

Improving NASA's Multiscale Modeling Framework for Tropical Cyclone Climate Study

One of the current challenges in tropical cyclone research is how to improve our understanding of TCs' interannual variability as well as climate change's impact. Modern advances in global modeling, visualization, and supercomputing technologies at NASA show potential, but scalability is an issue. Recent improvements to the multiscale modeling framework make long-term TC-resolving simulations much more feasible.

Studies of tropical cyclone interannual variability and the impact of climate change on TCs have received increasing attention, particularly as during the past 10 years statistics indicate that TCs are the deadliest weather events in the US (www.nws.noaa.gov/os/hazstats.shtml). Improving short- and long-term hurricane forecasts is imperative. Depending on their location, TCs can be called other names, such as hurricanes (in the Atlantic region), typhoons (in the West Pacific region), tropical storms, cyclonic storms, and tropical depressions.

TC dynamics involve multiscale processes. To accurately simulate a TC's evolution on a medium, or *meso*, scale, an ideal solution would be to improve the model to realistically capture both the downscaling processes associated with large-scale

flows, such as Madden-Julian oscillations or tropical waves, and the upscaling processes associated with the feedback from small-scale flows, such as convection or precipitation. However, it has been challenging to accurately simulate TC activities with traditional numerical models because of artificial scale separations. For example, Earth atmospheric-modeling activities are conventionally divided into three major categories: *global*, or large, scale; *mesoscale*, and *cloud*, or micro, scale. As Figure 1 shows, typical resolutions for the global, mesoscale (or regional), and cloud models are on the order of 100 km, 10 km, and 1 km, respectively.

Due to limited access to computing resources, researchers have had to conduct TC climate studies primarily with coarse-resolution general circulation models (GCMs)¹ and partially with fine-resolution, regional mesoscale models (MMs). The former have the advantage of simulating the impact of global-scale flows on TC activities but might not accurately simulate meso- and cloud-scale flows. In contrast, the latter make it possible to simulate realistic TC intensity and structure with fine grid spacing, but this approach has problems capturing the accurate impact of large spatial- and temporal-scale flows. Furthermore, the resolutions used in GCMs and MMs are still too coarse to resolve small-scale

1521-9615/13/\$31.00 © 2013 IEEE
COPUBLISHED BY THE IEEE CS AND THE AIP

BO-WEN SHEN

University of Maryland, College Park, and NASA Goddard Space Flight Center

BRON NELSON AND SAMSON CHEUNG

NASA Ames Research Center

WEI-KUO TAO

NASA Goddard Space Flight Center

PREDICTABILITY OF TROPICAL CYCLONE GENESIS

By examining the role of different tropical waves in tropical cyclogenesis with observations, William Frank and Paul Roundy¹ found a strong relationship between tropical cyclone (TC) formation and enhanced activity in equatorial Rossby waves, mixed Rossby gravity waves,² tropical depression (TD)-type disturbances (or easterly waves), and Madden-Julian oscillations (MJOs).³ With a 45- to 60-day time scale, eastward-propagating MJOs, which are typically characterized by deep convection originating over the Indian Ocean, have one of the most prominent large-scale features of the tropical general circulation. These large-scale tropical systems could provide determinism with regards to TC genesis timing and location. Thus, it becomes possible to extend the lead time for TC genesis prediction and thus increase our confidence in the model's ability to simulate TC climates as long as the model can improve the simulations of the precursor and its modulation on TC activities. More information on modeling the association between TC genesis and these large-scale tropical systems appears elsewhere.⁴⁻⁷

References

1. W.M. Frank and P.E. Roundy, "The Role of Tropical Waves in Tropical Cyclogenesis," *Monthly Weather Review*, vol. 134, no. 9, 2006, pp. 2397–2417.
2. T. Matsuno, "Quasi-Geostrophic Motions in the Equatorial Area," *J. Meteorological Soc. Japan*, vol. 44, no. 1, 1966, pp. 25–43.
3. R.A. Madden and P.R. Julian, "Detection of a 40–50 Day Oscillation in the Zonal Wind in the Tropical Pacific," *J. Atmospheric Sciences*, vol 28, no. 5, 1971, pp. 702–708.
4. B.-W. Shen, W.-K. Tao, and B. Green, "Coupling Advanced Modeling and Visualization to Improve High-Impact Tropical Weather Prediction (CAMVis)," *Computing in Science & Eng.*, vol. 13, no. 5, 2011, pp. 56–67.
5. B.-W. Shen et al., "Genesis of Twin Tropical Cyclones as Revealed by a Global Mesoscale Model: The Role of Mixed Rossby Gravity Waves," *J. Geophysical Research*, vol. 117, no. D13, 2012; doi:10.1029/2012JD017450.
6. B.-W. Shen et al., "Advanced Visualizations of Scale Interactions of Tropical Cyclone Formation and Tropical Waves," *Computing in Science & Eng.*, vol. 15, no. 2, 2013, pp. 47–52.
7. B.-W. Shen et al., "Genesis of Hurricane Sandy (2012) Simulated with a Global Mesoscale Model," *Geophysical Research Letters*, vol. 40, 2013, pp. 1–7; doi:10.1002/grl.50934.

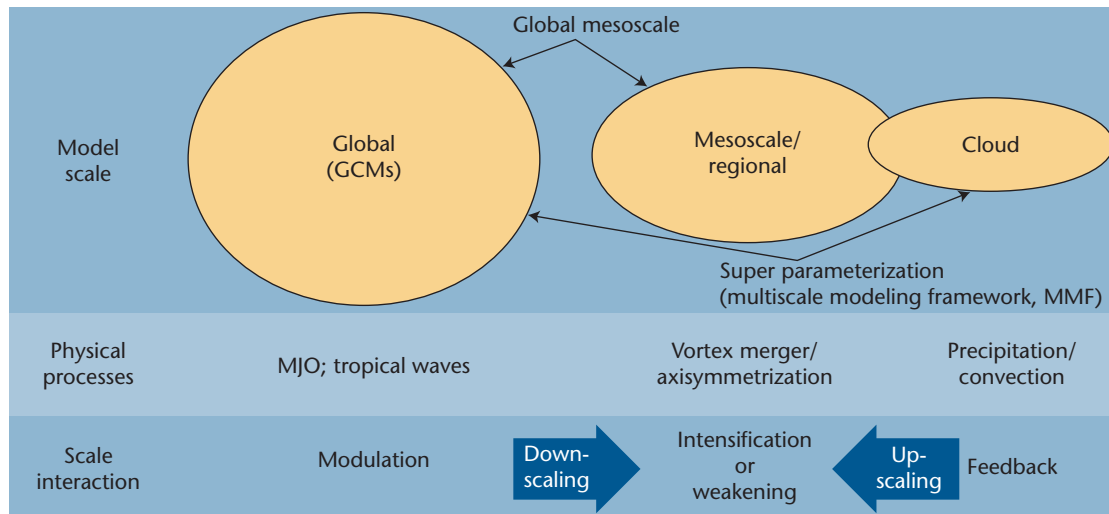


Figure 1. The three major categories of Earth atmospheric modeling (from left to right): global (large) scale, meso (medium) scale, and cloud (micro) scale. Typical resolutions for these models are roughly 100 km, 10 km, and 1 km, respectively.

