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 by 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 dynamics. The German Weather Service (DWD) and Max Planck Institute for Meteorology developed the Icosahedral 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 Architecture (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 parallelization (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 K20x GPUs to one 2011-generation, dual-socket Intel Westmere CPU (Yashiro et al. 2014). Other dynamical cores parallelized for the GPU, including the finite volume cubed (FV3) model (Lin 2004) used in Goddard Earth Observing System Model, version 5 (GEOS-5) (Putnam 2011), and the High-Order Method Modeling Environment (HOMME) (Carpenter et al. 2013), have shown some speedup 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 using 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 as multiple versions must be maintained and optimized. When different-generation hardware is used (e.g., 2010 CPUs vs 2013 GPUs), adjustments should be made to normalize reported speedups. Similarly, when 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 researchers began exploring fine-grain computing. Research 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 comprehensive parallelization for the MIC with National Oceanic and Atmospheric Administration (NOAA)'s Flow-Following Finite-Volume Icosahedral Model (FIM) (Bleck et al. 2015) included dynamics and physics 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 KNC was difficult. This all changed with the release of Intel's Knights Landing (KNL) processor in early 2016. Research groups are now reporting 2-times or 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 codeveloped 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

AFFILIATIONS: Govett, Lee,* and MacDonald*—Global Systems Division, NOAA/Earth System Research Laboratory, Boulder, Colorado; Rosinski, Middlecoff, Henderson,* Wang, and Schramm—Cooperative Institute of Research in the Atmosphere, Colorado State University, Fort Collins, Colorado; Madden* and Duarte*—Cooperative Institute for Research in Environmental Sciences, University of Colorado Boulder, Boulder, Colorado
* **CURRENT AFFILIATIONS:** Henderson, Lee, MacDonald, and Madden—Spire Global, Inc., Boulder, Colorado; Duarte—Cheesecake Laboratories, Florianopolis, Brazil
CORRESPONDING AUTHOR: Mark Govett, mark.w.govett@noaa.gov

The abstract for this article can be found in this issue, following the table of contents.
DOI:10.1175/BAMS-D-15-00278.1

In final form 5 March 2017

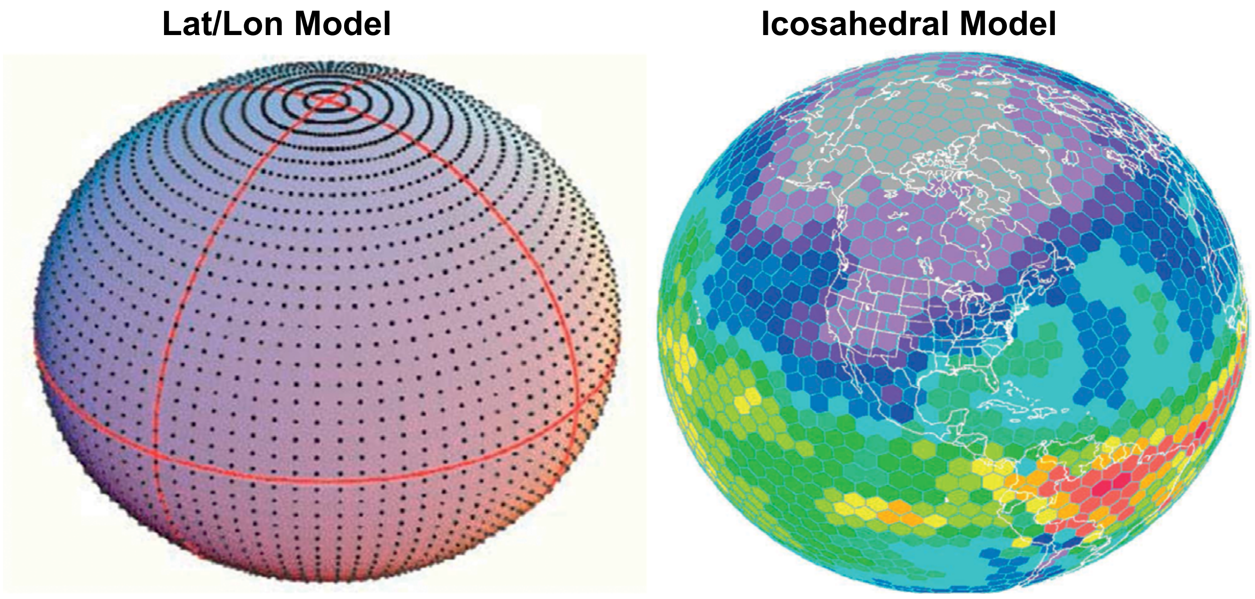

Fig. 1. An illustration contrasting the converging grid points of a lat–lon grid vs the nearly uniform grid spacing of an icosahedral–hexagonal grid.

