# The Scalable Modeling System: Directive-based code parallelization for distributed and shared memory computers

*M. Govett[1], L. Hart, T. Henderson, J. Middlecoff* and D. Schaffer[*]*
*National Oceanic and Atmospheric Administration*
*Forecast Systems Laboratory*
*Boulder, Colorado 80305, USA*

***Submitted to the Journal of Parallel Computing (July 2002)***

**Abstract**

A directive-based parallelization tool called the Scalable Modeling System (SMS) is described. The user inserts directives in the form of comments into existing Fortran code. SMS translates the code and directives into a parallel version that runs efficiently on shared and distributed memory high-performance computing platforms including the SGI Origin, IBM SP2, Cray T3E, Sun, and Alpha and Intel clusters. Twenty directives are available to support operations including array re-declarations, inter-process communications, loop translations, and parallel I/O operations. SMS also provides tools to support incremental parallelization and debugging that significantly reduces code parallelization time from months to weeks of effort. SMS is intended for applications using regular structured grids that are solved using finite difference approximation (FDA) or spectral methods. It has been used to parallelize ten atmospheric and oceanic models but the tool is sufficiently general that it can be applied to other structured grids codes. Recent performance comparisons demonstrate that the Eta, HYCOM and ROMS models, parallelized using SMS, perform as well or better than their OpenMP or MPI counterparts.

*Keywords:* Directive-based parallelization tool; Weather and ocean models; Automatic parallel code generation; Fortran source code translator; Distributed memory computers

## 1. Introduction

Both hardware and software of high-performance computers (HPCs) have evolved significantly in the last decade. Computers quickly become obsolete; typically a new generation is introduced every two to four years. HPCs now comprise a wide range and class of systems including shared memory systems called Symmetric Multi-Processors (SMPs), fully distributed memory systems and hybrid systems that connect multiple SMPs using some form of high speed network. To scale to large numbers of processors, these systems are either distributed or hybrid systems, a classification referred to in this paper as distributed memory computers (DMCs). Typically DMC systems are proprietary vendor-based solutions, but commodity-based DMCs have emerged as an attractive alternative due to their superior price / performance and to the increasing adoption of hardware and software standards by the industry.

While DMC systems offer the possibility of scalable high-performance, the additional work required to port codes to these systems and to make them run efficiently can be significant. DMCs are generally more difficult to program because they require the user to break up the problem domain into smaller sub-domains that are then solved in parallel. If memory is physically distributed, messages must be passed between systems when data sharing is required. These considerations often require significant changes to the serial code, and in some cases, the code may need to be completely rewritten. Additional considerations arise when codes are

[1] *E-mail address*: govett@fsl.noaa.gov (M. Govett)
[*] *Cooperative Institute for Research in the Atmosphere, Colorado State University Fort Collins, CO 80523 USA*

moved between different computing platforms for operational and developmental use. If the codes rely on a specific computer architecture or vendor specific routines, diverging versions of the code become increasingly difficult to maintain.

Despite these issues, operational weather centers and research groups are increasingly selecting large DMCs to meet their computational needs. For example, in 1996 the United Kingdom Meteorological Office purchased a 696-node Cray T3E, and in 1997 the National Centers for Environmental Prediction (NCEP) purchased a 768-node IBM-SP system that was recently upgraded to over 2700 processors. During the procurement of these systems, operational centers typically request that vendors provide software support staff as part of their budget. These people are tasked with porting the operational codes to the vendor's DMCs – a task that often takes multiple man-years to complete. Research laboratories do not typically have the luxury of paying vendors to parallelize their models; instead they must expend a great deal of time and effort to learn the complexities of parallel programming. In either case however, porting codes to new systems can represent a significant commitment of time and resources.

The primary mission of the National Oceanic and Atmospheric Administration's (NOAAs) Forecast Systems Laboratory (FSL) is to transfer atmospheric science technologies to operational agencies within NOAA, such as the National Weather Service, and to others outside the agency. In the last decade, FSL has parallelized a variety of weather and ocean models that run on DMCs in operational centers and at research institutions around the world.

Central to FSL's success with model parallelization and the use of DMCs has been the development of the Scalable Modeling System (SMS). SMS is high-level directive-based tool that was designed to reduce the time and effort required to parallelize codes targeted for DMCs. Further, a single source code can be maintained for both serial and parallel execution that can easily be ported between DMCs. SMS also provides debugging tools that enable code parallelization to be measured in weeks rather than months of effort.

SMS is intended for applications using regular structured grids that are solved using finite difference approximation (FDA) or spectral methods. SMS provides support for mesh refinement of nested models, and can transform data between grids that have been decomposed differently. Models parallelized using SMS include the Global Forecast System (GFS) [23] and the Typhoon Forecast System (TFS) [9] for the Central Weather Bureau in Taiwan, the Regional Ocean Modeling System (ROMS) [17], the Hybrid Coordinate Ocean Model (HYCOM) [5], the National Centers for Environmental Prediction (NCEP) Eta model [26], the high resolution limited area Quasi Non-hydrostatic model (QNH) [24], the Princeton Ocean Model (POM) [8], and the 20 km Rapid Update Cycle (RUC) model running operationally at NCEP [4]. These models have demonstrated good performance and scaling on most high-performance distributed memory computing platforms including the IBM SP, Cray T3E, SGI-Origin, Alpha-Linux Clusters, and Intel-Linux Clusters.

The rest of this paper describes SMS in more detail. Section 2 surveys common approaches to code parallelization and is followed by a description of SMS in Section 3. This section describes the code parallelization process using SMS and highlights some advanced and unique features of the software that are not available in other parallelization tools. In Section 4, case studies are presented that describe the SMS parallelization of three models: the operational Eta model used by the National Weather Service, the HYCOM ocean model, and the ROMS ocean model. In these case studies, the performance of the SMS parallelized models are compared to

OpenMP and MPI versions of these codes. Finally, Section 5 concludes and highlights some additional work that is planned.

## 2. Approaches to parallelization

In the past decade, several distinct approaches have been used to parallelize Fortran codes.

### *2.1 Message passing libraries*

Message-passing libraries, such as Message Passing Interface (MPI) [16], represent an approach suitable for shared or distributed memory architectures. Message passing typically requires the sender and receiver to be involved in the communication: the source process makes a call to send data and the destination process makes a call to receive it. To reduce communications overhead, the Shared Memory Access Library (SHMEM) developed by Cray, does one-sided get/put operations between the source and receiver processes. This optimization, also used in some MPI implementations, significantly speeds communications but typically requires explicit handling of inter-process synchronization. Although the scalability of parallel codes using message-passing libraries can be quite good, they are relatively low-level and can require the programmer to expend a significant amount of effort to parallelize their code. Further, the resulting code may differ substantially from the original serial version and code restructuring is often desirable or necessary.

### *2.2 Parallelizing compilers*

These solutions offer the ability to automatically produce a parallel code that is portable to shared and distributed memory machines. The compiler does the dependence analysis and offers the user directives and/or language extensions that reduce the development time and the impact on the serial code. The most notable example of a parallelizing compiler is High-performance Fortran (HPF) [21]. In some cases the resulting parallel code is quite efficient (Portland Group, 1999), but there are also deficiencies in this approach. Compilers are often forced to make conservative assumptions about data dependence relationships, which impact performance [12][20]. In addition, weak compiler implementations by some vendors result in widely varying performance across systems [30].

Another compiler approach, called Co-Array Fortran [31], provides a language extension to Fortran 95, in the form of square bracket syntax, to provide limited message passing functionality. While the syntax is simple, extensible and flexible, the programming model is similar to MPI and is therefore sufficiently low level that significant modifications to the serial code are required during code parallelization. Portability is also an issue since Co-Array Fortran is only available on the Cray T3E.

### *2.3 Interactive parallelization tools*

This classification combines the automatic analysis capabilities of a parallelization tool with user knowledge of the code in order to produce a parallel version. One interactive tool, called the Parallelization Agent, automates the tedious and time consuming tasks while requiring the user to provide the high-level algorithmic details [22]. Another tool, called the Computer-Aided Parallelization Tool (CAPTools), attempts a comprehensive dependence analysis [20]. This

commercially available tool is highly interactive, querying the user for both high level information (decomposition strategy) and lower level details such as loop dependencies and ranges that variables can take. While interactive tools offers the possibility of a quality parallel solution in a fraction of the time required to analyze dependencies and generate code by hand, limitations exist in their ability to offer efficient code parallelization of scientific codes containing more advanced functionality such as nesting, multiple decompositions, and Fortran 90 constructs. In addition, some hand rewriting may be required to obtain good parallel performance [13].

### *2.4 High-level library-based tools*

Library-based tools, such as the Runtime System Library (RSL) [27] and the Nearest Neighbor Tool (NNT) [35], are built on top of the lower level libraries, such as MPI, and serve to relieve the programmer of handling many of the details of message passing programming. Performance optimizations can be added to these libraries that target specific machine architectures. While a number of models were successfully parallelized using NNT [2][11][18], the serial and parallel versions of the code were distinctly different and had to be maintained separately. Further, parallelization is time-consuming and invasive, since code must be inserted by hand and the user is still required to do dependence analysis themselves.