convective motion in TC climate studies, forcing researchers to use cumulus parameterizations (CPs) to emulate the effects of unresolved subgrid-scale cloud motion. Because the development of CPs has been slow, their performance is a major limiting factor in TC simulations. To overcome the issues with CPs, a recent trend is to take full advantage of modern supercomputing power to

explicitly resolve the impact of clouds in a global environment. Two approaches are to deploy high-resolution global models²⁻⁷ or to couple a coarse-resolution global model with massive copies of cloud models. The latter approach is known as superparameterization, or the multiscale modeling framework (MMF), and it's the one we focus on here.⁸⁻¹⁰ Recently, we successfully integrated

both models with visualizations into the coupled advanced multiscale modeling and visualization system (CAMVis) on NASA's Pleiades supercomputer,⁴⁻⁷ which shows promise for short-term and extended-range TC simulations.

In the MMF, you replace the conventional CP at each GCM grid point with a copy of a cloud-resolving model (CRM) to accurately represent non-hydrostatic, cloud-scale convection and its interaction with environmental flows. Thus, the MMF has the advantages of both the global-scale processes of a GCM and the sophisticated microphysical processes of a CRM, and can be viewed as an alternative to a global CRM. However, this approach poses great challenges in terms of computing resources, data storage, and data analyses for multidecadal simulations of global TC activities. These challenges include, but are not limited to, running more than 10,000 instantiations of the CRM with great parallel performance, increasing the GCM's resolution to capture realistic TC structure, efficiently archiving massive volumes of data, and effectively conducting analyses to discover the predictive relationship between meteorological (short-term) and climatological (long-term) events. To address the first two issues, we propose a revised MMF coupling approach to improve the model's scalability, enabling higher resolutions in both the GCM and CRM, reducing the time to finish TC climate simulations.

Throughout this article, we adopted the following conventions: the name of each routine ends with parentheses (); the name of each variable is in italics; the term *process* is a running instance of model codes; and the terms *task* and *process* are used as follows: the former emphasizes the work being done, and the latter emphasizes the worker doing it (for example, task A is performed using process A).

NASA's MMF

The NASA Goddard MMF^{9,10} is based on the NASA Goddard finite-volume GCM (fvGCM) and the Goddard cumulus ensemble (GCE) model. While the high-resolution fvGCM has shown remarkable capabilities in simulating large-scale flows, and thus hurricane tracks,^{2-7,11} the GCE is well known for its superior performance in representing small, cloud-scale motions and has been used to produce more than 100 referred journal papers.^{12,13} In the MMF, the fvGCM runs at a coarse ($2^\circ \times 2.5^\circ$) resolution, and one copy of the GCE runs within each of the fvGCM grids.¹⁰

Researchers still widely use the $2^\circ \times 2.5^\circ$ resolution in climate simulations with conventional climate models and recent MMFs because it's computationally affordable. This resolution has grid points of (91, 144) in the (y, x) directions, giving a total of 91×144 (13,104) horizontal cells. Thus, 13,104 GCEs are "embedded" in the fvGCM to allow explicit simulation of cloud processes in a global environment. Currently, only averaged thermodynamic fields such as temperature and water vapor in the GCEs are fed back to the fvGCM, in a process called *thermodynamic feedback*. The timestep for the individual 2D GCE is 10 seconds, and the fvGCM-GCE coupling interval is one hour at this resolution. Under this configuration, 99 percent or more of the total wall time for running the MMF is spent on the GCEs. Thus, wall time could be significantly reduced by efficiently distributing the large number of GCEs over a massive number of processors on a supercomputer.

Let's look more closely at the computational parts of the GCE and fvGCM, before discussing a revised strategy for coupling the latter with massive copies of the GCE to improve scalability.

The GCE Model

Typical MMF model runtime configurations are 64 grid points in the x direction, with a grid spacing of 4 km; 32 vertical stretched levels; cyclic lateral boundary conditions; and a time step of 10 seconds. The GCE itself has been implemented with a 2D domain decomposition using message-passing interface version 1 (MPI-1) with good parallel efficiency.¹⁴ Thus, an ideal solution for the course-grain parallelism implemented in the MMF is to run more copies of GCEs with higher CPU counts in parallel, while still keeping the option of taking advantage of the fine-grain parallelism inside the GCE.

The fvGCM

Resulting from a development effort of more than 15 years, the fvGCM is a unified numerical weather prediction (NWP) and climate model that can run on daily, monthly, decadal, or century time scales.^{11,15} The 1990s model was originally designed for climate studies at a coarse resolution of approximately 2×2.5 degrees, and its resolution was increased to 1 degree in 2000 and $\frac{1}{2}$ degree in 2002 for NWP. Since 2005, the high-resolution—that is, $\frac{1}{8}$ and $\frac{1}{12}$ degree—fvGCM has been deployed on NASA's Columbia supercomputer, showing remarkable TC forecasts.²

The fvGCM's parallelization was carefully designed to achieve efficiency, parallel scalability,