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) directives for parallelization. Scalable Modeling System (SMS) directives and run-time library support MPI-based distributed-memory parallelism, including domain 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 execution. 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. 2014, 2015, 2016).

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 gradient 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 integrals 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 nearly 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 uniform grid represents the poles without 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 more 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 regions of code called kernels. Loop-level calculations are typically executed in parallel in kernels. The OpenACC programming model designates three levels of parallelism for loop calculations: *gang*, *worker*, and *vector* that are mapped to execution threads and blocks on the GPU. Gang parallelism is for coarse-grain calculations. Worker-level parallelism is fine grain, where each gang will contain one or more workers. Vector 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. The programming model used to express parallelism on MIC hardware is traditional OpenMP threading along with vectorization. User code can be written to offload computationally intensive calculations from the CPU to the MIC (similar to GPU), run in MIC-only mode, or shared between MIC and CPU host.

NIM uses a fully three-dimensional finite-volume discretization scheme designed to improve pressure gradient calculations 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 collocated 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 evaluated as finite-volume integrals around each grid cell. Passive tracers are strictly conserved to the round-off limit of single-precision floating-point operations. NIM governing equations are cast in conservative flux forms with mass flux to transport both momentum and tracer variables.

Computational design. NIM is a FORTRAN code containing a mix of FORTRAN 77 and FORTRAN 90 language constructs. It does not use derived types, pointers, or other constructs that can be challenging for compilers to support or run efficiently.[1] The SMS library used by NIM for coarse-grain parallelism employs the message passing interface (MPI) library to handle domain decomposition, 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.

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 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 a 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 model initialization, interprocess communications, and output. GPU-to-GPU interprocess communications are handled via SMS directives and initiated by the CPU. Since physical parameterizations have not yet been ported to the GPU, data must also be moved between the GPU and CPU every physics time step. This constraint can be removed once the physics is also running on the GPU.

Parallelization of NIM for the MIC was trivial since the code had already been modified to run on the CPU and GPU. As a result, few code changes and optimizations were needed to run efficiently on the MIC processor.

[1] The OpenACC specification only recently added support for derived types; pointer abstractions may limit the ability of compilers to fully analyze and optimize calculations.

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 (×86) 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, F2C-ACC was the primary compiler being used to parallelize and run NIM on NVIDIA GPUs. F2C-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 Embedded 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 F2C-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, 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*, *worker*, or *vector*, to identify 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 calculations are organized in NIM with threading 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 synchronization at the end of the threaded region. These costs are generally higher on the MIC than 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 designated 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, vectorization 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,[2] where it has run on more than 250,000 CPU cores and more than 15,000 GPUs. It has also been run on up to 320 Intel MIC (Xeon Phi) processors at the Texas Advanced Computing Center (TACC).[3] Optimizations targeting Xeon Phi and GPU have also improved CPU performance.

Since NIM has been optimized for the CPU, GPU, and MIC, it is a useful way to make comparisons between chips.[4] Every attempt was made to make fair comparisons between same-generation hardware, using identical source code optimized for all architectures. Given the increasing diversity of hardware solutions, results are shown in terms of device, node, and multinode performance.

[2] 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).

[3] Runs were made on Stampede, an Intel CPU–MIC system supported by the National Science Foundation (NSF).

[4] GPU performance relied on the F2C-ACC compiler. Based on our evaluations, we believe openACC compilers would yield similar results.

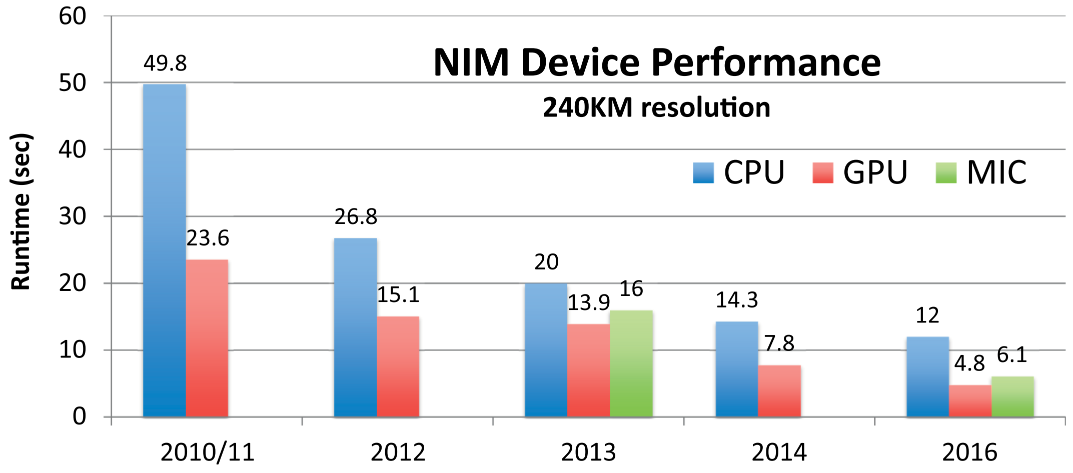

Fig. 2. Run times for the NIM running at 240-km resolution (10,242 horizontal points, 96 vertical levels) for 100 time steps using CPU, GPU, and MIC chips identified in Table 1.