Source translation tools have been developed to help modify these codes automatically. One such tool, the Fortran Loop and Index Converter (FLIC), generates calls to the RSL library using command line arguments to identify decomposed arrays and loops needing transformations [28]. While useful, this tool has limited capabilities. For example, it was not designed to handle multiple data decompositions, inter-process communications, or nested models.

### *2.5 Modeling frameworks*

This classification loosely describes a software engineering approach whose aim is to minimize the impact of parallelization on serial codes using techniques such as code restructuring, abstraction, and modular code design. One example of this approach is the Weather Research and Forecast (WRF) model. This model was designed to limit the impact of parallelization and parallel code maintenance by confining MPI-based communications calls into a minimal set of model routines called the mediation layer [29]. Another approach, called the Flexible Modeling System [3], uses an object oriented design to encapsulate data structures used in the application as objects in a class library. These objects are manipulated using operators, defined in the library, in order to perform arithmetic, rotational and differential operations required by the application. Lower layer class libraries are provided to support domain decomposition, communication and parallel I/O. Both of these approaches provide a good way to separate the needs of the application from the complexities of parallel programming, but significant code restructuring may be required.

Two additional efforts, the Earth System Modeling Framework (ESMF) [10] and the Program for Integrated Earth System Modeling [32] projects, have begun recently with a goal of standardizing the interfaces between the framework and the applications that use them. Standard interfaces allow scientific codes to be more easily moved between models when they are built using the same framework. In addition to providing traditional library-based low-level communications routines, these approaches plan to support coupling of models and the robust handling of model grids that are required by climate applications. These tools are slated to be

available in 2005, but at this time, it is unclear the extent to which source code changes will be required in order to conform to the frameworks.

*2.6 Directive-based parallelization*

Companies such as Cray and SGI historically used this approach to support loop level shared memory parallelization. More recently, a standard a set of directives called OpenMP was developed and has become widely used in the scientific community. OpenMP can be used to quickly produce parallel code, with minimal impact on the serial version. However, OpenMP is not available on many distributed memory architectures. To get around this restriction, software developers have obtained limited success combining OpenMP (shared memory parallelism) with MPI (distributed memory parallelism) on hybrid DMCs. In either case, however, obtaining good scalable performance often requires as much effort as when MPI is used.

Another approach, and the topic of this paper, is a directive-based tool called SMS that can be used on distributed or shared memory machines.

## 3. Overview of SMS

SMS is a layered set of software and directives built on top of low level communications libraries such as MPI or SHMEM. User access to code parallelization is provided through the highest layer component of SMS called the Parallel Pre-Processor (PPP). PPP is a Fortran code analysis and translation tool built using the Eli compiler construction system [15]. PPP analyzes the serial code and user-inserted SMS directives to determine how the code should be modified.

Translation of Fortran 77 codes is fully supported; however, handling of Fortran 90 language constructs is currently limited to modules, full array assignments, and other commonly used statements. Further details of the supported Fortran 90 language constructs are provided on the SMS web site (http://www-ad.fsl.noaa.gov/ac/sms.html).

The code parallelization process is illustrated in **Figure 1**. The programmer inserts the directives, which appear as comments, directly into the serial code. PPP translates the directives and serial code into a parallel version. Code modifications include loop translations, array re-declarations, inter-process communications, and the transformation of I/O statements to handle decomposed and non-decomposed variables. Since the programmer adds only comments to the code, there is no impact to the serial version. Once the parallel code is running it can be debugged and optimized for high performance using tools provided by SMS. Further, no code changes are required when porting the SMS serial version to other shared and distributed memory machines.

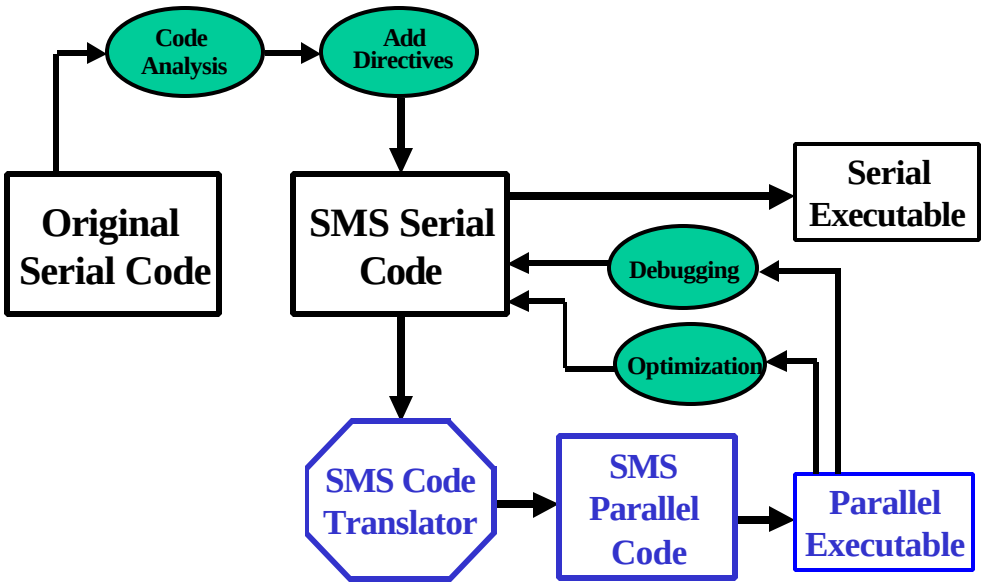


**Figure 1:** SMS directives are added to the original serial code during code parallelization. The SMS serial code can then be run serially as before, or translated by SMS to generate parallel code. Once the code is running, SMS provides tools to debug the parallel code and to optimize it for high performance.

To ensure the translated code is recognizable to the author, SMS changes only those lines of code that must be modified; the rest of the serial code including comments and white space remain untouched. While this feature makes it easy to look at the generated code, experience has shown that most users never do.

SMS has been designed to handle most aspects of parallel programming including code transformation, debugging and optimization operations. The breadth of SMS support for these operations is illustrated in **Figure 2**. Access to these functions is primarily through twenty SMS directives that are inserted into the application code. Key SMS directives used to support these operations will be briefly described. All directive names are preceded by "CSMS$" or "!SMS$" which, for brevity, will be omitted when referring to them in this paper. Further information about these directives is available in the SMS User's Guide [19] and the SMS Reference Guide [14].

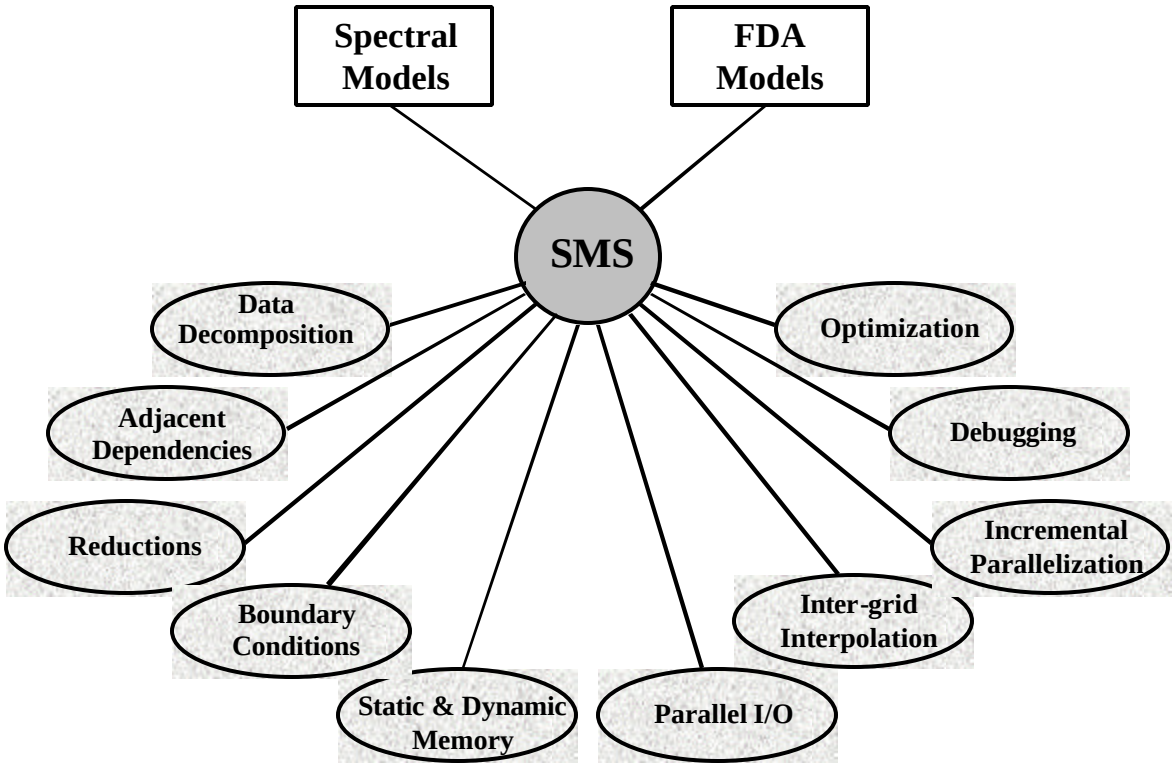


**Figure 2:** A functional diagram of SMS. SMS support is provided for the transformation of the serial code, and for debugging and optimization of the parallel code.

