running the entire NIM dynamical core on five generations of CPU, GPU, and MIC hardware (see Table 1). CPU results are based on standard two-socket node configurations. A roughly 2-times performance benefit favoring accelerators is observed for 2010–16-generation GPU chips. CPU performance has continued to benefit from increasing cores per chip, improvements in memory speeds, and the introduction of advanced vector instructions. Both the KNC and KNL processors are faster than same-generation CPU chips, with the 2016 KNL processor giving a 2-times performance benefit versus the CPU. The NVIDIA Pascal processor is even better, giving a 2.5-times speedup over the CPU, and 1.3 times faster than the KNL.

While device comparisons are useful, they do not include the cost of a CPU host that is required by the GPU accelerator. This practical and economic consideration motivates further examination and performance comparisons with up to eight GPUs attached to a single CPU.

Single-node performance (GPU only). Compute nodes normally have two CPU sockets, memory, network interconnect (NIC), peripheral component interconnect express (PCIe) bus, and a motherboard. Deviations from this basic configuration are available but more expensive since the volumes manufactured are lower. Therefore, most computing centers use standard, high-volume parts that offer the best price performance. GPUs can be attached to these nodes and communicate with the CPU host via the PCIe bus. When more than two GPUs are attached to the host, they must share the PCIe bus, which can impact performance. More specialized solutions are available that improve communications performance. Figure 3 illustrates the architecture of a Cray Storm node, with eight attached accelerators (GPUs are shown, but MIC processors can also be used), and additional PCIe hardware. Communications between sockets are handled with Intel’s QuickPath Interconnect (QPI).

Figure 4 shows weak scaling performance as the number of GPUs per CPU node increases from two to eight on a Cray Storm system. These results primarily indicate PCIe bandwidth limitations on the Cray Storm system. An additional performance bottleneck may be the limited bandwidth of the single communications path off node that is shared by the eight attached accelerators. Alternative node configurations are available, including ones with multiple InfiniBand (IB) connections, nested PCIe architectures, and solutions that avoid use of QPI because of reported latency issues (Ellis 2014; Cirrascale 2015). While such solutions can increase performance, testing remains the best way to measure cost–benefit.

Table 1. 2010–16-generation CPU, GPU, and MIC chips with corresponding numbers of processing cores. The number of cores for the CPU chips is based on two sockets.

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU: two sockets</th>
<th>Cores</th>
<th>GPU</th>
<th>Cores</th>
<th>MIC</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010/11</td>
<td>Westmere</td>
<td>12</td>
<td>Fermi</td>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Sandy Bridge</td>
<td>16</td>
<td>Kepler K20x</td>
<td>2,688</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>Ivy Bridge</td>
<td>20</td>
<td>Kepler K40</td>
<td>2,880</td>
<td>Knights Corner</td>
<td>61</td>
</tr>
<tr>
<td>2014</td>
<td>Haswell</td>
<td>24</td>
<td>Kepler K80</td>
<td>4,992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>Broadwell</td>
<td>30</td>
<td>Pascal</td>
<td>3,584</td>
<td>Knights Landing</td>
<td>68</td>
</tr>
</tbody>
</table>

Fig. 2. Run times for the NIM running at 240-km resolution (10,242 horizontal points, 96 vertical levels) for 100 time steps using CPU, GPU, and MIC chips identified in Table 1.

Fig. 3. Illustration of the Cray Storm node architecture containing eight accelerators per node. NVIDIA GPUs are shown, but other PCIe-compatible devices can be used. IB refers to a type of node interconnect called “InfiniBand”, others are also available.

Fig. 4. Communications scaling using 40 Pascal GPUs with 2–8 GPUs per node. Computation times for each of the 15- and 30-km runs are not shown but were consistently 28.6 and 7.5 s, respectively.

Scaling. To run efficiently on hundreds to thousands of processors requires good scaling. Both strong and weak scaling measures are useful for performance comparisons. Strong scaling is measured by applying increasing numbers of compute resources to a fixed problem size. This metric is particularly important for operational weather prediction where forecasts should run in under 1% of real time. The requirement is normally achieved by increasing the number of processors until the given time threshold is met. For example, a 1-day forecast that runs in 15 min represents 1% of real time; therefore, runs in 2% of real time would take 30 min.

