

# **SANDIA REPORT**

SAND2012-10431  
Unlimited Release  
Printed 2012

## **MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing; Version 1.0**

Richard F. Barrett, Courtenay T. Vaughan, and Michael A. Heroux

Center for Computing Research  
Sandia National Laboratories  
P.O. Box 5800  
Albuquerque, NM 87185-1319

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



**Sandia National Laboratories**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
U.S. Department of Energy  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Telephone: (865) 576-8401  
Facsimile: (865) 576-5728  
E-Mail: [reports@adonis.osti.gov](mailto:reports@adonis.osti.gov)  
Online ordering: <http://www.osti.gov/bridge>

Available to the public from  
U.S. Department of Commerce  
National Technical Information Service  
5285 Port Royal Rd  
Springfield, VA 22161

Telephone: (800) 553-6847  
Facsimile: (703) 605-6900  
E-Mail: [orders@ntis.fedworld.gov](mailto:orders@ntis.fedworld.gov)  
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



# MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing; Version 1.0

A broad range of scientific computation involves the use of difference stencils. In a parallel computing environment, this computation is typically implemented by decomposing the spacial domain, inducing a “halo exchange” of process-owned boundary data. This approach adheres to the Bulk Synchronous Parallel (BSP) model. Because commonly available architectures provide strong inter-node bandwidth relative to latency costs, many codes “bulk up” these messages by aggregating data into a message as a means of reducing the number of messages. A renewed focus on non-traditional architectures and architecture features provides new opportunities for exploring alternatives to this programming approach.

In this report we describe miniGhost, a “miniapp” designed for exploration of the capabilities of current as well as emerging and future architectures within the context of these stencil-based applications. MiniGhost joins the suite of miniapps developed as part of the Mantevo project, <http://mantevo.org>.

# Acknowledgment

## Acknowledgment

The authors thank the DOE ASC program for funding this research.

# Contents

<b>Executive Summary</b>	<b>9</b>
<b>1 Introduction</b>	<b>11</b>
<b>2 miniGhost</b>	<b>13</b>
2.1 Implementations .....	13
2.2 Execution and Verifying Correctness .....	16
2.3 Output .....	18
2.4 Code description .....	18
2.5 Parallel Programming Model .....	21
2.6 Peer implementations .....	22
2.7 Checkpointing .....	22
<b>3 Summary</b>	<b>25</b>
<b>References</b>	<b>26</b>
<b>Appendix</b>	
<b>A Code categorization</b>	<b>29</b>

# List of Figures

1.1	Stencil inter-process communication requirement . . . . .	11
2.1	Five point stencil in Fortran . . . . .	14
2.2	MiniGhost boundary exchange and computation . . . . .	15
2.3	miniGhost code flow diagram . . . . .	19
2.4	Sketch of miniGhost boundary exchange . . . . .	22
A.1	miniGhost SLOC summary by file . . . . .	31
A.2	miniGhost BSPMA SLOC summary by file . . . . .	32

# List of Tables

2.1	Input parameters .....	17
2.2	MiniGhost functionality .....	20
2.3	MPI functionality employed .....	21
2.4	MPI-IO functionality employed for checkpointing .....	23





# Executive Summary

A broad range of scientific computation involves the use of difference stencils. In a parallel computing environment, this computation is typically implemented by decomposing the spacial domain, inducing a “halo exchange” of process-owned boundary data. This approach adheres to the Bulk Synchronous Parallel (BSP) model. Because commonly available architectures provide strong inter-node bandwidth relative to latency costs, many codes “bulk up” these messages by aggregating data into a message as a means of reducing the number of messages. A renewed focus on non-traditional architectures and architecture features provides new opportunities for exploring alternatives to this programming approach.

In this report we describe miniGhost, a “miniapp” designed for exploration of the capabilities of current as well as emerging and future architectures within the context of these stencil-based applications. MiniGhost joins the suite of miniapps developed as part of the Mantevo project, <http://mantevo.org>.

Our work is motivated by recent experiences with new node interconnect architectures, including that used by Cielo [11, 17, 20].

Various experiments involving miniGhost have already been completed, supporting the value of miniGhost as a proxy for represented applications. Additional experiments have been defined, with implementations underway. Future reports will describe the outcome of those experiments, and will also include the definition and use of a performance model incorporating many issues defined by the DOE exascale codesign efforts [3, 12, 18].