### *3.1 Data decomposition*

The most important step in code transformation is for the user to define one or more data decompositions and to identify the variables that will be associated with them. Five SMS directives are available to support data decomposition. The directives DECLARE_DECOMP, CREATE_DECOMP, and DISTRIBUTE, in combination, enable the programmer to decompose the data among the processes. DECLARE_DECOMP names a decomposition and CREATE_DECOMP initializes it at run-time. The DISTRIBUTE directive is used to specify if and how dimensions of individual arrays are decomposed based on the decompositions defined by DECLARE_DECOMP. During code translation decomposed arrays will be re-declared into the locally defined sub-portions of the global array. Do-loops that operate on these decomposed arrays will be modified to the appropriate local start and stop values using the PARALLEL directive.

In the event multiple decompositions are required, such as in nested models, the TRANSFER directive handles the communication necessary to move data between coarse and fine nests. Multiple decompositions are also useful in spectral models where computations occur in alternating phases. For example, a decomposition in latitudes may be optimal for the physics and Fast Fourier Transform code, while a decomposition in longitudes or in the vertical may be ideal for the Legendre transforms.

For example, **Figure 3** shows an SMS program in which the decomposition `my_dh` is declared (DECLARE_DECOMP: line 3) and then referenced by the DISTRIBUTE directive

(lines 5, 9) to associate the decomposed array dimensions of *x* and *y* with the data decomposition *my_dh*. Once SMS understands how arrays are decomposed, parallelization becomes primarily an issue of where in the code the user wishes to perform communications and not how data will be moved to accomplish these operations. SMS retains all of the information necessary to access, communicate, and input and output decomposed and non-decomposed arrays through the use of the user-named decomposition.

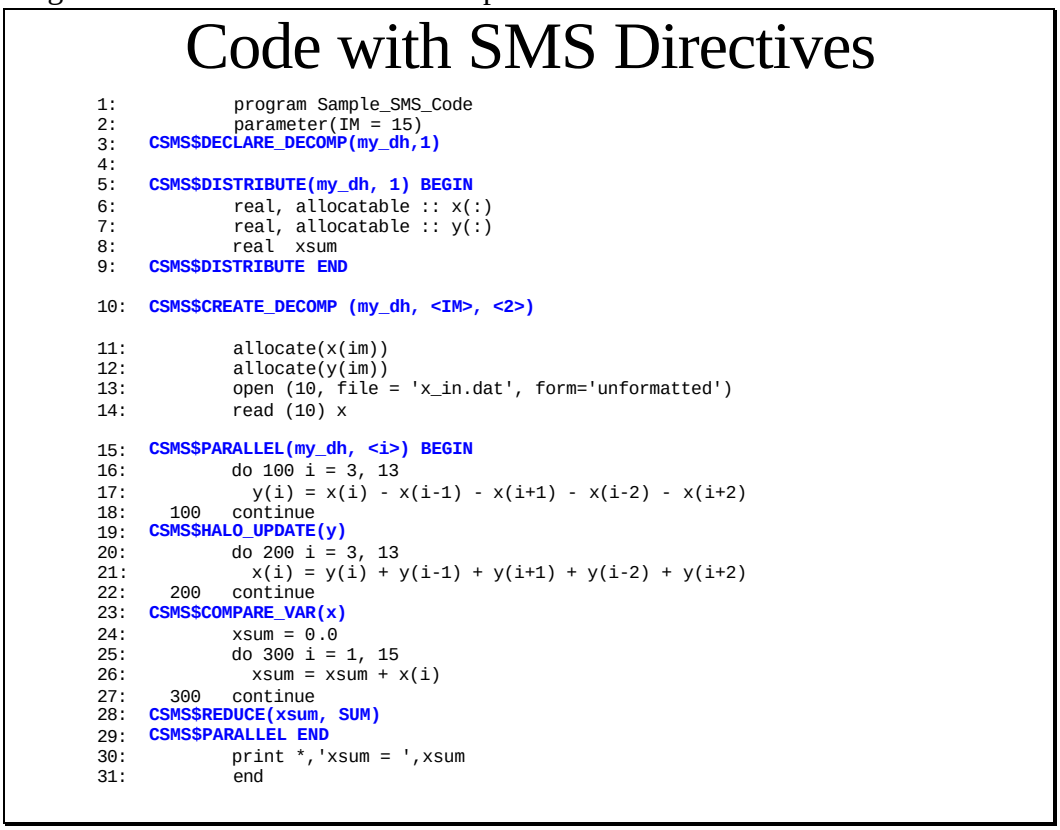


**Figure 3**: An example of a program using SMS directives. The DISTRIBUTE directive is used to map sub-sections of the arrays x and y to the decomposition named by "my_dh". Each process executes on its portion of these decomposed arrays in the parallel region bounded by PARALLEL BEGIN / END (lines 15-29). Using the decomposed arrays, each process computes a local sum (lines 25-27). A global sum is then calculated using the REDUCE directive (line 28) and then output.

### *3.2 Adjacent dependencies*

Once a data decomposition is chosen, the code must be analyzed to determine where data dependencies occur. For example, the computation of y(i,j) in the statement:

```
y(i,j) = x(i+1,j) + x(i-1,j) + x(i,j+1) + x(i,j-1)
```

depends on the (i+1,j), (i-1,j), (i,j+1), and (i,j-1) points of the "x" array. This is called an adjacent dependence. When these data points are not local to the processor, they will need to be obtained from another processor. The most common approach is to define a halo region so that off process data values can be communicated efficiently. This type of operation is called a halo

update or exchange and is illustrated in **Figure 4**. The SMS directive that provides this capability is called HALO_UPDATE.

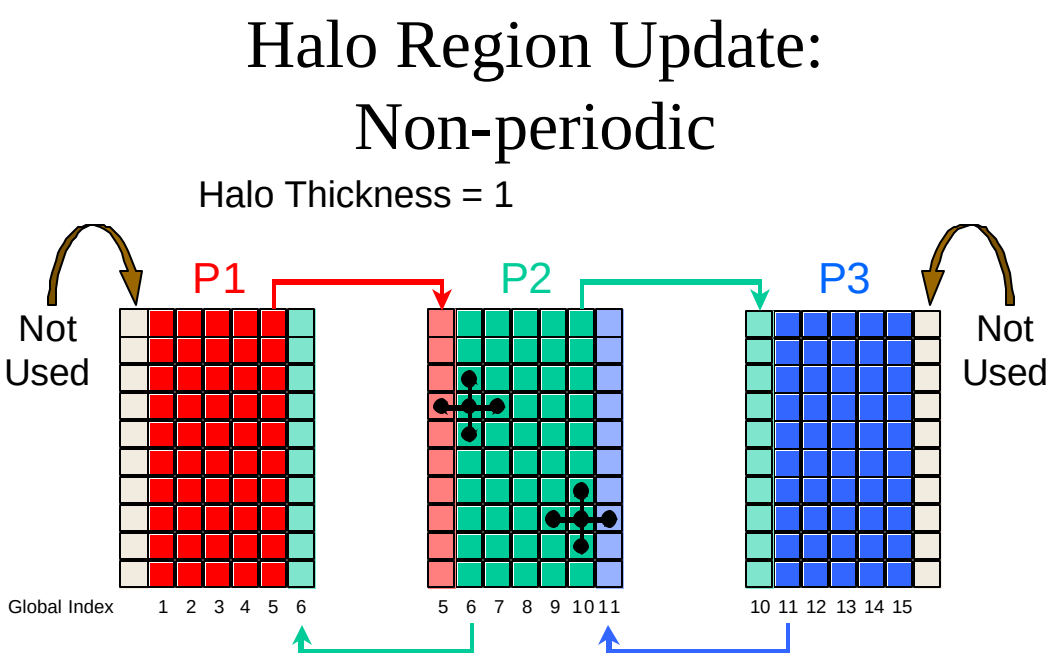


**Figure 4**: An illustration of how the HALO_UPDATE directive is used to update halo regions (checkerboard areas) on processes P1, P2 and P3. For example, the first column of P2 is sent to P1 where it is stored in the halo region just to the right of P1's data. The last column of P1 is stored in the left halo region of P2. The other communication works analogously.

For example, the HALO_UPDATE directive in **Figure 3** is used to update the halo points of array *y* to handle an adjacent data dependency that exists between loops 100 and 200. The user is only required to specify the name of the variable requiring a halo update. Using information from the directive and obtained via code analysis, SMS determines how much of the halo region each process must communicate, where the information must go, and where it should be stored. Process synchronization is also handled by SMS for communications operations where appropriate.

*3.3 Reductions*

The SMS directive, REDUCE, is used to compute reductions involving global sums, maximums and minimums. Since summation is not associative, the computation of global sums may not lead to exactly the same results on different numbers of processors. To alleviate this inconsistency, SMS provides a bit-wise exact reduction capability which performs exactly the same order of arithmetic operations that are executed in the serial program.

*3.4 Boundary conditions*

Handling of global boundaries requires special treatment to ensure that only the process that owns a data point will perform the given operation. For example, the initialization of boundary points on three processes (P1, P2 and P3) is illustrated in **Figure 5**. Only process P1 should do

the assignment for `x(1)` and process P3 for `x(15)`. The SMS directive GLOBAL_INDEX is provided to do this operation.