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

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.

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

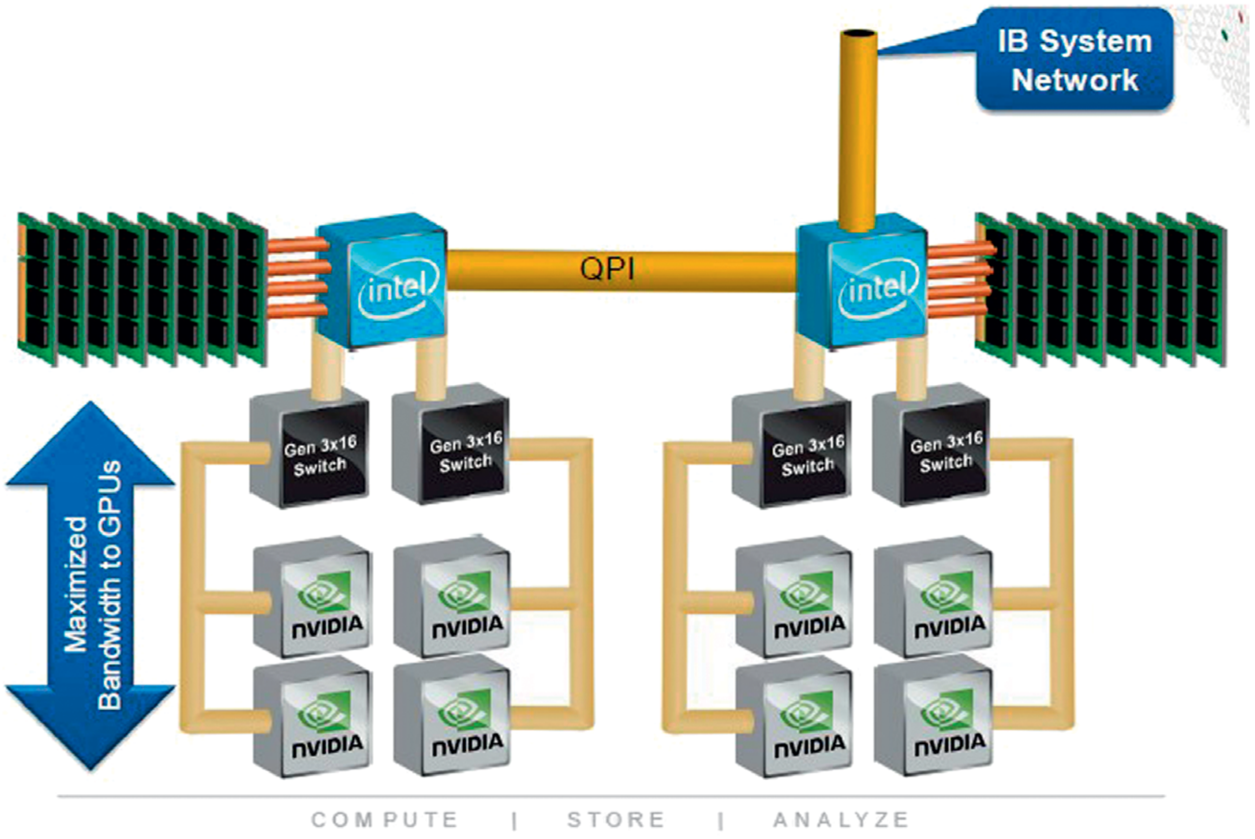

Fig. 3. Illustration of the Cray Storm node architecture containing eight accelerators per node. NVIDIA GPUs are shown, but other PCIe-compatible devices can be used. IB refers to a type of node interconnect called "InfiniBand"; others are also available.

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.