Figure 5 shows multi-node scaling results for 20–160 Haswell CPUs (2015), NVIDIA Pascal GPUs (2016), and Intel KNL MIC (2016) processors. Up to 2.5-times speedup for GPU versus CPU is observed for
the 10-node result when 65,536 columns of work are given to each node or GPU. The decrease in scaling efficiency is almost completely due to interprocess communications overhead. For example, when communications are removed from the 80-GPU run, scaling efficiency increases from 63% to over 90%. CPU and MIC scaling also show similar degrading communications performance.

Weak scaling is a measure of how solution time varies with increasing numbers of processors when the problem size per processor and the number of model time steps remains fixed. It is considered a good way to determine how a model scales to high numbers of processors and is particularly useful for measuring communications overhead.

Table 2 gives performance results for a single node with a 2.284 column per GPU for 120- and 60-km resolution runs using two and eight GPUs. NVIDIA K80s packaged with two GPUs were used for the runs. Computation time was nearly identical for all runs, with communications time increasing to 3.19 s for the one-node, eight-GPU run. An additional run using two nodes illustrates the substantial increase in off-node communications time. Given communications time within a node (3.19 s) is less than the off-node time (7.23 s). The results show that more GPUs could be added to each node without adversely affecting model run times. This is because all processes must wait for the slowest communication to complete before model execution can continue.

**Spiral grid order.** To run efficiently on hundreds to thousands of nodes requires efficient interprocess communications. For most models, communications normally include gathering and packing data to be sent to neighboring processes, MPI communications of the data, and then unpacking and distributing the received data. Analysis of NIM dynamics performance showed that message packing and unpacking accounted for 50% of inter-GPU communications time (Middlecoff 2015). Since NIM relies on a lookup table to reference horizontal grid points, data can be reorganized to eliminate packing and unpacking. This optimization, configured during model initialization, is called “spiral grid order.” Figure 6 illustrates spiral grid ordering used in NIM. In the figure, points are organized according to where data must be sent (as interior points) or received (as halo points). Each point in the figure represents an icosahedral grid column that contains 96 vertical levels. The section labeled “spiral grid ordering” illustrates the method used to order points within each MPI task. The “data storage layout” section illustrates how grid points are organized in memory for optimal communications and computation. Use of the spiral grid order gave performance benefit on all architectures, with a 20% improvement in model run times on the GPU, 16% on the MIC, and 5% on the CPU.

**Cost–benefit.** Cost–benefit is determined using list prices as specified from Intel and NVIDIA in Table 3. The CPU node estimate was based on a standard two-socket, 24-core, Intel Haswell node, which includes the processor, memory, network interconnect, and warranty. The system interconnect was not included in cost calculations, based on the assumption that the cost for each system would be similar. While significant discounts are normally offered to customers, it would be impossible to fairly represent them in any cost–benefit evaluation here.

Figure 7 shows a cost–benefit based on running NIM dynamics at 30-km model resolution. Each of the five system configurations shown produced a 3-h forecast in 23 s or 0.20% of real time. The CPU-only configuration (upper-left point) required 960 cores or 40 Haswell nodes. The rightmost configurations used 20 NVIDIA K80 GPUs that were attached to 20, 10, 8, and 5 CPUs, respectively. The execution time of 23 s can be extrapolated to 1.6% of real time for a 3.75-km-resolution model when per-process work load remains fixed (weak scaling).³

³ Based on list prices in Table 3, a 40-node CPU would cost $260,000. Systems configured with 1–4 GPUs per node would offer a 32% cost savings.
DISCUSSION. The NIM demonstrates that weather prediction codes can be designed for high-performance and portability-targeting CPU, GPU, and MIC architectures. Inherent in the design of NIM has been the simplicity of the code, use of basic FORTRAN language constructs, and minimal branching in loop calculations. Use of FORTRAN pointers, derived types, and other constructs that are not well-supported or are challenging for compilers to analyze were avoided. NIM’s icosahedral–hexagonal grid permits grid cells to be treated identically, which minimizes branching in gridpoint calculations. Further, code design separated fine-grain and coarse-grain (MPI) parallelism. This was primarily due to limitations in F2C-ACC but had a benefit of organizing calculations to avoid creation and execution of small parallel regions, where synchronization and thread start-up (CPU, MIC) or kernel start-up (GPU) time can be significant.