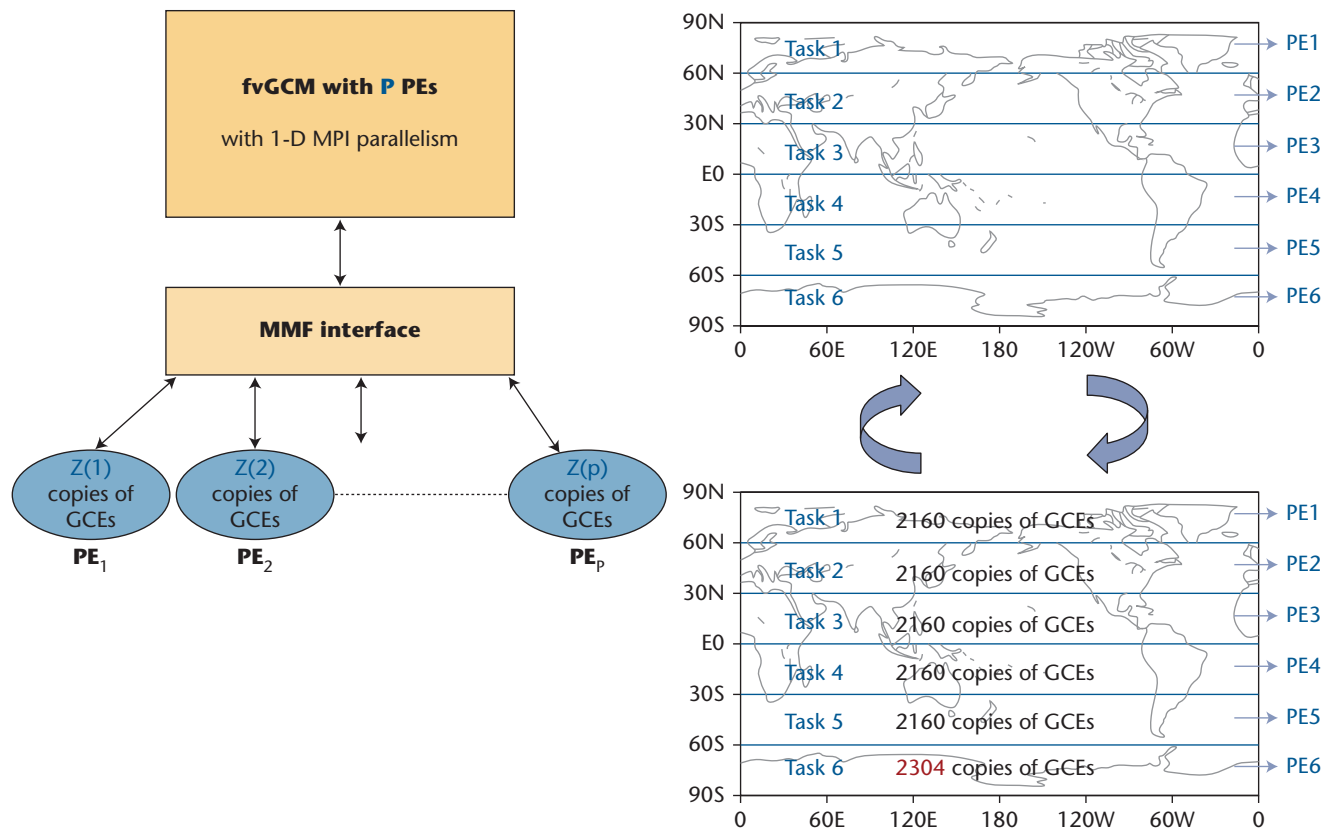


Figure 2. Parallelism in the finite-volume general circulation model (fvGCM) and the multiscale modeling framework (MMF) version 1.0. The fvGCM has the message-passing interface (MPI) parallel implementation with a 1D domain decomposition along the y direction. The approach of embedding one copy of Goddard cumulus ensemble (GCE) into each of the fvGCM's grids inherits the parallelism and thus limited scalability of the fvGCM. The right panel shows the distribution of tasks over six processing elements (PEs). Task J (here, $J = 1 \sim 5$) performs 2,160 copies of GCEs, while task 6 has 2,304 copies of GCEs: $Z(J) = 2,160$, $J = 1 \sim 5$; $Z(6) = 2,304$; $\sum_{J=1}^P Z(J) = 13,104$, with $P = 6$ in this case.

flexibility, and portability. Its implementation had distributed- and shared-memory, two-level parallelism, including a coarse-grain parallelism with MPI and fine-grain parallelism with OpenMP (throughout this article, we refer to MPI as any one of MPI-1, MPI-2, MLP, or SHMEM communication paradigms).¹⁶ However, the latter isn't applicable for current MMF runs. Because MPI parallelism is applied to a 1D decomposition over latitude in the fvGCM, and each MPI task needs at least three grid points in latitude for parallel efficiency, MPI parallelism is very limited for small computational grids (see Figure 2). Assume NY is the number of grid points in the y direction. In general, NY is equal to $(180^\circ/DY + 1)$, where DY is an increment in latitude that can be 0.08° , 0.125° , 0.25° , 0.5° , 1° , or 2° . For the 2×2.5 degree grids where $NY = 91$ (with $DY = 2$), the maximum of MPI tasks is only 30 ($\sim 91/3$). In addition, because NY generally isn't divisible by the number of selected processes (P), the domain decomposition is nonuniform, which leads to load unbalancing

because some MPI processes receive more latitudes than others.

Revised Parallelism in the MMF

We discuss the revised parallelism by first introducing the metaglobal GCE (mgGCE) and then presenting the details on parallel implementation, including the creation of two process groups—process mapping and load balancing—and the dynamical distribution of mgGCE inputs over mgGCE processes.

The Metaglobal GCE

In MMF version 1, each of 13,104 GCEs is embedded into an fvGCM grid cell. Although the implementation of this version is straightforward, by adopting the fvGCM's parallel framework, it inherits both the fvGCM framework's advantages and disadvantages. One of the disadvantages is that this approach limits the MMF's parallel scalability—that is, it can only decompose the whole domains into small-number subdomains and each

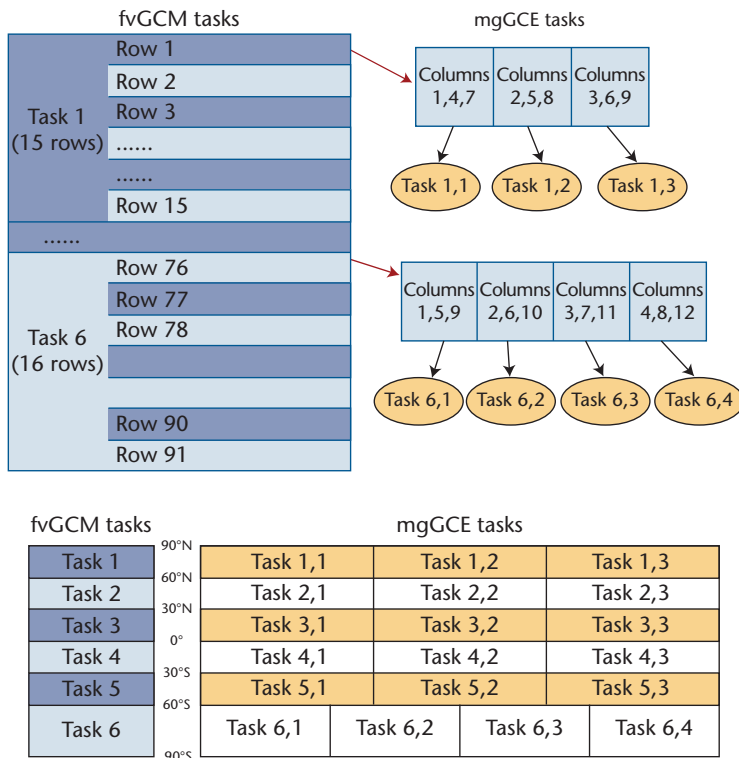


Figure 3. A conceptual diagram of the revised parallelism implementation using six fvGCM tasks and 19 megaglobal GCE (mgGCE) tasks. In the original 1D decomposition, each fvGCM process (top left panel) gets three or more latitudes (or rows) and computes the mgGCEs for each longitude (or column) sequentially within a row. The new version adds a second level of parallelism, decomposing a single row into individual columns (top right panel). Each fvGCM process has an associated set of mgGCE processes that compute the GCEs for a row in parallel. More mgGCE processes are assigned to the fvGCM process with more rows (that is, more copies of the GCEs). The 13,104 GCE copies are distributed over these mgGCE processes (and the fvGCM processes as needed), achieving an effective 2D domain decomposition shown in the bottom panel. This revised parallelism can improve the MMF’s scalability by allowing more copies of GCEs running in parallel, thus reducing wall time significantly.

requires many copies of GCEs to run within one processing element (PE). For example, 2,160 copies of the GCE run in series in one PE when we use only six PEs (see Figure 2). Let’s assume we use the maximum number PEs (30), and each PE performs one copy of the GCE at a time and needs to run 432 copies of the GCEs sequentially. Therefore, making more copies of the GCEs to run in parallel with more PEs is the key to reducing the wall-clock time.