This document is released in support of the initial release of miniGhost. As miniapps are intended to be modified, additional supporting documentation is expected to be produced.



# Chapter 1

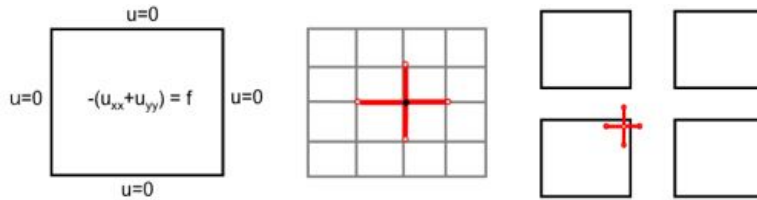
## Introduction

A broad range of physical phenomena in science and engineering can be described mathematically using partial differential equations. Determining the solution of these equations on computers is commonly accomplished by mapping the continuous equation to a discrete representation. One such solution technique is the finite differencing method, which lets us solve the equation using a difference stencil, updating the grid as a function of each point and its neighbors, presuming some discrete time step. The algorithmic structure of the finite difference method maps naturally to the parallel processing architecture and single-program multiple-data (SPMD) programming model. For example, on a regular, structured grid,  $O(n^2)$  computation is performed, with nearest neighbor  $O(n)$  inter-process communication requirements.

On parallel processing architectures, these stencil computations require data from neighboring processes. Inter-process communication is typically abstracted into some sort of functionality that may be loosely described as *boundary exchange* (likewise also called ghost-exchange or halo-exchange). This notion of mapping a continuous problem to discrete space and the inter-process communication requirement induced by spatially decomposing the grid across parallel processes is illustrated in Figure 1.1.

**Figure 1.1.** Stencil inter-process communication requirement

*The left figure shows a partial differential equation (the Poisson equation) described on a continuous domain, with homogeneous Dirichlet boundary conditions. The discretization of this problem is shown in the figure on the middle. The right figure illustrates the inter-process communication requirements when the discretized domain is decomposed across four parallel processes.*



This approach adheres to the bulk-synchronous parallel programming model (BSP [19]),

arguably the dominant model for implementing high performance portable parallel processing scientific applications. As widely available parallel processing architectures focused node interconnect performance on bandwidth (relative to latency), code developers often aggregated data from various structures into single messages [4]. Although many such applications have continued to perform well even up to peta-scale [2, 7], the situation appears to be changing with the push to exascale [1, 18].

In the BSP/message aggregation (BSPMA) model, data from multiple (logical) memory locations are combined into a user-managed array with other data, then subsequently transmitted to the target process. This step incurs three additional costs, none of which directly advances the computation: memory utilization (the message buffers), on-node bandwidth (copies into the buffer), and synchronization (leading up to and including the data transfer). Further, this model interferes in some sense with the natural mapping of algorithms to programming languages in that the code developer must organize computation with the intent to aggregate and exchange data as a means of maximizing bandwidth and avoiding latency rather than organizing computation in a manner natural to the algorithm.

A renewed focus on non-traditional architectures and architecture features provides new opportunities for exploring alternatives to this programming approach. In previous work [17, 20] we saw that codes configured for the BSPMA model realized an evolutionary improvement in performance. However, codes that sent a large number of small messages realized a significant improvement in performance. This improvement is attributable to the significantly increased message injection rate of the node interconnect and supported by the node architecture, a trend we see continuing as nodes become more powerful and complex.

In order to study the performance characteristics of the BSPMA configuration within the context of computations widely used across a variety of scientific algorithms, we have developed a “miniapp”, called miniGhost. As a miniapplication [13], miniGhost is designed for modification and experimentation. It is an open source, self-contained, stand-alone code, with a simple build and execution system. It creates an application-specific context for experimentation, allowing investigation of different programming models and mechanisms, existing, emerging, and future architectures, and enabling investigation of entirely new algorithmic approaches for achieving effective use of the computing environment within the context of complex application requirements.

We begin with a discussion of difference stencils, followed by a description of the miniGhost miniapp, focusing on its computation and communication requirements. We include a discussion of the implementation, with a description of some MPI semantic options for implementing the boundary exchange. We then describe CTH, an application code for which miniGhost is intended to serve as a proxy, including a listing of the intentional differences between miniGhost and the implementation of CTH. Next we present some runtime results that serve to support the claim that intended connection. We conclude with a summary of this initial work and a discussion of future work.