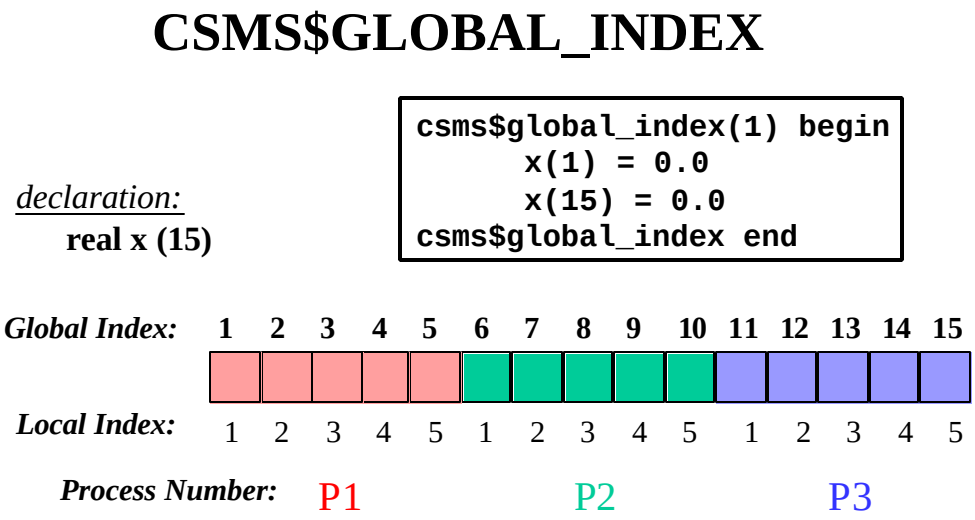


**Figure 5**: An illustration of SMS handling of global boundaries using GLOBAL_INDEX on the array "x" that is decomposed over three processes: P1, P2 and P3.

Periodic boundary conditions are also handled by SMS. The user can specify periodic boundary conditions using the PERIODIC keyword when creating the data decomposition using the CREATE_DECOMP directive. This option causes SMS to fill the halo points representing the model's global boundaries during initialization, and to update the halo region boundaries when HALO_UPDATE is used. For example in **Figure 4**, process P1's boundary points, tagged "not used", will be updated with P3's data (global index 15).

*3.5 Static & dynamic memory*

SMS provides support for both static and dynamic memory allocation. For models using dynamic memory allocation, SMS retains the concept of a global address space, so translations between the serial and process local index values are not required. For models using static memory allocation, additional arguments in DECLARE_DECOMP are used to declare the process local sizes of the decomposed arrays. Global arrays are referenced using process local indices (**Figure 5**). Two SMS directives, TO_LOCAL and TO_GLOBAL, are provided to convert between global and process local addressing when required.

*3.6 Parallel I/O*

SMS supports standard Fortran input and output of decomposed and non-decomposed arrays without requiring SMS directives. SMS acquires the information it needs to handle I/O though the decomposition directives described in Section 3.1. Using this information, SMS automatically generates the communication calls needed to read or write data from disk. An SMS server process, shown in **Figure 6**, is designated, by default, to handle all I/O operations. On input, data are read by the server and then scattered to the appropriate compute processes.

Similarly, decomposed output data are gathered by the server and then written to disk in the proper order. This server process can also be optionally omitted at run-time for runs made on small numbers of processes.

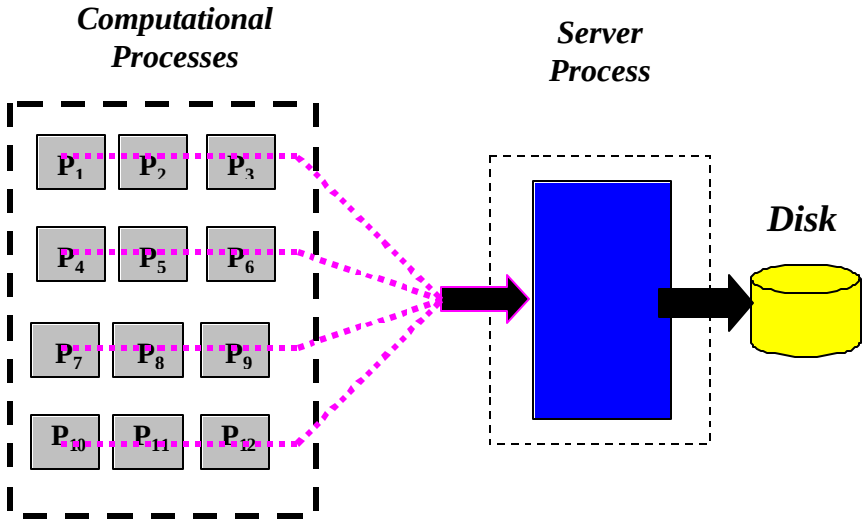


**Figure 6:** An illustration of SMS output when a server process is used. SMS output operations pass data from the computational domain to the server process. The data are re-ordered on the server process before being written to disk.

### *3.7 Inter-grid interpolation*

For nested models that use mesh refinement to improve grid resolution in critical areas, interpolation between nests may be required. Interpolated data from the coarse "parent" grid may be used to compute boundary information for the fine "child" grid or visa versa. Inter-grid interpolation is also used to couple models that are based on different grids. One example is the coupling of ocean and atmospheric models. SMS supports this functionality through a user-specified interpolation scheme. Once an interpolation scheme has been defined, the TRANSFER directive is used to move data between grids using this scheme.

### *3.8 Incremental parallelization*

SMS provides support for simplifying code parallelization with a directive called SERIAL that permits serial execution of selected portions of the user's code. This directive has several important uses. First, it allows users to parallelize their code incrementally rather than being forced into an all-or-nothing approach. Once assured of correct results, the user can remove these serial regions and further parallelize their code. Second, this directive can be used to avoid the parallelization of some sections of code that are either executed infrequently (e.g. model initialization) or cannot be parallellized by SMS, such as NetCDF I/O [34]. Third, users may simply choose to not parallelize some sections of code if adequate performance is attained.

Serial regions are implemented by gathering all decomposed arrays, executing the code segment on a single process, then scattering decomposed or broadcasting non-decomposed results back to each processor as illustrated in **Figure 7**. In this figure, the routine `not_parallel` executes on a single process and references global arrays that have been gathered by the appropriate SMS routines. While the extra communications required to do

gather or scatter operations will slow performance, this directive's versatility has proven to be very useful during code parallelization.

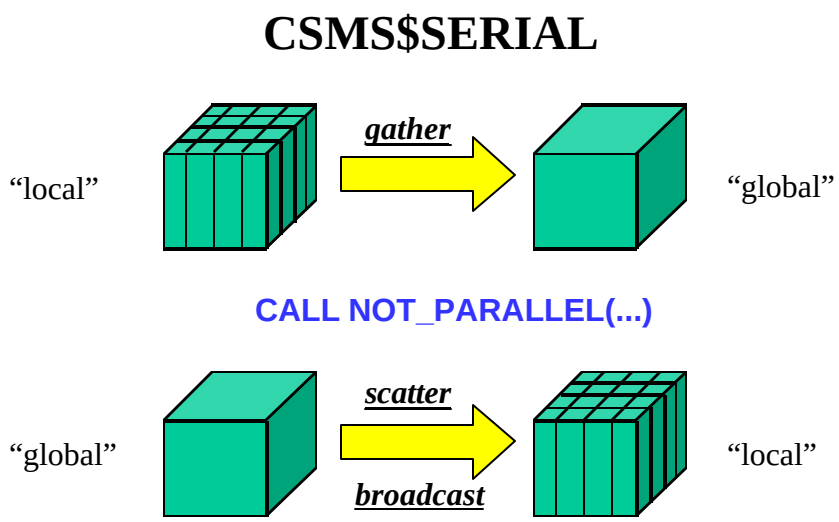


**Figure 7**: An illustration of how SMS supports incremental parallelization. Prior to execution of the serial region of code, decomposed arrays are gathered into global arrays, referenced by the serial section of code, and then results are scattered or broadcast back to the processors at the end of the serial region.

*3.9 Debugging*

Finding run-time bugs during the initial code parallelization or ensuing code maintenance phase can be the most difficult and time consuming task required in running codes on a DMC system. SMS provides a number of advanced features that significantly simplify debugging and streamline code parallelization. The best way to verify the accuracy of any parallelization effort is to compare the output files of different parallel runs (e.g. 1, 2, 4 processors) to a serial "control" run. This requires that the slower but more accurate bit-wise exact reductions, described in Section 3.3, be turned on during the control runs. Once the model is producing the correct results, bit-wise exact reductions can be turned off for the production runs when high-performance is required.

In addition, two SMS directives have been developed to support debugging. As illustrated in **Figure 8**, the COMPARE_VAR directive is used to verify interior region data points are correct, and CHECK_HALO is used similarly for the halo points.