From a computational perspective, the concept of embedding GCEs into the fvGCM restricts the MMF’s view of the parallelism—namely, it can only inherit it from the fvGCM. Because a GCE uses a periodic, lateral boundary condition, each embedded GCE’s execution is independent of the other GCEs during a time step of the

fvGCM model, and thus the GCE can run in a separate MPI task. Accordingly, we propose a new coupling approach to improve the MMF’s parallel scalability. Conceptually, we refer to the 13,104 (144 × 91) copies of GCEs as a supercomponent called a metaglobal GCE (mgGCE) in a meta grid-point system. To facilitate discussion, we assume this grid system is the same as the latitude-longitude grid structure in the fvGCM, although it isn’t necessarily tied to any specific grid system.

With this concept in mind, each of the two individual components (the fvGCM and mgGCE) in the MMF could have its own domain decompositions. Note that because each GCE uses cyclic, lateral boundary conditions, there’s no data communication between any two GCEs. In other words, the mgGCE has no ghost region, which is defined as the edge points of the computing subdomain in one processor that store the data belonging to adjacent processors. Thus, a 2D domain decomposition in the mgGCE can significantly reduce the runtime for massive copies of the GCE, which can greatly improve MMF scalability because most of the wall time is spent on these GCEs.

To couple the fvGCM and mgGCE, the parallelism is implemented to do the following: create two groups of processes with the MPI intercommunication, one group with P fvGCM processes and the other with Q mgGCE processes; build a sophisticated static mapping between the P fvGCM and Q mgGCE processes; and dynamically distribute input values to mgGCE processes from the fvGCM processes that can execute 13,104 copies of the GCEs and handle load balancing. This implementation leads to an effective 2D domain decomposition in the mgGCE, while the original 1D domain decomposition remains in the fvGCM. Simply speaking, the first-level parallelism decomposes latitudes in the fvGCM, and the second-level parallelism decomposes longitudes in the mgGCE. Figure 3 offers a schematic diagram of these domain decompositions; we discuss the technical details in the next section.

Parallel Implementation

We implemented fvGCM parallelism with MPI to allow point-to-point, collective communication among processes in the same group. This kind of “conventional” communication, which also appears in many existing MPI single-program multiple-data codes, is called intracommunication. In contrast, intercommunication occurs among processes in local and remote groups, where a local group is one within which a process initiates an

intercommunication operation—the local group is the sender (or receiver) in a send (or receive) call. A remote group is one that contains the target process, which is the receiver (or sender) in a send (or receive) call. Note that these two groups don't overlap: when the target process needs to be addressed, the local process uses an (intercommunicator, rank) pair with the rank relative to the remote group.

Because of its unique features, we used intercommunication in the revised parallelism implementation to couple the fvGCM and the mgGCE. The main program's calling tree, fvGCM.F, contains the following steps to finish an MMF run:

1. A master process calls the `mmf_gce2d_task_init()` to create two groups of processes and intercommunicators. One group has P fvGCM processes and the other has Q mgGCE processes. Inside this subroutine, the mgGCE processes start waiting for inputs to perform GCE calculations and interact with the fvGCM processes to receive their inputs in step 5; all fvGCM processes return back to the main program to continue the execution.
2. All fvGCM processes call `mp_init()` and `y_decomp()` to perform a 1D domain decomposition.
3. Each of the fvGCM processes calls the `mmf_gce2d_task_assignment()` to associate itself with a subset of $(T(\mathcal{J}))$ mgGCE processes. Here, \mathcal{J} is the fvGCM process's rank, and $T(\mathcal{J})$ is proportional to the latitudes assigned to the fvGCM process, aimed at achieving load balancing.
4. All fvGCM processes call `mmf_init()` to initialize the MMF run.
5. All P fvGCM processes invoke the `mmf_run()`, where the `mmf_invoke_gce2d()` is called, to interact with Q mgGCE processes that run the `gce2d_task_main()` to finish 144×91 gce2d runs.
6. The fvGCM processes sequentially call `mmf_finalize()` and `mmf_gce2d_task_finalize()` to finalize the fvGCM tasks and mgGCE tasks, respectively.

Let's look more closely at steps 1, 3, and 5.

Creation of two process groups and MPI intercommunicators. Figure 4 displays major functions in the subroutine `mmf_gce2d_task_init()` that create both two groups of processes by calling MPI subroutine `mpi_comm_split()` and the intercommunicators among these two groups with the `mpi_intercomm_create()`, which include

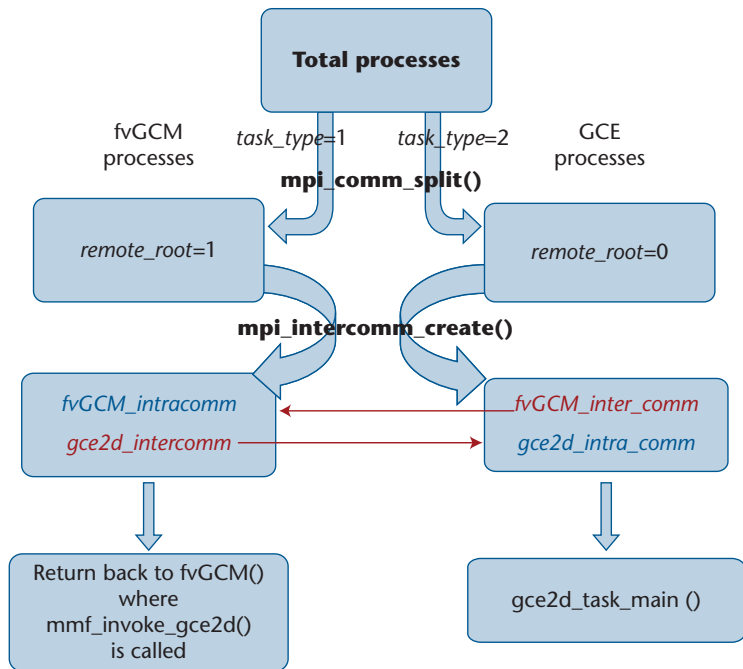


Figure 4. Creation of two groups of processes and their intracommunicators and intercommunicators in the subroutine `mmf_gce2d_task_init()`. The fvGCM group appears on the left-hand side, whereas the mgGCE group is on the right. `gce2d_intercomm` and `fvGCM_inter_comm` (in red) are intercommunicators, and `gce2d_intra_comm` and `fvGCM_intracomm` (in blue) are intracommunicators.