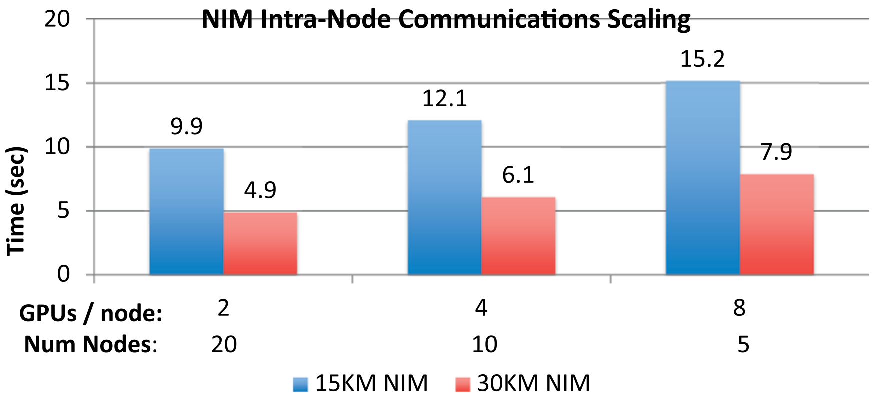

Fig. 4. Communications scaling using 40 Pascal GPUs with 2–8 GPUs per node. Computation times for each of the 15- and 30-km runs are not shown but were consistently 28.6 and 7.5 s, respectively.

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 multinode 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

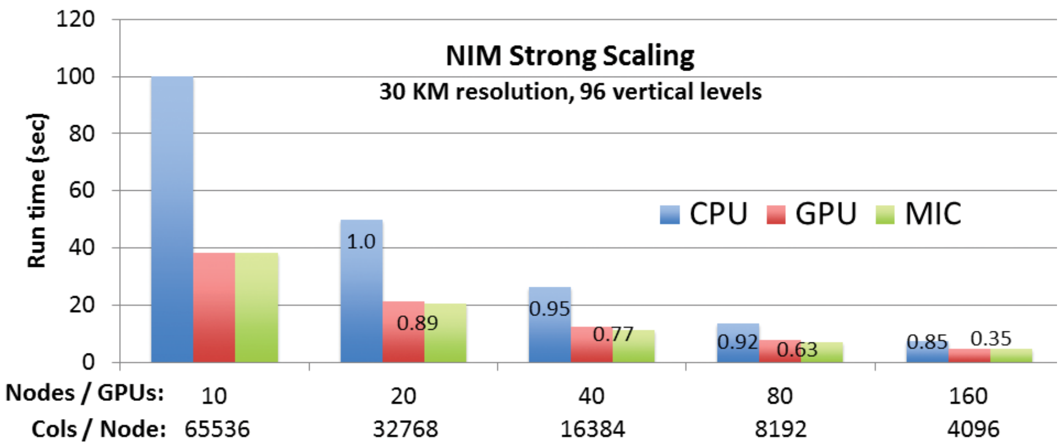

Fig. 5. NIM strong scaling comparison with dual-socket Haswell CPU, NVIDIA Pascal GPU, and Intel KNL (MIC) processors. The horizontal axis gives the number of nodes used for a fixed problem size. The "cols/node" numbers indicate computational workload per node. Speedup efficiency compared to the 10-node CPU and GPU run times appear as numeric values in each performance bar. An MIC baseline run with 10 processors could not be run because of insufficient memory.

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 20,284 columns 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

Table 2. Weak scaling performance for a single node (not shaded) and multiple nodes (shaded) for 100 time steps on K80 GPUs. For a fixed computational workload (20,482 columns), single-node communications time increases from 0.56 to 3.19 as the number of GPUs increase from two to eight. A further increase to 7.23 s is observed when four nodes are used.

GPUs per node	No. of nodes	Model resolution (km)	Columns per GPU	Computation time (s)	Communications time (s)	Total run time
2	1	120	20,482	25.13	0.56	25.71
8	1	60	20,482	25.16	3.19	28.35
2	4	60	20,482	25.22	7.23	33.45