# SMS Debugging Directives

Insert directives in the code to verify
array values are correct

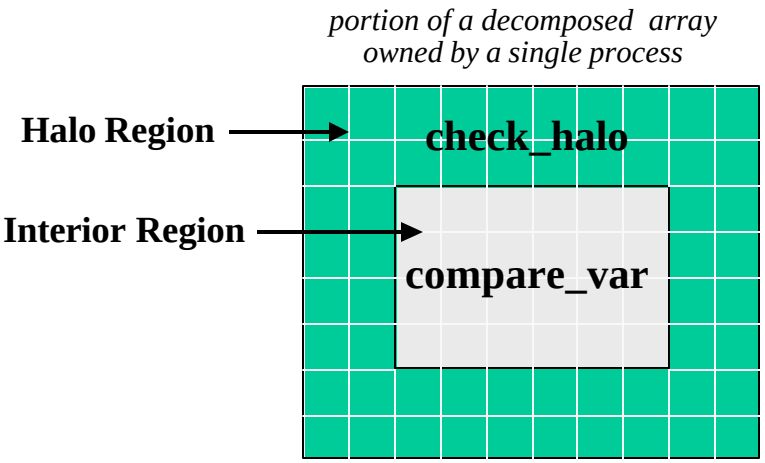


**Figure 8**: An illustration of two debugging directives that are available to verify decomposed array values are correct. Scalars and non-decomposed arrays can also be compared. These directives have greatly simplified debugging and parallel code development.

**Figure 9** illustrates how the CHECK_HALO directive works: halo region values from each user-specified array are compared with their corresponding interior points on the neighboring process. When data values differ, SMS outputs an error message containing the array name, and the location where the problem occurred, and then terminates execution.

# CSMS$CHECK_HALO

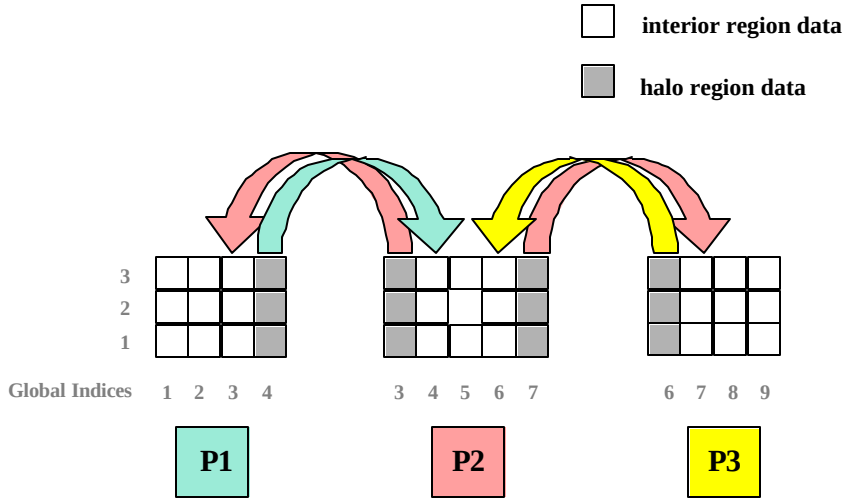


**Figure 9**: This SMS directive is used to verify that each processor's halo region is up to date. In this example, process P2 compares data one step into its left halo (global index 3) with the corresponding interior points on process p1. Similarly, the right halo region points (global index 7) are compared to the interior points on p3. Similar comparisons are made on processors P1 and P3 where appropriate.

The COMPARE_VAR directive, patterned after work by O'Keefe [6], provides the ability to compare array or scalar values between a correctly working code and another run that uses different numbers of processors. For example, the programmer can specify a comparison of the array "x", for a single processor run and for a multiple process run by inserting the directive:

```
csms$compare_var ( x )
```

in the code and then entering appropriate command line arguments to request concurrent execution of the code. Wherever COMPARE_VAR directives appear in the code, user-specified arrays will be compared as shown in **Figure 10**. If differences are detected, SMS will display the name of the variable, the array location (e.g., the i, j, k index) and values from each run, the location in the code, and then terminate execution. Conversely, if no differences are found, SMS will continue executing the code.

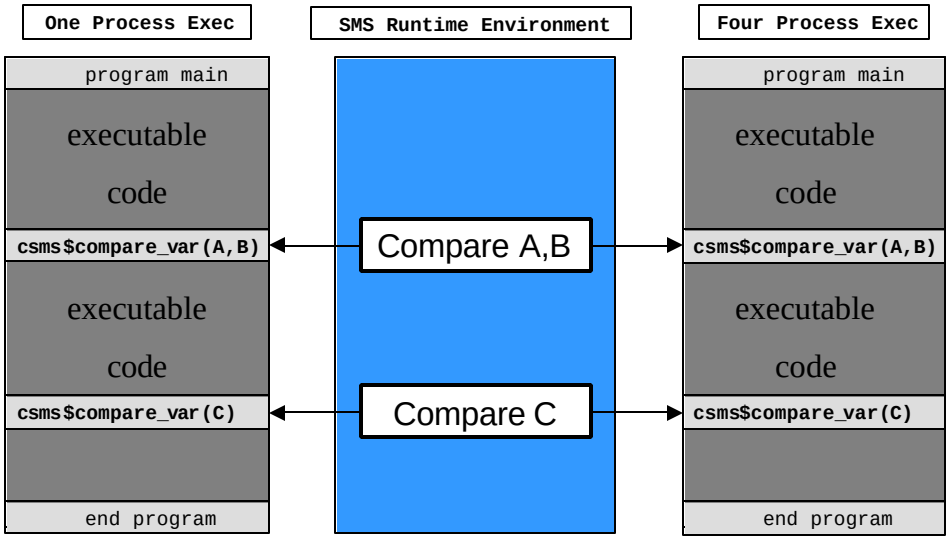


**Figure 10**: An illustration of how COMPARE_VAR is implemented in SMS. In this example, two executables are launched concurrently from the command line. When a COMPARE_VAR directive is encountered, the executables synchronize, and then compare the specified arrays. If any elements of the arrays differ, SMS will print the location and values of the data point and then terminate the execution of the runs.

The ability to compare intermediate model values anywhere in the code has proven to be a powerful debugging tool during code parallelization. For example, the time required to debug a recent code parallelization was reduced from an estimated eight weeks down to two simply because the programmer did not have to spend inordinate amounts of time determining where the parallelization mistakes were made.

These directives have also proven to be a useful way to ensure that model upgrades continue to produce the correct results. For example, a scientist can verify source code changes by simply comparing the output files of the serial "control" run and subsequent parallel runs. In the event results differ, they can turn on SMS debugging (a run-time option) which compares the intermediate results of the arrays specified by COMPARE_VAR. In the event differences appear, they can quickly locate the problem and determine the best solution. In this way, SMS users have found the debugging directives very useful because they allow the code author to control the maintenance and upgrades of their parallel codes rather than requiring the help of a computer specialist.

### *3.10 Optimization*

SMS performance optimizations can be divided into two areas: automatic, and user assisted.

#### *3.10.1 Automatic optimizations*

SMS provides several optimizations that require no action by the user. These optimizations are designed to take advantage of the software and hardware on the target system. For example, on some machines, the more general MPI communications library is replaced by the faster but

less portable SHMEM library. Another optimization provided by SMS are two strategies to minimize the communications time required when TRANSFER operations are done. One strategy minimizes the amount of data that must be moved between processes to reduce communications bandwidth. The other strategy reduces the number of communications that are done in order to minimize communications latency. Finally, the most efficient combination of the variants of MPI send and receive routines are used when SMS is configured and built on the target system. For example, on some machines the mpi_isend/mpi_recv is faster than mpi_isend/mpi_irecv due to how message passing is implemented. Since these optimizations are automatically selected when SMS is built, no changes to the application codes are necessary.

### *3.10.2 User-assisted optimizations*

Several communications optimizations are provided by SMS that permit tuning of the application by the user. These optimizations, controlled via directive, are limited thickness exchanges, aggregation, and redundant computations. Limited thickness exchanges allow the user to minimize the amount of data transferred between neighboring processes. Instead of communicating the entire halo all of the time (the default), communication can be limited to only the portion required. To further reduce communications bandwidth, SMS also allows array sections to be specified. For example:

```
csms$halo_update(x<1:1>, y(:,:,1) )
```

will exchange one point of the halo in the first decomposed dimension of the array *x* and will update the entire halo region, but only for the surface layer (1 in the 3rd dimension) of *y*. Combinations of limited thickness and array section control may also be specified.