`gce2d_intercomm` with the fvGCM and `fvGCM_inter_comm` with the mgGCE processes as local processes. While mgGCE processes enter a loop of `gce2d_task_main()` and start waiting for inputs to perform GCE calculations, all fvGCM processes return back to the main program where `mmf_invoke_gce2d()` is called to send data to the mgGCE processes; Lists 1 and 2 (Figures 5 and 6) display these two subroutines' major functionalities, which are discussed later.

The mgGCE processes are homogenous and undifferentiated. In fact, they're designed to be stateless, meaning they don't need to know anything about the fvGCM side of things. They don't know what time step it is, who might send them input, or anything about load balancing or task assignments. They simply receive a block of input from the fvGCM processes, use that input to call the `gce2d()` routine, and then return a block of output back to whomever sent them the input. Everything they need to know is either constant or provided within the block of input.

Process mapping and load balancing. For a given *total_number_of_processes* ($P + Q$) and *total_number_of_mgGCE_processes* (Q) at runtime, the mapping between P fvGCM and Q mgGCE processes

```

1. mainLoop: do
2. call mpi_recv(inputValues, , , MPI_ANY_SOURCE, MPI_ANY_TAG, fvGCM_inter_comm,
   status, ierror)
3. tag = status(MPI_TAG)
4. source = status(MPI_SOURCE)
5. if (tag .gt. 0) then
6. call mmf_call_gce2d(inputValues,outputValues) !! Compute gce2d()
7. !! Pass the outputs back to the sender, using the supplied tag
8. call mpi_send(outputValues, , MPI_BYTE, source, tag, fvGCM_inter_comm, ierror)
9. else if (tag .eq. 0) then
10. !! Exit the do-loop and terminate
11. exit mainLoop
12. else
13. !! Unexpected tag value
14. end if
15. enddo mainLoop

```

Figure 5. List 1: pseudocode of the `gce2d_task_main()` routine. Each of the mgGCE processes does a simple loop: receive `inputValues` from any fvGCM process, call `gce2d()` model, and return `outputValues` to the same fvGCM process that sent the `inputValues`.

happens in the routine `mmf_gce2d_task_assignment()` illustrated in Figure 7. The fvGCM processes agree among themselves to carve up the group of mgGCE processes into disjoint subsets, with each fvGCM process getting one of these subsets. Although the assignment is static, it isn't necessarily uniform.

For simplicity, we choose these disjoint subsets to be a contiguous block of processes, where “contiguous” is relative to their rank numbering within the intercommunicator (with the mgGCE as a remote group). The array `firstExtraGce2dTask`, associated with the fvGCM processes, gives the comm rank of the first mgGCE process of the subset assigned to that fvGCM process; `numGce2dTasksAssigned` is the number of mgGCE processes assigned to that fvGCM process, where `firstExtraGce2dTask` and `numGce2dTasksAssigned` are referred to as $S(\mathcal{J})$ and $T(\mathcal{J})$, respectively, in Figure 7, with \mathcal{J} representing the fvGCM's rank. For example, an fvGCM process with the rank of 13 has `numGce2dTasksAssigned(13)` or $T(13)$ processes, starting with intercomm rank `firstExtraGce2dTask(13)` or $S(13)$, and going up contiguously. $T(\mathcal{J})$ is roughly equal to (Q/P) but can be larger for some of fvGCM processes that need more latitudes than the other processes. The summation of all $T(\mathcal{J})$ should be equal to Q , namely, $\sum_{\mathcal{J}=1}^P T(\mathcal{J}) = Q$, where $S(1) = 1$ and $S(\mathcal{J} + 1) = S(\mathcal{J}) + T(\mathcal{J})$, $\mathcal{J} = 1 \sim P - 1$. These intercomm task index values are what the fvGCM process uses for the “destination”

field of the `mpi_send()` call, which requires an (intercommunicator, rank) pair with the “rank” relative to the remote group.

We can achieve load balancing by giving more mgGCE processes to the fvGCM processes that have more latitudes. As shown in the top panel of Figure 3, if one fvGCM process has to do three latitudes (or rows), and a different fvGCM process has to do four, we give the second fvGCM process more mgGCE processes (that is, the larger `numGce2dTasksAssigned` or T) than we give to the first. Thus, the second fvGCM process might be able to do the work within a single latitude in only three-quarters as much time, so the two fvGCM processes will both finish their work in roughly equal amounts of time. While each of fvGCM processes can also help perform GCE calculations to improve performance, no fvGCM process shares its work with other mgGCE processes outside of its assigned subset.

Simply speaking, although the load imbalance was introduced in association with the 1D non-uniform decomposition in the y direction in the original MMF, it can be mitigated or resolved by another 1D nonuniform decomposition in the x direction in the new MMF.

Dynamical distribution of mgGCE inputs over mgGCE processes. Lists 1 and 2 (Figures 5 and 6) provide pseudocodes to show how mgGCE and fvGCM processes interact to perform the calculations of 13,104 GCE copies. List 1 displays a main loop in

```

1. sendLoop: do d=1, depth
2. do i=1, numGce2dTasks
3. tag = 2 * nextColumn
4. call mpi_isend(inputValues(nextColumn), ..., tag, gce2d_intercomm, request
   (nextColumn), ierror)
5. nextColumn = nextColumn + 1
6. enddo
7. enddo sendLoop
8. sendRecvLoop: do while (nextColumn .le. NumLongitudes)
9. call mpi_iprobe (MPI_ANY_SOURCE, MPI_ANY_TAG, gce2d_intercomm, pending, status,
   ierror)
10. If (pending) then
11. whichTask = status (MPI_SOURCE)
12. whichColumn = status (MPI_TAG) / 2
13. call mpi_recv (outputValues(whichColumn),..., whichTask, ..., gce2d_intercomm ...)
14. call mpi_request_free(request(whichColumn), ierror)
15. tag = 2* nextColumn
16. call mpi_isend(inputValues(nextColumn), ... tag, gce2d_intercomm,
   request(nextColumn), ierror)
17. else
18. call mmf_call_gce2d (tmpInput .....) ! Running with fvGCM processes
19. endif
20. numCompleted = numCompleted + 1
21. nextColumn = nextColumn + 1
22. enddo sendRecvLoop
23. recvLoop: do while(numCompleted. lt. NumLongitudes)
24. call mpi_probe (MPI_ANY_SOURCE, MPI_ANY_TAG, gce2d_intercomm, pending, status,
   ierror)
25. whichColumn = status(MPI_TAG) / 2
26. call mpi_recv(outputValues(whichColumn), ... gce2d_intercom .....)
27. call mpi_request_free (request(whichColumn), ierror)
28. numCompleted = numCompleted + 1
29. enddo recvLoop

```

Figure 6. List 2: pseudocode of the `mmf_invoke_gce2d()` routine, which addresses distribution of global initial values to Q mgGCE processes from P fvGCM processes. The fvGCM processes send inputs to mgGCE processes (which run in parallel) and then receive outputs from them. The fvGCM processes continue to do so until the GCE calculation over the entire globe (that is, the calculation of the 13,104 GCE copies) is completed. Q is equal or larger than the multiple of P , so $Q \geq C * P$, where C is an integer.