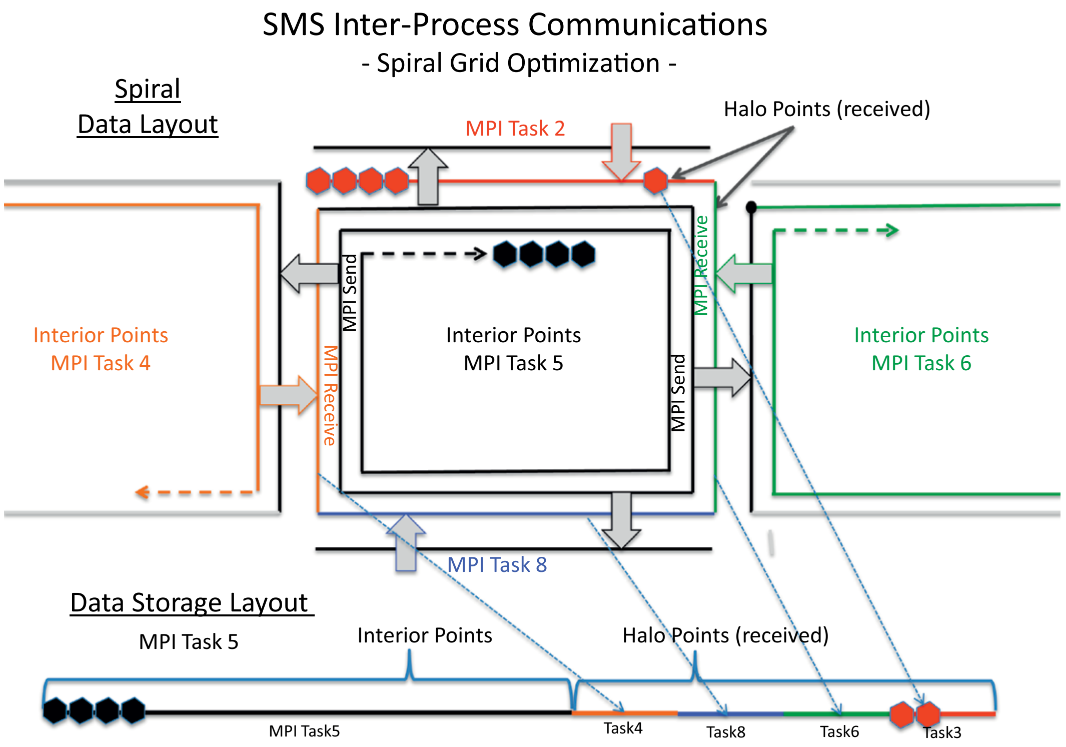

FIG. 6. An illustration of the spiral layout. The upper portion of the figure, titled "spiral data layout," shows a traversal of icosahedral grid cells (hexagons) for MPI tasks 4, 5, and 6. "Data storage layout" illustrates how data are organized to be contiguous in memory. Line color indicates who owns the cells (e.g., task 5 is in black). The orange, red, green, and blue lines in task 5 are halo cells, duplicated in task 5 memory but owned by tasks 4, 2, 6, and 8. Arrows indicate MPI interprocess communications to update these halo cells.

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 workload remains fixed (weak scaling).[5]

Based on list prices in Table 3, a 40-node CPU would cost $260,000. Systems configured with 1–4

[5] Each doubling in horizontal resolution requires 4 times more compute power and a 2-times increase in the number of model time steps. Assuming perfect scaling, a threefold increase in model resolution from 30 to 3.75 km requires 64 times (4^3) more GPUs and an 8-times (2^3) increase in the number of model time steps. Therefore, scaling to 3.75 km is calculated as 8 × 0.20 = 1.6% of real time. Additional increases in compute power and time to solution are expected when physics calculations are included.

NVIDIA K80s per CPU are shown that lower the price of the system from $230,000 to $132,500, respectively. For these tests, 20 NVIDIA K80s were used packaged with two GPUs per K80. No changes in run times were observed for the four CPU–GPU configurations. Systems such as Cray Storm support up to eight GPU per node, which could give additional cost benefit.

TABLE 3. List prices for Intel Haswell CPU, Intel MIC, and NVIDIA K80 GPU processors. The CPU node is based on the cost of a Dell R430 rack-mounted system.

Chip	Part	Cores	Power (W)	OEM Price
Haswell	E5–2690-V3 (2)	24	270	$4,180[a]
NVIDIA K80	K80	4,992	300	$5,000[b]
Intel MIC (KNC)	7120P	61	300	$4,129[c]
Haswell CPU Node	Dell R430	24	—	$6,500[d]

[a] http://ark.intel.com/products/81713/Intel-Xeon-Processor-E5-2690-v3-30M-Cache-2_60-GHz
[b] www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu
[c] http://ark.intel.com/products/75799/Intel-Xeon-Phi-Coprocessor-7120P-16GB-1_238-GHz-61-core
[d] This price quote, from 2 Mar 2015, is for a rack-mounted Dell PowerEdge R430 server.

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 and optimize 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.