A second communications optimization, called aggregation, permits multiple model variables to be combined into a single communications call in order to reduce message passing latency. For example, to update the halos of the variables *a* and *b* together:

```
csms$halo_update(a,b)
```

requires half the communications latency needed to exchange them individually:

```
csms$halo_update(a)
csms$halo_update(b)
```

A third optimization provides the ability to eliminate communications by performing redundant computations in the halo region using the HALO_COMP directive. **Figure 11** and **Figure 12** show how this works. In **Figure 11**, halo updates are required before and after loops 150 and 250. However, in **Figure 12**, HALO_COMP is used to modify the do-loop bounds so computations will be executed one point into the halo region in each direction, as indicated by `<1,1>`. This action eliminates the need for a HALO_UPDATE after loop 150 reducing communications latency. Further, by increasing the halo update on *a* (before loop 150) to two points, HALO_COMP eliminates the update of variables *y* and *z*. This reduces communications bandwidth requirements by 50 percent (from six halo points: *a<1,1>, y<1,1>, z<1,1>* to four: *a<2,2>*).

```
CSMS$HALO_UPDATE( a<1,1>)
  do 150 i= 2, IM-1
         y(i) = a(i) - a(i+1) - a(i-1)
         z(i) = a(i) – (a(i+1) - a(i-1)) * 0.5
  150 continue
CSMS$HALO_UPDATE( y<1,1>,z<1,1> )
       do 250 i= 2, IM-1
          x(i) = y(i)*z(i) + y(i+1)*z(i-1)
               + y(i-1)*z(i+1)
  250 continue
```

**Figure 11:** Sample code where no redundant computations are performed. An exchange of arrays "y" and "z" are required for loop 250 to produce the correct answer on each processor.

```
CSMS$HALO_UPDATE(a<2,2>)
CSMS$HALO_COMP(<1,1>) BEGIN
       do 150 i= 2, IM-1
          y(i) = a(i) - a(i+1) - a(i-1)
          z(i) = a(i) – (a(i+1) - a(i-1)) * 0.5
  150 continue
CSMS$HALO_COMP END
       do 250 i= 2, IM-1
          x(i) = y(i)*z(i) + y(i+1)*z(i-1)
               + y(i-1)*z(i+1)
  250 continue
```

**Figure 12:** A version of the same code that uses redundant computation. Since "y" and "z" are computed one step into the halo region, their halo regions are up to date after loop 150. Consequently, the exchanges of "y" and "z" after loop 150 can be eliminated.

This strategy is a technique that can yield significant performance benefits. In recent tests on an Alpha cluster, this technique boosted performance by 38 percent in one routine of the RUC model that contained communications within dependent cascading loops. A study detailing the performance benefits of this optimization will be presented in a forthcoming performance paper on SMS.

Performance optimizations have also been built into SMS I/O operations. Since scientific models typically output results several times during a model run, these operations can significantly affect the overall performance. By default, SMS uses a server process to overlap output with computations. Output is normally a two step process: the compute processes send their data to the server, and then the server re-assembles these data into the original serial ordering and writes the output data to disk. While the server is working on the output operations, the compute processes resume execution of the code, effectively hiding the cost of output operations during the course of a model run. If output files are large, additional server processes can be added at run-time. Conversely, for runs on small numbers of processes where output performance is not an issue, applications can be run without a server process.

Finally, the user can configure the layout of processors to the problem domain in order, for example, to handle a static load imbalance in the code. By default, SMS uses a pre-determined set of rules to determine how many data points are assigned to each process and how many processes are allocated to each decomposed dimension. To optimize performance, SMS also allows the user to choose the number of data points assigned to each process at run-time.

## 4. Case studies.

The three models parallelized using SMS in these case studies have several similar properties: they were finite difference models, they used static memory allocation and they were coded using a mix of Fortran 77 and Fortran 90 language features. In describing the codes, references for the number of source code lines will exclude comments and blank lines. Additionally, the number of directives added to the code during parallelization will exclude debug directives, and count begin/end pairs as one directive.

Each code runs in parallel and produces the correct results on the Cray T3E, SGI Origin 2000, SGI Origin 3000, IBM SP2, Compaq Alpha and Intel Linux Clusters and SUN workstations. Once an SMS parallelized code runs correctly on one system, it can be quickly ported and run on another computing platform. For example, it took two hours to port the ROMS model, parallelized for Alpha-Linux, to an SGI Origin.

Finally, in lieu of dedicated machine time, all performance tests were run repeatedly to ensure accurate timings. Further investigation of the performance results and benefits of SMS optimizations will be given in a performance related paper that is planned.

### *4.1 Eta model parallelization*

As a high-level software tool, some overhead exists within the support layer of SMS. To show that SMS does not add significant overhead, a comparison was done between the hand-coded MPI based version of the Eta model running operationally at NCEP, and the same Eta model parallelized using SMS. The MPI Eta model was considered a good candidate for fair comparison since it is an operational model optimized, by IBM, for high-performance on the IBM SP2.

The Eta model [26] is a mesoscale weather prediction model run by NCEP to produce daily forecasts for the National Weather Service. The horizontal structure of the dynamics package is a semi-staggered grid known as the Arakawa E-grid [1]. The model is organized into two major packages: dynamics and physics. The dynamics package includes the solution of the momentum and thermodynamics equations (advection, diffusion, coriolis, and pressure tendencies) and management of model boundaries. The physics package includes parameterizations for convection, turbulence, and radiation.

The version of the Eta model FSL used for this performance study has a 32km horizontal resolution (223 by 365) with 45 vertical layers [25]. In January 2000, an MPI version of this model began running operationally on NCEP's IBM SP2 four times per day. Each run produced forecasts out to 48 hours and output files were written every hour. Production runs of this code were discontinued in October 2000 when the model resolution was increased to 22 km; however, the model remains fundamentally unchanged.

To accomplish parallelization, the MPI Eta code was reverse engineered to return the code to its original serial form. Code changes included restoring the original serial loop bounds, removing MPI-based communications routines, and restoring array declarations. This code was then parallelized using SMS. Two-hundred and seventy-nine directives were added to the 19,000 line Eta model during code parallelization.

To improve performance, both codes did redundant computations in the halo, described in Section 3.10. This optimization was used extensively in the codes; 51 HALO_COMP directives were used in the SMS parallelization. Also, both codes took advantage of IBM's parallel I/O capability and therefore did not gather decomposed data before being written to disk. Instead, each processor outputs its section of the decomposed data to separate files.

| Number of Processors | MPI-Eta Time | SMS-Eta Time | SMS faster | SMS-Eta Efficiency |
|---|---|---|---|---|
| 4 | 11197 | 10781 | 4 % | 1.00 |
| 8 | 5317 | 5258 | 1 % | 1.03 |
| 16 | 2878 | 2774 | 4 % | 0.97 |
| 32 | 1471 | 1446 | 2 % | 0.93 |
| 64 | 872 | 820 | 6 % | 0.82 |
| 88 | 694 | 643 | 7 % | 0.76 |

**Table 1**: Eta model performance for MPI-Eta and SMS-Eta run on NCEP's IBM SP-2. Run times are given in seconds for a full 48 hour model run including model initialization and the generation of hourly output files.

Table 1 shows the performance of the Eta model on NCEP's IBM SP2 (375 MHz processor). The results indicate SMS-Eta was faster than MPI-Eta in all cases. In addition, super-linear speedup is noted at 8 processors for both codes, due to better cache utilization since the decomposed arrays fit better in memory. Further analysis of these results indicate that most of the performance gains in SMS-Eta were due to more efficient communications. In particular, the differences were primarily due to the generalized communication aggregation capabilities available in SMS. MPI-Eta only aggregates communication when the arrays are contiguous in memory; SMS has no such restriction. While the amount of data to be communicated per model time step is the same in both models, 87 percent more halo updates are in the MPI-Eta code per model time step than the SMS-Eta (28 vs. 15). On 88 processors of the IBM SP2, the model achieved 11.27 GFLOPS.

Results on FSL's Alpha Linux cluster are shown in Table 2. Once again, the more efficient communications routines in SMS account for the bulk of the faster run times. In a one hour run, the model ran at 277 MFLOPS on a single processor (833 MHz) of FSL's Alpha Cluster or 16.6 percent of the peak performance. On 88 Alpha processors, the model ran at 16.58 GFLOPS.

| Number of Processors | MPI-Eta Time | SMS-Eta Time | % SMS faster | SMS-Eta Efficiency |
|---|---|---|---|---|
| 2 | 16547 | 16435 | 1 % | 1.00 |
| 4 | 7757 | 7706 | 1 % | 1.07 |
| 8 | 3633 | 3575 | 2 % | 1.15 |
| 16 | 1961 | 1911 | 3 % | 1.08 |
| 32 | 978 | 951 | 3 % | 1.08 |
| 64 | 589 | 562 | 5 % | 0.91 |
| 88 | 470 | 437 | 7 % | 0.85 |

**Table 2**: Eta model performance for MPI-Eta and SMS-Eta run on FSL's Linux Cluster. Run times are given in seconds for a full 48 hour model run including model initialization and the generation of hourly output files.

### *4.2 HYCOM model parallelization*

The Hybrid Coordinate Ocean Model [5] is a general circulation ocean model that evolved from the widely used Miami Isopycnic-Coordinate Ocean Model (MICOM) [7]. HYCOM's vertical coordinate structure was upgraded to include terrain following sigma coordinates in shallow water regions, and z coordinates in a weakly stratified upper ocean mixing layer, while retaining MICOM's isopycnic coordinates in the open and stratified ocean areas. HYCOM, like MICOM, is a primitive-equation model containing 5 prognostic equations, two for the horizontal velocity components, a layer thickness tendency, and two conservation equations for a pair of thermodynamic variables such as salt and temperature. HYCOM is the result of collaborative work between the University of Miami, the Los Alamos National Laboratory (LANL), and the Naval Research Laboratory. The model grid size is 135 by 256 with 14 vertical levels.

Two parallel versions of the HYCOM model currently exist: the official version, and the Bleck development version. The development version has been used to test new ideas, such as fully global model grids, which are not supported in the official version. The development version, used in this study, had been parallelized by LANL for shared memory using 115 OpenMP directives. The decomposition strategy was in one dimension (latitude).