# Chapter 2

## miniGhost

Stencil computations form the basis for finite difference, finite volume, and in fact many other algorithms. The basic idea is to update a value as an average of that value and some set of neighboring points. In the simplest case, heat diffusing across a homogeneous two dimensional domain is modeled as the non-weighted averages of the points surrounding the point to be updated. This can be described using a 5-point stencil defined as

$$u_{i,j,k}^{t+1} = \frac{u_{i,j-1,k}^t + u_{i-1,j,k}^t + u_{i,j,k}^t + u_{i+1,j,k}^t + u_{i,j+1,k}^t}{5}, \text{ for } i, j, k = 1, \dots, n, \text{ for timestep } t.$$

A 9-point stencil would include the (up to four) points diagonally adjacent to

$$u_{i,j,k}^t : u_{i-1,j-1,k}^t, u_{i-1,j+1,k}^t, u_{i+1,j-1,k}^t, \text{ and } u_{i+1,j+1,k}^t.$$

A three dimensional domain might need to include neighbors in adjacent two dimensional “slices”, creating 7-point (analogous to 5-point) stencil or 27-point (analogous to a 9-point) stencil.

A Fortran implementation of the 5-point stencil shown in Figure 2.1 with the notion of decomposing across parallel processes illustrated above in Figure 1.1.

This problem definition presumes regular, equally spaced grid points across the global domain. This greatly simplifies the implementation of the algorithm, allowing us to focus in on the performance aspects of interest in our experiments.

## 2.1 Implementations

The basis of miniGhost is the BSPMA implementation described above. Since miniGhost was originally developed to explore alternative message passing implementations, we include a variant<sup>1</sup>, which also uses MPI for parallelization. The three options are:

---

<sup>1</sup>Future plans include additional variants.

**Figure 2.1.** Five point stencil in Fortran

*This codesegment implements a five point differencing scheme on a three dimensional (  $NX \times NY \times NZ$  ) grid. Note the extra (ghost) space allocated for the boundary condition.*

```
REAL, DIMENSION(0:NX+1,0:NY+1,0:NZ+1) :: X, Y
DO K = 1,NZ
  DO J = 1,NY
    DO I = 1,NX
      X(I,J,K) =
        ( Y(I-1,J,K)+
          Y(I,J-1,K) + Y(I,J,K) + Y(I,J+1,K) +
            Y(I+1,J,K) )
        / 5.0
    END DO
  END DO
END DO
```

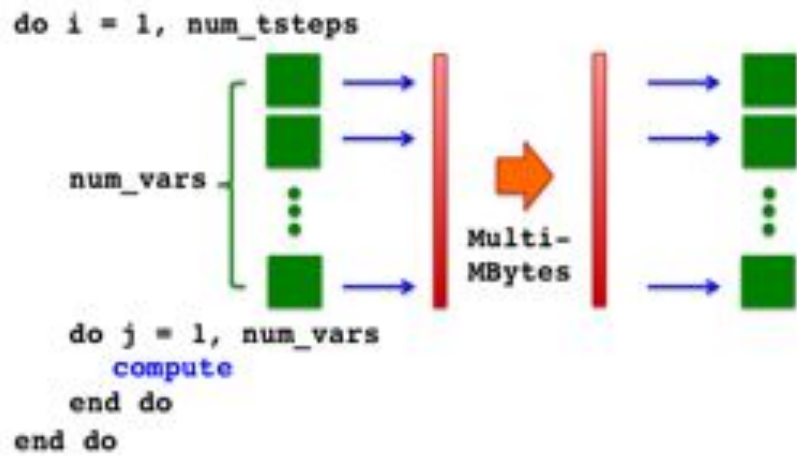
**Bulk synchronous parallel with message aggregation (BSPMA)** Face data is accumulated from each variable into user managed buffers. The buffers are then transmitted to (up to) six neighbor processes, and computation of the selected stencil is applied to each variable. (This implementation is illustrated in Figure 2.3(a).)

**Single variable, aggregated face data (SVAF)** This version transmits data as soon as computation on a variable is completed, face data aggregated. Thus six messages are transmitted for each variable (up to 40), one to each neighbor, each time step. (Illustrated in Figure 2.3(b), this eliminates the inner `END DO` and `DO I = 1, NUM_VARS` from the BSPMA implementation.)