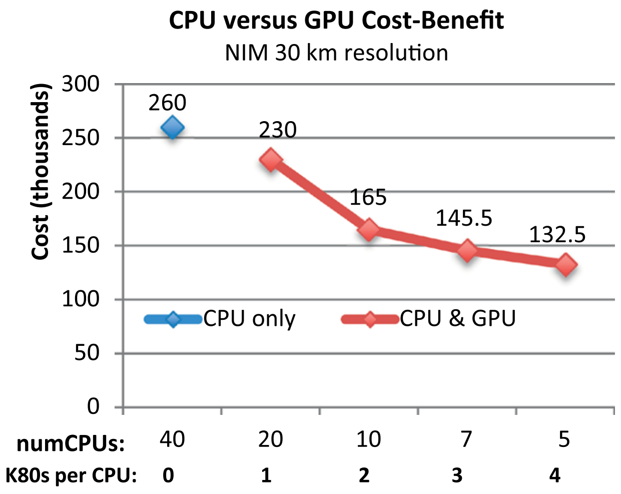

FIG. 7. Cost comparison for CPU-only and CPU–GPU systems needed to run 100 time steps of NIM dynamics in 23 s. Run times do not include model initialization or I/O. Cost estimates are based on list prices for hardware given in Table 3. The CPU-only system used 40 Haswell CPU nodes. Four CPU–GPU configurations were used, where "numCPUs" indicates the total number of CPUs used, and "K80s per CPU" indicates the number of accelerators attached to each node.

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 of 96 levels 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 dynamics 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 benefitting 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. OpenACC compilers continue to mature, benefiting from 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 run-time 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 scientists had not decided which 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.

The 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 naïve 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 CPU 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 cost–benefit 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 underlying hardware. The work reported here represents a successful development effort by a team of domain and computer scientists and software engineers working together during design and development. Scientists wrote the code, computer scientists were responsible for the directive-based parallelization and optimization, and software engineers maintained the software infrastructure capable of supporting development, testing, and running the model on diverse supercomputer systems.

The selection of the FV3 dynamical core to be part of the Next Generation Global Prediction System (NGGPS) run by NWS in 2020 has accelerated efforts to port it to MPFG processors. Experience with the NIM on achieving performance portability will guide these efforts. Evaluation of FV3 performance in 2015 indicated good scaling to 130,000 CPU cores (Michalakes et al. 2015). While these results indicate sufficient parallelism is available, significant work is expected to adapt FV3 to run efficiently on GPU and MIC processors (Govett and Rosinski 2016).

In the next decade, HPC is expected to become increasingly fine grained, with systems containing potentially hundreds of millions of processing cores. To take advantage of these systems, new weather prediction models will need to be codeveloped by scientific and computational teams to incorporate parallelism in model design, code structure, algorithms, and underlying physical processes.

ACKNOWLEDGMENTS. Thanks to technical teams at Intel, Cray, PGI, and NVIDIA who were responsible for fixing bugs and providing access to the latest hardware and compilers. Thanks also to the staff at ORNL and TACC for providing system resources and helping to resolve system issues. This work was also supported in part by the Disaster Relief Appropriations Act of 2013 and the NOAA HPCC program.

REFERENCES

Arakawa, A., and V. R. Lamb, 1977: Computational design of the basic dynamical processes of the UCLA general circulation model. *General Circulation Models of the Atmosphere*, J. Chang, Ed., Vol. 17, *Methods in Computational Physics: Advances in Research and Applications*, Academic Press, 173–265.

Bleck, R., and Coauthors, 2015: A vertically flow-following icosahedral grid model for medium-range and seasonal prediction. Part I: Model description. *Mon. Wea. Rev.*, **143**, 2386–2403, doi:10.1175/MWR-D-14-00300.1.

Carpenter, I., and Coauthors, 2013: Progress towards accelerating HOMME on hybrid multi-core systems. *Int. J. High Perform. Comput. Appl.*, **27**, 335–347, doi:10.1177/1094342012462751.

Cirrascale, 2015: Scaling GPU compute performance. Cirrascale Rep., 11 pp. [Available online at www.cirrascale.com/documents/whitepapers/Cirrascale_ScalingGPUCompute_WP_M987_REVA.pdf.]

Ellis, S., 2014: Exploring the PCIe bus routes. CirraScale. [Available online at www.cirrascale.com/blog/index.php/exploring-the-pcie-bus-routes/.]

Fuhrer, O., C. Osuna, X. Lapillone, T. Gysi, B. Cumming, M. Bianco, A. Arteaga, and T. Schulthess, 2014: Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. *Supercomput. Front. Innovations*, **1**, 45–62, doi:10.14529/jsfi140103.

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.]

——, and J. Rosinski, 2016: Evaluation of the FV3 dynamical core. NGGPS Supplementary Rep., 10 pp. [Available online at www.esrl.noaa.gov/gsd/ato/FV3_Analysis-final.pdf.]

——, L. Hart, T. Henderson, and D. Schaffer, 2003: The scalable modeling system: Directive-based code parallelization for distributed and shared memory computers. *Parallel Comput.*, **29**, 995–1020, doi:10.1016/S0167-8191(03)00084-X.