For comparison and simplicity, the SMS parallelization of the 4300 line HYCOM model was also done in one dimension. Parallelization of HYCOM took nine days to complete; this included the time required for dependence analysis, insertion of 120 SMS directives, performance tuning and verification of model results. SMS parallelization was simplified because the code contained no apparent serial bugs and it was well structured. In addition, SMS debugging directives were used extensively to find the parallelization errors.

In these tests the model was run for 20 time steps and the serial code achieved 97 MFLOPS or 20 percent of the Origin 2000's 233 MHz processor's peak performance. Initialization and input times were not included in these timings because they would become relatively insignificant for full-length runs. For example, ocean scientists typically run the model for hundreds of thousands of time steps, which correspond to decades or centuries of simulated time. Output times were also ignored since they are also done infrequently.

| Number of Processors | OpenMP Time | SMS Time | SMS Efficiency | % SMS faster |
|---|---|---|---|---|
| 1 | 135.0 | 127.0 | 1.00 | 6% |
| 8 | 21.1 | 14.2 | 1.12 | 48% |
| 16 | 11.9 | 7.5 | 1.06 | 59% |
| 32 | 9.0 | 4.7 | 0.84 | 91% |

**Table 3**: Performance of the HYCOM model on the SGI Origin 2000 system. Run-times for 20 model time steps are given; these results do not include initialization, or I/O. The original serial code ran in 134 seconds.

Several observations can be made about the parallel results shown in **Table 3** First, SMS model performance super-scales out to 16 processors. Beyond that, performance falls off rapidly due to the small global model domain (135 by 256). At 32 processors, for example, there are just eight interior points in the j (latitude) dimension. Inter-process communication of the four (two in each direction) halo points becomes the dominant factor affecting performance. Second, SMS performance is considerably better than the OpenMP performance. This may be due to a less than optimal OpenMP parallelization of the code; the fine-grained approach causes many synchronization points and limits run-time performance. Third, the performance of the serial model is slower than SMS single processor run (134 vs. 127). This may be due to a difference in the way the model handles the periodic boundaries. The OpenMP code accesses the model's periodic boundaries using "mod" function, while the SMS code does not require these computations since these data are replicated in the array's halo region. Finally, a static load imbalance existed in HYCOM due to the presence of land masses in the model domain. Reconfiguring the layout of data to processors, discussed in Section 3, improved SMS model run times by 10 percent.

| Number of Processors | Alpha Cluster Time | Alpha Cluster Efficiency | Cray T3E Time | Cray T3E Efficiency |
|---|---|---|---|---|
| 1 | 37.49 | 1.00 | 152 | 1.00 |
| 2 | 18.65 | 1.05 | 80 | 0.95 |
| 4 | 9.92 | 0.94 | 42 | 0.90 |
| 8 | 5.57 | 0.84 | 23 | 0.82 |
| 16 | 3.48 | 0.67 | 13 | 0.73 |

**Table 4**: Performance of the SMS HYCOM model on FSL's Alpha-Linux cluster and Cray T3E. Run-times are for 20 time steps and do not include model initialization or I/O. The serial code ran in 38.0 seconds on the Alpha Cluster and 161 seconds on the T3E.

The SMS performance of HYCOM on the Alpha cluster and Cray T3E is shown in **Table 4**. The run-times for the Alpha were significantly faster (37 vs. 135 seconds) than the Origin 2000 due to the faster processor (833 MHz vs. 233 MHz). While the processor speeds are much higher, the inter-processor bandwidth of the Alpha cluster is only slightly faster than the Origin and the Alpha latency is higher. This accounts for the decreased scaling since, relative to processor speed, the inter-process communication time becomes a larger factor in overall run

times. Cray T3E run times (600 MHz processor) do not see super-scaling demonstrated on the Origin and Alpha systems, which is likely due to the use of Cray "streams", a hardware data pre-fetch mechanism.

*4.3 ROMS model parallelization*

The Regional Ocean Modeling System (ROMS) [17] is a hydrostatic, primitive equation model, developed jointly by the Rutgers University and UCLA, that is used by modeling groups around the world. The model uses terrain following coordinates in the vertical and curvilinear coordinates in the horizontal. The hydrostatic primitive equations for momentum are solved using a split-explicit time-stepping scheme that requires a coupling between the barotropic (fast) modes and the baroclinic (slow) modes.

The model was previously parallelized for shared-memory architectures using a coarse-grained parallelization paradigm via horizontal data domain decomposition. The decomposed pieces (tiles) are assigned to parallel threads using 191 SGI directives, or optionally OpenMP directives. Redundant computations are done to reduce the synchronization points in the code. An MPI version of the code exists at UCLA; however, it has diverged from the community version maintained by Rutgers and therefore not suitable for a direct performance comparison.

To leverage the existing shared memory parallelism, the ROMS code was converted to dynamic memory. One-hundred twenty directives were added to the 13,000 line ROMS code. For example, PARALLEL directives were not necessary since the loop bounds are already specified in terms of tile starts and ends. The model also used redundant computations to reduce the number of HALO_UPDATE directives required, which improved performance.

The OpenMP and SMS performance results, shown in **Table 5**, are for a coastal Gulf of Alaska scenario using a model resolution of 128x128x30. The model was run for 20 time steps on the SGI Origin 3000 and a serial run achieved 159 MFLOPS or 20 percent of the 400 MHz processor's peak performance. Timings are for the main model loop excluding I/O. These results indicate SMS was slower when small numbers of processors were used, but faster when the number of processors increased. Both one and two dimensional data decompositions were used and the fastest run times are shown for each case. In general, the fastest SMS run times were observed when data were decomposed in two dimensions, since communications scale better; whereas OpenMP did better when a one dimensional decomposition was used. This observation is currently being investigated. In addition, the SMS one processor run was slower than the serial code because the SMS version used dynamic memory.

| Number of Processors | OpenMP Time | SMS Time | SMS Efficiency | % SMS Faster |
|---|---|---|---|---|
| 1 | 143.9 | 148.5 | 1.00 | -3.1 |
| 2 | 70.0 | 74.4 | 1.00 | -5.9 |
| 4 | 36.8 | 39.4 | 0.94 | -6.5 |
| 8 | 19.6 | 20.7 | 0.90 | -5.6 |
| 16 | 10.9 | 11.1 | 0.84 | -1.8 |
| 32 | 6.9 | 6.5 | 0.71 | 5.8 |

**Table 5**: A comparison of OpenMP and SMS runs of the ROMS model on the SGI Origin 3000. The serial code ran in 144.7 seconds.

## 5. Conclusion and future work

### *5.1 Conclusion*

A high-level directive-based tool called SMS has been developed that simplifies and speeds the parallelization of Fortran codes. While the tool has been tailored toward finite difference approximation and spectral weather and ocean models, the approach is sufficiently general to be applied to other structured grid codes. SMS directives, which appear as comments, allow for complete retention of the original serial code. When a parallel version of the code is required, a component of SMS called PPP is used to translate the directives and serial code into portable parallel code. Directives encapsulate most of the low level parallelization details required to accomplish complicated operations such as inter-process communication, process synchronization, gather and scatter operations, and work re-distribution.

SMS provides a simple, flexible user interface with a variety of code generation and run-time options available to the user. In addition, a number of advanced capabilities were described that permit incremental parallelization, and simplify the debugging and maintenance of parallel codes. These advanced capabilities have led to dramatically decreased code parallelization times which can now be measured in days or weeks rather than months of effort. For example, in Section 4, we presented a case study of the HYCOM model which took an SMS programmer only nine days to parallelize.

We believe SMS provides flexible high-performance portable solutions that are competitive with hand-coded vendor specific solutions. We compared the SMS performance of the HYCOM and ROMS models to their OpenMP and native SGI counterparts. In these tests, SMS HYCOM performed significantly better than the OpenMP version, convincing LANL to use the SMS version for their future work. The SMS version of the ROMS model offered equivalent performance to the SGI parallel version on the Origin system but also runs on other distributed memory machines.

We also demonstrated in the parallelization of NCEP's Eta model that the SMS solution performs as well as the MPI-based operational version of the code. Since both models are fundamentally MPI-based codes, the MPI-Eta could, with sufficient effort, be made as or more efficient than SMS-Eta. However, we believe SMS makes it easier to achieve good performance.

*5.2 Future work*

SMS has been used to parallelize many atmospheric and oceanic codes, but some limitations exist. First, the translation tool does not support all of the Fortran 90 language constructs; work is continuing to remove this shortcoming. Second, the application area where SMS provides good support and performance is fairly narrow. Additional work to enhance SMS support for a wider range of applications is ongoing. For example, we plan to provide the capability to support applications with non-uniform grids. We also plan to demonstrate support for the coupling of atmospheric and oceanic models.

To support modern Fortran 90 codes that utilize object-oriented design concepts, we plan to provide the capability to associate data decompositions with user-defined types. This will allow decompositions to be defined with limited scope, and used in the context of the objects that require them. For example, this would permit different grid objects and their corresponding data decompositions to be passed into dynamical core routines.

Another enhancement would be to have the SMS translator generate OpenMP code. Further, for state-the-of-art machines that consist of clusters of SMPs, a parallel code that implements tasking "within the box" using OpenMP and message passing "between the boxes" using MPI may be optimal. The SMS translator could be designed to generate both message passing and micro tasking parallel code. Recent tests have demonstrated that SMS directives can be used in conjunction with OpenMP but more work needs to be done to verify the appropriateness and performance benefits of a hybrid programming approach.