The choice to organize arrays and loop calculations with an innermost vertical dimension and indirect addressing to access neighboring grid cells simplified code design without sacrificing performance. It also improved code portability and performance in unanticipated ways. First, the innermost vertical dimension was sufficient for CPU and MIC vectorization but essential for the GPU’s high-core-count devices. With few dependencies in the vertical dimension, vectorization (CPU, MIC) and thread parallelism (GPU) were consistently available in dynamic routines. Second, indirect addressing of grid cells gave flexibility and benefit in how they could be organized. As a result, spiral grid reordering eliminated MPI message packing and unpacking and decreased run times by up to 20%. Optimizations benefiting one architecture also helped the others. In the rare event performance degraded on one or more architecture, changes were reformulated to give positive benefit on all. For example, the use of F2C-ACC comparisons that exposed bugs and performance issues that were corrected. Parallelization is becoming simpler with OpenACC because data movement between CPU and GPU is managed by the runtime system. Unified memory on the GPU is expected to further simplify parallelization, narrowing the ease-of-use gap versus OpenMP.

The scope of this paper primarily focused on the dynamical core, largely because domain science teams had not adopted physics suite to use for high-resolution (<4 km) runs. Parallelization of select microphysics and radiation routines improved performance on all architectures, but lower speedups over the CPU were observed than for the dynamics routines (Henderson et al. 2015; Michalakes et al. 2016). This is likely due to more branching (i.e., if statements) in the code and less available parallelism in model physics than dynamics.

A paper gives a cost-benefit calculation for NIM dynamics that shows increasing value as more accelerators per node are used. However, there are several limitations in the value of these results. First, the comparison was only for model dynamics; when physics is included, model performance and cost-benefit favoring the GPU is expected to decrease. Second, use of list price is naive as vendors typically offer significant discounts, particularly for large installations. Third, calculations did not include the cost of the system interconnect. For small systems with tens of nodes, this was deemed acceptable for comparison as there would be little difference in price or performance. However, comparisons with hundreds to thousands of nodes would amplify the role of the interconnect and would need to be included in cost-benefit calculations.

CONCLUSIONS. The NIM is currently the only weather model capable of running on CPU, GPU, and MIC architectures with a single-source code. Performance of the NIM dynamical core was described. CPU, GPU, and MIC comparisons were made for device, node, and multinode performance. Device comparisons showed that NIM ran on the MIC and GPU 2.0 and 2.5 times faster, respectively, than the same-generation CPU hardware. The 2.0-times MIC speedup for KNL versus a dual-socket Broadwell GPU is a significant improvement over the previous generation KNC. Multinode scaling targeted a goal of running NIM at 3-km resolution in 1% of real time. The spiral grid ordering was described that eliminated data packing and unpacking and gave performance benefit on all architectures. Finally, a carefully crafted diagnostic analysis demonstrated increasing benefits favoring the K80 GPUs when up to eight accelerators are attached to each CPU host. Further analysis of cost-benefit using the latest Pascal and KNL chips is planned.

A critical element in achieving good performance and portability was the design of NIM. The simplicity of the code, looping, and array structures and the indirect addressing of the icosahedral grid were all chosen to expose the maximum parallelism to the un- derscored CPU and MIC vectorization but essential for the GPU’s performance. It also improved code portability and performance on all architectures, but lower speedups over the CPU were observed than for the dynamics routines (Henderson et al. 2015; Michalakes et al. 2016). This is likely due to more branching (i.e., if statements) in the code and less available parallelism in model physics than dynamics. A paper gives a cost-benefit calculation for NIM dynamics that shows increasing value as more accelerators per node are used. However, there are several limitations in the value of these results. First, the comparison was only for model dynamics; when physics is included, model performance and cost-benefit favoring the GPU is expected to decrease. Second, use of list price is naive as vendors typically offer significant discounts, particularly for large installations. Third, calculations did not include the cost of the system interconnect. For small systems with tens of nodes, this was deemed acceptable for comparison as there would be little difference in price or performance. However, comparisons with hundreds to thousands of nodes would amplify the role of the interconnect and would need to be included in cost-benefit calculations.