——, J. Middlecoff, and T. Henderson, 2010: Running the NIM next-generation weather model on GPUs. *10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, Melbourne, Australia, Institute of Electrical and Electronics Engineers/Association for Computing Machinery, 792–796, doi:10.1109/CCGRID.2010.106.

——, ——, and ——, 2014: Directive-based parallelization of the NIM weather model for GPUs. *First Workshop on Accelerator Programming Using Directives*, New Orleans, LA, Institute of Electrical and Electronics Engineers, 55–61, doi:10.1109/WACCPD.2014.9.

——, T. Henderson, J. Rosinski, J. Middlecoff, and P. Madden, 2015: Parallelization and performance of the NIM for CPU, GPU and MIC. *First Symp. on High Performance Computing for Weather, Water, and Climate*, Phoenix, AZ, Amer. Meteor. Soc., 1.3. [Available online at https://ams.confex.com/ams/95Annual/webprogram/Paper262515.html.]

——, ——, J. Middlecoff, and J. Rosinski, 2016, A cost benefit analysis of CPU, GPU and MIC chips using NIM performance as a guide. *Second Symp. on High Performance Computing for Weather, Water, and Climate,* New Orleans, LA, Amer. Meteor. Soc., 1.3. [Available online at https://ams.confex.com/ams/96Annual/webprogram/Paper286165.html.]

Henderson, T., M. Govett, and J. Middlecoff, 2011: Applying Fortran GPU compilers to numerical weather prediction. *2011 Symp. on Application Accelerators in High Performance Computing*, Knoxville, TN, Institute of Electrical and Electronics Engineers, 34–41, doi:10.1109/SAAHPC.2011.9.

——, J. Michalakes, I. Gokhale, and A. Jha, 2015: Optimizing numerical weather prediction. *Multicore and Many-Core Programming Approaches*, J. Reinders and J. Jeffers, Eds., Vol. 2, *High Performance Parallelism Pearls*, Morgan Kaufmann, 7–23, doi:10.1016/B978-0-12-803819-2.00016-1.

Kim, Y., 2013: Performance tuning techniques for GPU and MIC. *Third NCAR Multi-Core Workshop*, Boulder, CO, NCAR, 1. [Available online at https://www2.cisl.ucar.edu/sites/default/files/youngsung_1_2013.pdf.]

Lapillonne, X, and O. Fuhrer, 2014: Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science. *Parallel Process. Lett.*, **24**, 1450003, doi:10.1142/S0129626414500030.

Lee, J.-L., and A. E. MacDonald, 2009: A finite-volume icosahedral shallow water model on local coordinate. *Mon. Wea. Rev.*, **137**, 1422–1437, doi:10.1175/2008MWR2639.1.

——, R. Bleck, and A. E. MacDonald, 2010: A multistep flux-corrected transport scheme. *J. Comput. Phys.*, **229**, 9284–9298, doi:10.1016/j.jcp.2010.08.039.

Lin, S.-J., 2004: A "vertically Lagrangian" finite-volume dynamical core for global models. *Mon. Wea. Rev.*, **132**, 2293–2307, doi:10.1175/1520-0493(2004)132<2293:AVLFDC>2.0.CO;2.

MacDonald, A. E., J. Middlecoff, T. Henderson, and J. Lee, 2011: A general method for modeling on irregular grids. *Int. J. High Perform. Comput. Appl.*, **25**, 392–403, doi:10.1177/1094342010385019.

Michalakes, J., M. Govett, R. Benson, T. Black, H. Juang, A. Reinecke, and B. Skamarock, 2015: NGGPS level-1 benchmarks and software evaluation. Advanced Computing Evaluation Committee Rep., 22 pp. [Available online at www.nws.noaa.gov/ost/nggps/DycoreTestingFiles/AVEC%20Level%201%20Benchmarking%20Report%2008%2020150602.pdf.]

——, M. Iacono, and E. Jessup, 2016: Optimizing weather model radiative transfer physics for Intel's Many Integrated Core (MIC) architecture. *Parallel Process. Lett.*, **26**, 1650019, doi:10.1142/S0129626416500195.

Middlecoff, J., 2015: Optimization of MPI message passing in a multi-core NWP dynamical core running on NVIDIA GPUs. *Fifth NCAR Multi-Core Workshop*, Boulder, CO, NCAR, 4. [Available online at https://www2.cisl.ucar.edu/sites/default/files/Abstract_Middlecoff.pdf.]

Nguyen, H. V., C. Kerr, and Z. Liang, 2013: Performance of the cubed-sphere atmospheric dynamical core on Intel Xeon and Xeon Phi architectures. *Third NCAR Multi-Core Workshop,* Boulder, CO, NCAR, 3a. [Available online at https://www2.cisl.ucar.edu/sites/default/files/vu_3a.pdf.]