We also plan to further reduce the dependence analysis time and to decrease the number of modifications required to parallelize applications. Recent tests have shown that we can reduce the number of directives required by 30 to 50 percent. For example, the use of directives such as PARALLEL, GLOBAL_INDEX, and TO_GLOBAL could be reduced or eliminated in many codes by increasing the analysis capabilities of the translator.

To simplify dependence analysis, development has begun on a tool called *autogen*, to analyze the serial code and automatically insert some SMS directives. For example, the CHECK_HALO directive could be inserted into the code to document adjacent dependencies and simplify code analysis. A*utogen* could also be used to automatically insert DISTRIBUTE directives to identify decomposed arrays and add COMPARE_VAR directives for debugging. However, one limitation of *autogen* is that it does not provide inter-procedural analysis of the code. Therefore, we would like to combine SMS code translation capabilities with a semi-automatic dependence analysis tool. This tool would be used to analyze the code and insert SMS directives into the serial code, from which a parallel version of the code could be generated and run using SMS.

## 6. **References**


[1] A.Arakawa and V.Lamb, Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model., Methods Comput Phys 17 (1977) 173-265.

[2] C.Baillie, A.E.MacDonald and J.L.Lee, QNH: A Numerical Weather Prediction Model developed for MPPs. International Conference HPCN Challenges in Telecomp and Telecom: Parallel Simulation of Complex Systems and Large Scale Applications. Delft, The Netherlands (1996).

[3] V.Balaji, Parallel Numerical Kernels for Climate Models, Proceeds of the 9th ECMWF Workshop on the Use of High-performance Computing in Meteorology, (2001) 277-295.
[4] S.Benjamin, J.Brown , K.Brundage, D.Dévényi, G.Grell, D.Kim, B.Schwartz, T.Smirnova, T.Smith, S.Weygandt and G.Manikin, RUC 20 – The 20-km version of the Rapid Update Cycle, National Weather Service Technical Procedures Bulletin No. 490 (2002), http://ruc.fsl.noaa.gov/ppp_pres/RUC20-tpb.pdf .
[5] R.Bleck, An Oceanic General Circulation Model Framed in Hybrid Isopycnic-Cartesian Coordinates. Submitted to J. Ocean Modeling *(2001)*.
[6] R.Bleck, S.Dean, M.O'Keefe and A.Sawdey, A comparison of data-parallel and message passing versions of the Miami Isopycnic Coordinate Ocean Model (MICOM), Parallel Computing, 21 (1995).
[7] R.Bleck, C.Rooth, D.Hu, and L.Smith, Salinity-driven thermocline transients in a wind and thermohaline-forced isopycnic coordinate model of the North Atlantic., J. Phys. Oceanogr. 22 (1992) 1486-1505.
[8] A.F.Blumberg and G. L. Mellor, A description of a three-dimensional coastal ocean circulation model, Three-Dimensional Coastal ocean Models, edited by N. Heaps, 208 pp., American Geophysical Union (1987).
[9] D.S.Chen, K.N.Huang, T.C.Yeh, M.S.Peng , and S.W.Chang, Recent improvements of the typhoon forecast system in Taiwan. 23th Conference on Hurricanes and Tropical Meterology. Dallas, TX., (2000) 823-825.
[10] Earth System Modeling Framework Development Team, http://www.esmf.ucar.edu/ .
[11] J.Edwards, J.Snook, and Z.Christidis, Forecasting for the 1996 Summer Olympic Games with the NNT-RAMS Parallel Model, 13th International Information and Interactive Systems for Meteorology, Oceanography and Hydrology, Long Beach, CA., American Meteorological Society, (1997) 19-21.
[12] M.Frumkin, H.Jin, and J.Yan, Implementation of NAS Parallel Benchmarks in High-performance FORTRAN, NAS Technical Report NAS-98-009, NASA Ames Research Center, Moffett Field, CA (1999) http://ipdps.eece.unm.edu/1999/papers/114.pdf.
[13] M.Frumkin, H.Jin, A.Waheed, J.Yan, A Comparison of Automatic Parallelization Tools / Compilers on the SGI Origin 2000. Proceedings of Super Computing '98, Orlando, Florida, (1998) http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Hribar1140/.
[14] M.Govett, J.Edwards, L.Hart, T.Henderson, and D.Schaffer, SMS Reference Manual, (2001) http://www-ad.fsl.noaa.gov/ac/SMS_ReferenceGuide.pdf.
[15] R.Gray, V.Heuring, S.Levi, A.Sloane, and W.Waite, Eli, A Flexible Compiler Construction System, Communications of the ACM 35 (1992) 121-131.
[16] W.Gropp, E.Lusk, and A.Skjellum, Using MPI, Portable Parallel Programming with the Message Passing Interface. MIT Press (1994).
[17] D.B.Haidvogel, H.G.Arango, K.Hedstrom, A.Beckman, P.Malanotte-Rizzoli, and A.F Shchepetkin, Model Evaluation Experiments in the North Atlantic Basin: Simulations in Nonlinear Terrain-Following Coordinates, Dyn. Atmos. Oceans 32 (2000) 239-281.
[18] T.Henderson, C.Baillie, S.Benjamin, T.Black, R.Bleck, G.Carr, L.Hart, M.Govett, A.Marroquin, J.Middlecoff and B.Rodriguez, Progress Toward Demonstrating Operational Capability of Massively Parallel Processors at Forecast Systems Laboratory, Proceedings of the Sixth *ECMWF Workshop on the Use of Parallel Processors in Meteorology, European Centre for Medium Range Weather Forecasts*, Reading, England (1994).

[19] T.Henderson, D.Schaffer, M.Govett, and L.Hart, SMS User's Guide, http://www-ad.fsl.noaa.gov/ac/SMS_UsersGuide.pdf (2002).
[20] C.S.Ierotheou, S.P.Johnson, M.Cross, and P.F. Leggett, 1996: Computer aided parallelization tools (CAPTools) - Conceptual Overview and Performance on the Parallelization of Structured Mesh Codes, *Parallel Computing* 22 (1996) 163-195.
[21] C.Koelbel, D.Loverman, R.Shreiber, G.Steele Jr., and M.Zosel, The High-performance Fortran Handbook. MIT Press *(1994)*.
[22] S.Kothari and Y.Kim, Parallel Agent for Atmospheric Models, Proceedings of the Symposium on Regional Weather Prediction on Parallel Computing Environments (1997) 287-294.
[23] C.S.Liou, J.Chen, C.Terng, F.Wang, C.Fong, T.Rosmond, H.Kuo, C.Shiao, and M. Cheng, The Second-Generation Global Forecast System at the Central Weather Bureau in Taiwan, Weather and Forecasting 12 (1997) 653-663.
[24] A.E.MacDonald, J.L.Lee, and Y.Xie, QNH: Design and Test of a Quasi Non-hydrostatic Model for Mesoscale Weather Prediction. Monthly Weather Review 128 (2000) 1016-1036.
[25] G.Manikin, M.Baldwin, W.Collins, J.Gerrity, D.Keyser, Y.Lin, K.Mitchell, and E.Rodgers: Changes to the NCEP Eta Runs: Extended Range, added Input, added Output, Convective Changes, National Centers for Environmental Prediction Technical Procedures Bulletin (2000) http://www.nws.noaa.gov/om/tpb/465.htm.
[26] F.Mesinger, The Eta Regional Model and its Performance at the U.S. National Centers for Environmental Prediction. International Workshop on Limited-area and Variable Resolution Models. Beijing, China, WMO/TD 699 (1995) 42-51.
[27] J.Michalakes, RSL: A Parallel Runtime System Library for Regular Grid Finite Difference Models using Multiple Nests, Tech. Rep. ANL/MCS-TM-197, Argonne National Laboratory, (1994).
[28] J.Michalakes, FLIC: A Translator for Same-Source Parallel Implementation of Regular Grid Applications, Tech. Rep. ANL/MCS-TM-223, Argonne National Laboratory, (1997).
[29] J.Michalakes, J.Dudhia, D.Gill, J.Klemp and W.Shamarock, Design of a Next Generation Regional Weather Research and Forecast Model. Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology, European Centre for Medium Range Weather Forecasts (1998), Reading, England.
[30] T.Ngo, L.Snyder, and B.Chamberlain, Portable Performance of Data Parallel Languages. Supercomputing 97 Conference, San Jose, CA (1997).
[31] R.Numrich, and K.Reid, Co-Array Fortran for Parallel Programming. ACM Fortran Forum, **17**, no 2, (1998) 1-31.
[32] Program for Integrated Earth System Modeling (PRISM) Web Page: http://prism.hnes.org/ .
[33] Portland Group, Parallel Fortran for HP Systems, (1999) http://www.npac.syr.edu/hpfa/bibl.html.
[34] R.K.Rew and G.P.Davis, Unidata's netCDF Interface for Scientific Data Access, Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Anaheim, CA (1990).
[35] B.Rodriguez, L.Hart and T.Henderson, Parallelizing Operational Weather Forecast Models for Portable and Fast Execution, Journal of Parallel and Distributed Computing, 37 (1996) 159-170.