`gce2d_task_main()`, where all mgGCE processes wait to receive `inputValues` from any fvGCM process (line 2), call the `gce2d()` model (line 6), and return `outputValues` (line 8) to the same fvGCM process that sent the `inputValues`. In short, mgGCE process does a simple loop: `recv input - compute gce - send output`. As mentioned, these mgGCE processes are essentially stateless, but in the current implementation, each mgGCE process is bound (assigned statically) to a single, particular fvGCM process, which is just a convenience for the fvGCM side.

All the controlling business happens on the fvGCM side, as in List 2. Each fvGCM process uses the fixed and static process mapping to

interact with the corresponding subsets of mgGCE processes. In addition to handling inputs and outputs, the fvGCM processes can also help perform GCE calculations to take advantage of their computational potential. Therefore, the inputs and computations for the entire globe are dynamically distributed to the mgGCE processes via the following three phases, which correspond to the three loops in the routine in List 2:

- In the `send` loop (lines 1–7), the fvGCM process sends out a lot of inputs to all of its mgGCE processes, queuing up `depth` inputs to each mgGCE process. Here the variable `depth` indicates the number of inputs that are already

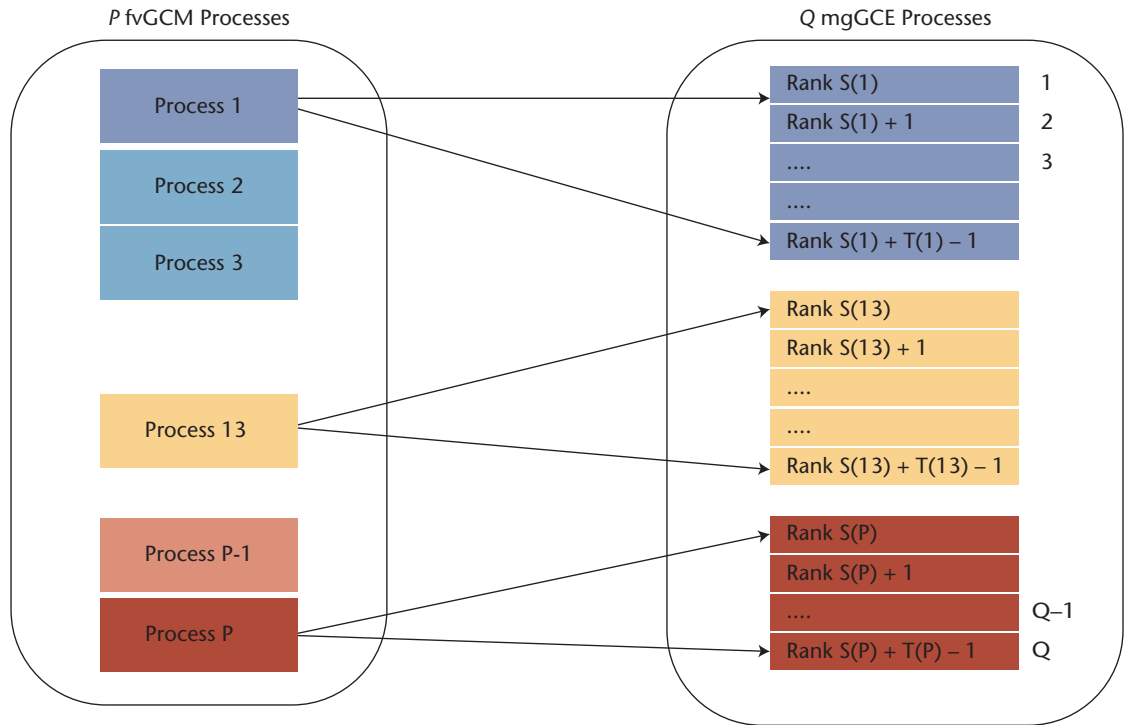


Figure 7. Mapping between P fvGCM process (left) and Q mgGCE processes (right). The fvGCM process with the rank of j receives a subset of Q mgGCE processes, say, $T(j)$. The choice of $T(j)$ depends on the amount of work (that is, the number of latitudes) assigned to the fvGCM task, and $\sum_{j=1}^P T(j) = Q$. Let $S(j)$ be the rank of the first mgGCE process (the starting index) in the j th subset of mgGCE processes, where $S(1) = 1$ and $S(j + 1) = S(j) + T(j)$, $j = 1, 2, \dots, P - 1$.

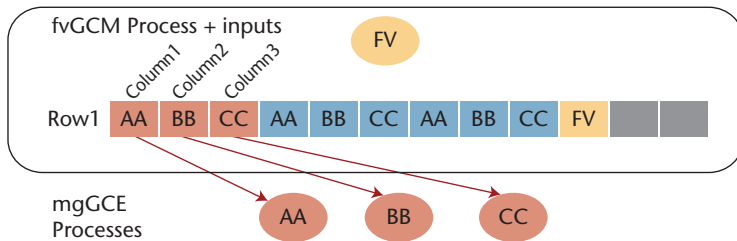


Figure 8. Dynamic distribution of mgGCE input values. Here, we assume that one specific fvGCM process (at the top, in yellow) has three mgGCE processes (at the bottom, in brown). The fvGCM process sends inputs to each of the three mgGCE processes, which begin processing them (brown). The fvGCM process also queues additional inputs (blue) for the mgGCE processes. While waiting for results from the mgGCE process, the fvGCM process can also help perform a GCE calculation (middle, in yellow). When computations with inputs in brown are done, the mgGCE process returns results and begins processing the next input in its queue. On receiving a result, the fvGCM process queues another input (gray) to whichever mgGCE process returned the result.

sent asynchronously to each mgGCE process's input queue.

- In the `sendRecv` loop (lines 8–22), the fvGCM process checks if any of its mgGCE processes have produced an output, and if so, sends another input piece of work to the mgGCE's queue. However, if none of the mgGCE processes

has produced an output, rather than waiting for the outputs from the mgGCE processes, one of the fvGCM process pitches in and does the next piece of work: running a copy of `gce2d()`. Then it goes back to checking for output. The fvGCM processes repeat the above until all the input has been sent, using `nextColumn` to keep track.

- In the `recvLoop` (lines 23–29), the fvGCM processes keep receiving outputs until all the output has been received, using `numCompleted` to keep track.

During these phases, the matchup between a particular `mpi_[i]send()` and a particular `mpi_[i]recv()` among fvGCM and mgGCE processes happens dynamically and asynchronously. Next, a simple illustration is given.