REFERENCES


Govett, J., and Coauthors, 2015: The NIM is currently the only weather model capable of running on CPU, GPU, and MIC architectures with a single-source code. Performance of the NIM dynamical core was described. CPU, GPU, and MIC comparisons were made for device, node, and multinode performance. Device comparisons showed that NIM ran on the MIC and GPU 2.0 and 2.5 times faster, respectively, than the same-generation CPU hardware. The 2.0-times MIC speedup for KNL versus a dual-socket Broadwell GPU is a significant improvement over the previous generation KNC. Multinode scaling targeted a goal of running NIM at 3-km resolution in 1% of real time. The spiral grid ordering was described that eliminated data packing and unpacking and gave performance benefit on all architectures. Finally, a carefully crafted diagnostic analysis demonstrated increasing benefits favoring the K80 GPUs when up to eight accelerators are attached to each CPU host. Further analysis of cost-benefit using the latest Pascal and KNL chips is planned.

A critical element in achieving good performance and portability was the design of NIM. The simplicity of the code, looping, and array structures and the indirect addressing of the icosahedral grid were all chosen to expose the maximum parallelism to the underscored CPU and MIC vectorization but essential for the GPU’s performance. It also improved code portability and performance on all architectures, but lower speedups over the CPU were observed than for the dynamics routines (Henderson et al. 2015; Michalakes et al. 2016). This is likely due to more branching (i.e., if statements) in the code and less available parallelism in model physics than dynamics. A paper gives a cost-benefit calculation for NIM dynamics that shows increasing value as more accelerators per node are used. However, there are several limitations in the value of these results. First, the comparison was only for model dynamics; when physics is included, model performance and cost-benefit favoring the GPU is expected to decrease. Second, use of list price is naive as vendors typically offer significant discounts, particularly for large installations. Third, calculations did not include the cost of the system interconnect. For small systems with tens of nodes, this was deemed acceptable for comparison as there would be little difference in price or performance. However, comparisons with hundreds to thousands of nodes would amplify the role of the interconnect and would need to be included in cost-benefit calculations.

REFERENCES


AMERICAN METEOROLOGICAL SOCIETY

OCTOBER 2017

AMERICAN METEOROLOGICAL SOCIETY

OCTOBER 2017

Ellis, S., 2014: Exploring the PCIe bus routes. Cirrascale.


Govett, M., 2013: Using OpenACC compilers to run FIM and NIM on GPUs. Third NCAR Multi-Core Workshop, Boulder, CO, NCAR, 6b. [Available online at https://www2.cisl.ucar.edu/sites/default/files/govett_6b.pdf]


Sawyer, W., C. Conti, and X. Lapillone, 2011: Porting the ICON non-hydrostatic dynamics and physics to GPUs. Fourth NCAR Multi-Core Workshop, Boulder, CO, NCAR, 19 pp. [Available online at https://www2.cisl.ucar.edu/sites/default/files/2011_09_08_ICON_NH_GPU.pdf]

ABSTRACT

The design and performance of the nonhydrostatic icosahedral model (NIM) global weather prediction model is described. NIM is a dynamical core designed to run on central processing unit (CPU), graphics processing unit (GPU), and Many Integrated Core (MIC) processors. It demonstrates efficient parallel performance and scalability to tens of thousands of compute nodes and has been an effective way to make comparisons between traditional CPU and emerging fine-grain processors. The design of the NIM also serves as a useful guide in the fine-grain parallelization of the finite volume cubed (FV3) model recently chosen by the National Weather Service (NWS) to become its next operational global weather prediction model.

This paper describes the code structure and parallelization of NIM using standards-compliant open multiprocessing (OpenMP) and open accelerator (OpenACC) directives. NIM uses the directives to support a single, performance-portable code that runs on CPU, GPU, and MIC systems. Performance results are compared for five generations of computer chips including the recently released Intel Knights Landing and NVIDIA Pascal chips. Single and multinode performance and scalability is also shown, along with a cost–benefit comparison based on vendor list prices.