NVIDIA, 2015: CUDA C programming guide. NVIDIA. [Available online at http://docs.nvidia.com/cuda/cuda-c-programming-guide/.]

Putnam, B., 2011: Graphics processing unit (GPU) acceleration of the Goddard Earth Observing System atmospheric model. NASA Tech. Rep., 15 pp.

Rosinski, J., 2015: Porting and optimizing NCEP's GFS physics package for unstructured grids on Intel Xeon and Xeon Phi. *Fifth NCAR Multi-Core Workshop*, Boulder, CO, NCAR, 4. [Available online at https://www2.cisl.ucar.edu/sites/default/files/Rosinski_Slides.pdf.]

Sadourny, R., A. Arakawa, and Y. Mintz, 1968: Integration of non-divergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere. *Mon. Wea. Rev.*, **96**, 351–356, doi:10.1175/1520-0493(1968)096<0351:IOTNBV>2.0.CO;2.

Sawyer, W., C. Conti, and X.Lapillonne, 2011: Porting the ICON non-hydrostatic dynamics and physics to GPUs. *First 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.]

——, G. Zaengl, and L. Linardakis, 2014: Towards a multi-node OpenACC implementation of the ICON model. *European Geosciences Union General Assembly 2014*, Vienna, Austria, European Geosciences Union, ESSI2.1. [Available online at http://meetingorganizer.copernicus.org/EGU2014/EGU2014-15276.pdf.]

Strohmaier, E., J. Dongarra, H. Simon, and M. Meuer, 2016: Top 500 supercomputers. Top500, accessed 22 January 2017. [Available online at www.top500.org/lists/2016/11/.]

Wang, N., and J. Lee, 2011: Geometric properties of the icosahedral-hexagonal grid on the two-sphere. *SIAM J. Sci. Comput.*, **33**, 2536–2559, doi:10.1137/090761355.

Williamson, D., 1971: A comparison of first- and second-order difference approximations over a spherical geodesic grid. *J. Comput. Phys.*, **7**, 301–309, doi:10.1016/0021-9991(71)90091-X.

Yashiro, H., A. Naruse, R. Yoshida, and H. Tomita, 2014: A global atmosphere simulation on a GPU supercomputer using OpenACC: Dry dynamical cores tests. *TSUBAME ESJ*, Vol. 12, Tokyo Institute of Technology Global Scientific Information and Computing Center, Tokyo, Japan, 8–12. [Available online at www.gsic.titech.ac.jp/sites/default/files/TSUBAME_ESJ_12en_0.pdf.]

AMS titles now available as eBooks at springer.com

AMS BOOKS

RESEARCH APPLICATIONS HISTORY

www.ametsoc.org/amsbookstore

Springer

Scan to see
AMS eBook titles
at springer.com

Springer

HOME | MY SPRINGER | SUBJECTS | SERVICES | IMPRINTS & PUBLISHERS | ABOUT US

» New User
LOGIN

» American Meteorological Society

Home > Societies & Publishing Partners > American Meteorological Society

Search... GO
Advanced Search

AMS BOOKS
RESEARCH APPLICATIONS HISTORY

American Meteorological Society

Promoting the development and dissemination of information and education on the atmospheric and related sciences.

American Meteorological Society eBooks

AMS Books grew out of a monograph publishing program related to the peer-reviewed journals published by the AMS and now encompasses a series of meteorological and historical monographs that reaches back five decades (formerly out-of-print volumes of which have been brought back on SpringerLink as eBooks) as well as a growing list of academic, policy, and general-interest books related to the atmospheric sciences.

Sort listing by: Relevance [✓ Publication date] Title Author Copyright Year
Export to CSV

AMERICAN METEOROLOGICAL SOCIETY EBOOKS
› Show all 11 results

Eloquent Science
A Practical Guide to Becoming a Better Writer, Speaker, and Atmospheric Scientist
Schultz, David
2009
Price from $45.00
Available Formats:

AMERICAN METEOROLOGICAL SOCIETY

AMERICAN METEOROLOGICAL SOCIETY 1919

» About the American Meteorological Societ...
» Order print copies of AMS Books
» Booksellers seeking print copies of AMS ...
» For AMS Journals Online

RESOURCES

Stay Informed
Receive notification of new releases from AMS Books

Submit a Book Proposal to the AMS
Download our book proposal guidelines

Become an AMS Member
Learn about the benefits of membership

CONNECT WITH THE AMERICAN METEOROLOGICAL SOCIETY

Like AMS on Facebook
Follow AMS on Twitter

AMERICAN METEOROLOGICAL SOCIETY