Let's assume that an fvGCM process has (`numGce2dTasks`, `NumLongitudes`) to be (3, 144) in List 2. Thus, it interacts with three mgGCE processes (we call them AA, BB, and CC in Figure 8) to dynamically distribute 144 inputs and finish 144 copies of GCE runs. With `depth=3` in the `sendLoop` (line 1 in List 2), the fvGCM process initiates nine (`depth*numGce2dTasks`)

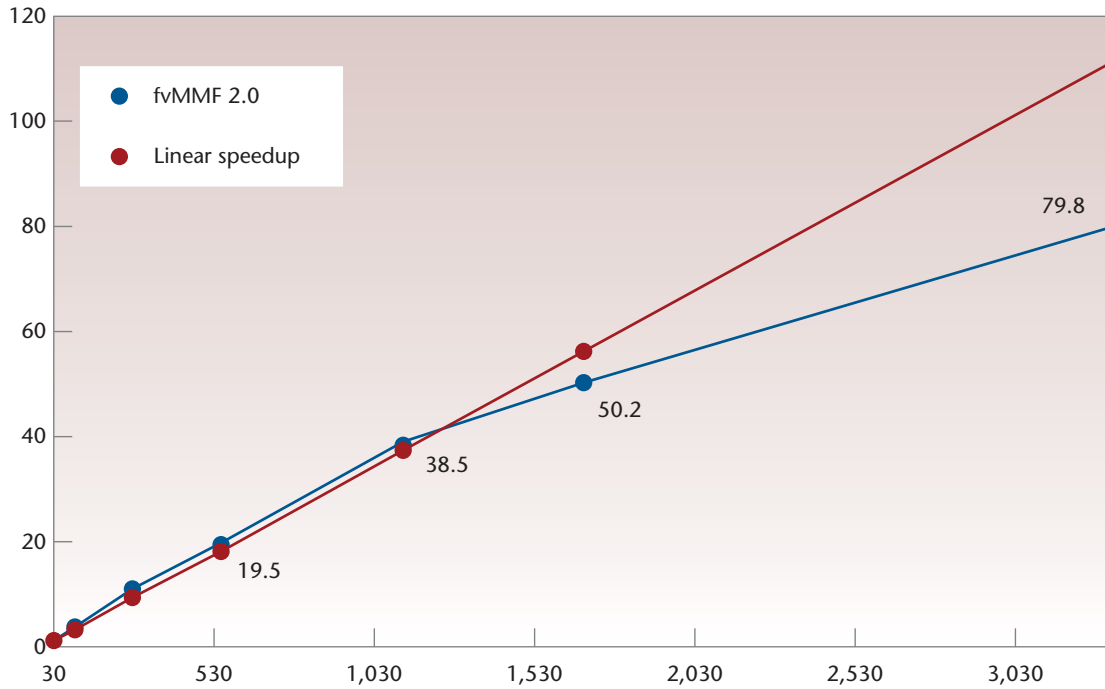


Figure 9. Parallel scalability of the MMF version 2.0 with a revised parallel implementation on the NASA Pleiades supercomputer. This figure shows that a speedup of nearly 80 \times is obtained as the number of cores increases from 30 to 3,335. Note that the original MMF could use only 30 cores.

`mpi_isend()` calls (line 4) to send `inputValue` records as follows: the first to AA, the second to BB, the third to CC, the fourth to AA, the fifth to BB, the sixth to CC, the seventh to AA, the eighth to BB, and the ninth to CC. This ends the `sendLoop`, and each of the three `mgGCE` processes now has three (`depth=3`) input records queued up.

Each of these `mgGCE` processes starts running the subroutine `gce2d()` as soon as it receives input, so the `fvGCM` process now enters the `sendRecv` loop (between lines 8–22) and calls `mpi_iprobe()` (line 9) to check for outputs sent from any of the `mgGCE` processes. Let's suppose that none has done so. Rather than wait, the `fvGCM` process takes the next piece of input (the 10th) and calls `gce2d()` (line 18). When the `fvGCM` process finishes and returns, it again checks for outputs from any `mgGCE` processes (line 9). Suppose BB has finished one `gce2d()` calculation, and so BB has returned the output for the second piece. BB then immediately begins work on its next piece, namely, the fifth piece. As soon as the `fvGCM` process sees the existence of a pending message (the outputs from the `mgGCE`), it calls `mpi_recv()` (line 13) to receive the outputs. The `fvGCM` process can tell it's the second piece by looking at the tag value and that it came from process BB. So the `fvGCM` process

sends the next piece of input (the 11th) to BB (line 16) and then returns back to the beginning of the loop in line 9 to check for outputs from any `mgGCE` processes. This loop continues until all 144 pieces of input have been handed out. The distribution of inputs and GCE runs is handed out dynamically as each previous piece of work is finished. The `fvGCM` process now enters the `recvLoop` (line 23). There's no more input to be handed out, so the `fvGCM` process waits until all the remaining pieces of work have been finished and returned.

Computational Results

To test our boosted scalability in action, we ran the improved MMF on the NASA Pleiades supercomputer, which is currently one of the most powerful general-purpose supercomputers in the world.

The Pleiades supercomputer is an SGI Altix ICE system with a peak performance of 2.88 Pflop/s. With 417 Tbytes of total memory, and 162,496 cores (plus 64 GPU nodes, each with 512 CUDA cores.), it achieves Linpack performance of 1.24 Pflop/s (as of June 2013). The system contains four different types of Intel Xeon processors—E5-2680v2 (Ivy Bridge), E5-2670 (Sandy Bridge), X5670 (Westmere), and X5570 (Nehalem)—to reach different needs and capacities on different

NASA projects. We used Nehalem processors in our benchmark.

Figure 9 shows a benchmark with encouraging parallel scalability up to 3,335 CPUs on Pleiades. Here, the speedup is determined by T_{30}/T , where T is the wall time to perform a five-day forecast with the MMF, and T_{30} is the time spent using 30 CPUs. We chose the run with 30 CPUs as a baseline because this configuration was previously used for production runs.^{9,10} We obtained a speedup of 3.42, 10.81, 19.46, 38.46, 50.16, and 79.77 by increasing the number of CPUs from 30 to 91, 273, 546, 1,115, 1,680, and 3,335 cores, respectively. As the baseline has load imbalances and excessive memory usage in the master process, it isn't surprising to obtain a superlinear speedup with lower CPU counts up to 1,115.

Further analysis of the MMF's throughput indicates that it takes roughly 41 minutes to finish a five-day forecast using 3,335 cores, which meets the requirement for performing real-time numerical weather prediction—that is, completion of a run within one hour. A three-year simulation would only take one day to run with 3,335 cores, as opposed to about 80 days with 30 cores. This speedup in wall time makes it far more feasible for increasing the resolution in the fvGCM and studying TCs' interannual variability.

Ideally, the parallelism that leads to an effective 2D domain decomposition in the mgGCE can be applied to other types of column-based physics parameterizations. The approach we described here also lays the groundwork for more sophisticated modeling to solve extraordinarily complex problems with advanced computing power. For example, improved scalability makes it possible to deploy a more advanced MMF with the fvGCM at a higher resolution, say, $0.25^\circ \times 0.36^\circ$, which has much larger grid points ($721 \times 1,000$) and thus requires many more copies of GCEs. In addition, by taking load balance into consideration, the current implementation makes it feasible to choose a variety of GCEs in the mgGCE. Further improvements include the implementation of an efficient I/O module or a parallel I/O module with a CPU layout that could differ from that in either the fvGCM or the mgGCE.

The current parallel implementation in the MMF is a coarse-grained parallelism compared to the parallelism inside a GCE. Because an individual GCE was previously implemented with

its native 2D domain decomposition, another level of parallelism (fine-grain parallelism) inside each copy of the GCE could greatly expand the number of CPUs for MMF runs. Potentially, the coupled MMF, along with the mgGCE, could be scaled at a multiple of 13,104 CPUs, which is a subject for future study.

SE

Acknowledgments

We're grateful to the following organizations for their support: the NASA Earth Science Technology Office, the Advanced Information Systems Technology (AIST) Program, the NASA Computational Modeling Algorithms and Cyberinfrastructure (CMAC) program, and the NASA Modeling, Analysis Prediction (MAP) Program. Resources supporting this work were provided by the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center. Finally, we thank Jill Dunbar of NASA ARC/NAS for proof-reading this manuscript.

References

1. L. Bengtsson, I. Hodges, and M. Esch, "Tropical Cyclones in a T159 Resolution Global Climate Model: Comparison with Observations and Re-analyses," *Tellus A*, vol. 59, no. 4, 2007, pp. 396–416.
2. B.-W. Shen et al., "Hurricane Forecasts with a Global Mesoscale-Resolving Model: Preliminary Results with Hurricane Katrina," *Geophysical Research Letters*, vol. 33, no. 13, 2006; doi:10.1029/2006GL026143.
3. B.-W. Shen et al., "Predicting Tropical Cyclogenesis with a Global Mesoscale Model: Hierarchical Multi-scale Interactions During the Formation of Tropical Cyclone Nargis," *J. Geophysical Research*, vol. 115, no. D14, 2010; doi:10.1029/2009JD013140.
4. B.-W. Shen, W.-K. Tao, and B. Green, "Coupling Advanced Modeling and Visualization to Improve High-Impact Tropical Weather Prediction (CAMVis)," *Computing in Science & Eng.*, vol. 13, no. 5, 2011, pp. 56–67.
5. B.-W. Shen et al., "Genesis of Twin Tropical Cyclones as Revealed by a Global Mesoscale Model: The Role of Mixed Rossby Gravity Waves," *J. Geophysical Research*, vol. 117, no. D13, 2012; doi:10.1029/2012JD017450.
6. B.-W. Shen et al., "Advanced Visualizations of Scale Interactions of Tropical Cyclone Formation and Tropical Waves," *Computing in Science & Eng.*, vol. 15, no. 2, 2013, pp. 47–52.
7. B.-W. Shen et al., "Genesis of Hurricane Sandy (2012) Simulated with a Global Mesoscale Model," *Geophysical Research Letters*, vol. 40, 2013, pp. 1–7; doi:10.1002/grl.50934.


8. D. Randall et al., "Breaking the Cloud Parameterization Deadlock," *Bull. Am. Meteorological Soc.*, vol. 84, no. 11, 2003, pp. 1547–1564.
9. W.-K. Tao et al., "A Goddard Multi-Scale Modeling System with Unified Physics," *WCRP/GEWEX Newsletter*, vol. 18, no. 1, 2008, pp. 6–8.
10. W.-K. Tao et al., "Multiscale Modeling System: Development, Applications and Critical Issues," *Bull. Am. Meteorological Soc.*, vol. 90, no. 4, 2009, pp. 515–534.
11. R. Atlas et al., "Hurricane Forecasting with the High-Resolution NASA Finite Volume General Circulation Model," *Geophysical Res. Letters*, vol. 32, no. 3, 2005; doi:10.1029/2004GL021513.
12. W.-K. Tao and J. Simpson, "The Goddard Cumulus Ensemble Model. Part I: Model Description," *Terrestrial, Atmospheric and Oceanic Sciences*, vol. 4, no. 1, 1993, pp. 19–54.
13. W.-K. Tao et al., "Convective Systems over South China Sea: Cloud-Resolving Model Simulations," *J. Atmospheric Science*, vol. 60, no. 24, 2003, pp. 2929–2956.
14. J.-M. Juang et al., "Parallelization of NASA Goddard Cloud Ensemble Model for Massively Parallel Computing," *Terrestrial, Atmospheric and Oceanic Sciences*, vol. 18, no. 3, 2007, pp. 593–622.
15. S.-J. Lin, B.-W. Shen, and W. P. Putman, "Application of the High-Resolution Finite-Volume NASA/NCAR Climate Model for Medium-Range Weather Prediction Experiments," *EGS-AGU-EUG Joint Assembly*, 2003, abstract 1738.
16. W. Putman, S.-J. Lin, and B.-W. Shen, "Cross-Platform Performance of a Portable Communication Module and the NASA Finite Volume General Circulation Model," *Int'l J. High Performance Computing Applications*, vol. 19, no. 3, 2005, pp. 213–223.


Bo-Wen Shen is a research scientist at the University of Maryland, College Park, and NASA Goddard Space Flight Center. His research interests include high-resolution global and regional modeling, high-end computing, and numerical hurricane and weather prediction. Shen has a PhD in atmospheric sciences from North Carolina State University. He's a member of the American Geophysical Union. Contact him at bowen.shen@gmail.com or bo-wen.shen-1@nasa.gov.


Bron Nelson is a senior software engineer for Computer Sciences Corporation (CSC) and a member of the Visualization Group at the NASA Ames Research Center's Advanced Supercomputing facility. Bron has an MS in computer science from the University of California, Los Angeles. Contact him at bron.c.nelson@nasa.gov.

Samson Cheung is a senior software engineer for Computer Sciences Corporation (CSC) and a member of the Application Group at the NASA Ames Research Center's Advanced Supercomputing facility. His research interests are in application performance and parallel I/O on new architectures. Samson has a PhD in applied mathematics from the University of California, Davis. Contact him at samson.h.cheung@nasa.gov.


Wei-Kuo Tao is a senior research meteorologist at the NASA Goddard Space Flight Center and the leader of Goddard Mesoscale Modeling and Dynamic Group. His research interests include cloud physics and modeling mesoscale convective systems. Tao has a PhD in atmospheric sciences from the University of Illinois. He's a Fellow of the American Meteorological Society and Royal Meteorological Society. Contact him at wei-kuo.tao-1@nasa.gov.

 Selected articles and columns from IEEE Computer Society publications are also available for free at <http://ComputingNow.computer.org>.



IEEE  computer society NEWSLETTERS
Stay Informed on Hot Topics

COMPUTING NOW
TRAINING SPOTLIGHT
TRANSACTIONS CONNECTION
WHAT'S NEW BUILD YOUR DIGITAL
IN COMPUTER CAREER COMPUTING LIBRARY
CS CONNECTION NEWS
DIGITAL LIBRARY NEWS FLASH
CONFERENCE CONNECTION
TRANSACTIONS CONNECTION
COMPUTING NOW
TRAINING SPOTLIGHT
MEMBER CONNECTION

 computer.org/newsletters