**Skeleton app** Although not an “official” implementation, by selecting the “no stencil” option (see Table 2.1), miniGhost runs in pure communication mode, based on the above configurations. This could serve as an interconnect stress test.

Optionally, summation of the (grid) elements for each variable may be computed, injecting collective communication into the execution. The MPI collective `MPI_ALLREDUCE` forms the global value, adding a runtime stress point typically seen in codes of this sort.

**Figure 2.2.** MiniGhost boundary exchange and computation



(a) BSPMA



(b) SVAF

## 2.2 Execution and Verifying Correctness

MiniGhost is not configured to solve any particular problem, allowing the user to control running time, by setting the number of time steps executed. The `GRID` arrays are loaded with random values (using the Fortran subroutine `RANDOM_NUMBER`). Because homogeneous Dirichlet boundary conditions are used, the grid values will eventually become zeros, so randomly generated source terms (called *spikes*) can be applied in order to maintain non-zero computation. Each spike will induce the requested number of time steps to be performed. That is, if 10 spikes and 50 time steps are requested, each spike will be inserted every 50 time steps, resulting in  $50 \times 10 = 500$  total time steps.

The reference version of a Mantevo miniapp may execute serially, or with parallel processes using MPI, optionally including OpenMP threads. In serial mode using the default settings, miniGhost is run as

```
% ./miniGhost.x
```

In MPI mode using the default settings, miniGhost is run as

```
% mpirun -np 1024 ./miniGhost.x
```

When using OpenMP with MPI, the number of threads per MPI rank is set using environment variable `OMP_NUM_THREADS`.

Runtime input parameters are listed in Table 2.1, and may also be listed using runtime input `--help`, i.e.

```
% ./miniGhost.x --help
```

or

```
% mpirun -np 1024 ./miniGhost.x --help
```

Correctness is ensured by comparing the current state (the sum of the global domain values), added to sum of the flux out of the domain, with the initial values. That is, the sum of each `GRID` array should be equal to the inserted source term (within some specified tolerance). The current implementation uses a scaled error check:

$$\frac{\text{SOURCE\_TOTAL}(\text{IVAR}) - \text{GRIDSUM}_{\text{IVAR}}}{\text{SOURCE\_TOTAL}(\text{IVAR})} < \text{TOL}, \text{ for } \text{IVAR} = 1, \dots, \text{NUM\_VARS}.$$



**Table 2.1.** Input parameters

*\*default setting; See MG\_OPTIONS.F for list of all parameterized options.*

<i>Parameter</i>	<i>Description</i>	<i>Options</i>
<code>--scaling</code>	Parallel scaling configuration	SCALING_STRONG SCALING_WEAK*
<code>--comm_method</code>	Boundary exchange implementation.	COMM_METHOD_BSPMA* COMM_METHOD_SVAF
<code>--stencil</code>	Stencil to be applied.	STENCIL_NONE STENCIL_2D5PT STENCIL_2D9PT STENCIL_3D7PT STENCIL_3D27PT*
<code>--nx --ny --nz</code> <i>or</i> <code>--ndim for nx = ny = nz</code>	Grid dimension in $(x, y, z)$ directions. Global values if strong scaling, local values if weak scaling.	$> 0$ ; $10^*$
<code>--num_vars</code>	Number of GRID arrays operated on.	$1 - 40^*$
<code>--percent_sum</code>	(Approximate) percentage of variables summation reduced	$0-100$ ; $0^*$
<code>--num_tsteps</code>	Number of time steps iterated.	$> 0$ ; $10^*$
<code>--num_spikes</code>	Number of source spikes inserted.	$> 0$ ; $1^*$
<code>--npx --npy --npz</code> <i>or</i> <code>--npdim for npx = npy = npz</code>	Logical processor grid in $(x, y, z)$ .	$> 0$ ; $(numpes, 1, 1)^*$
<code>--error_tol</code>	Error tolerance.	$10^{-error\_tol}$
<code>--report_diffusion</code>	Write error to <code>stdout</code> every $n$ time steps.	$n \geq 0^*$
<code>--debug_grid</code>	Initialize grids to 0, insert heat source in center.	0 or $1^*$
<code>--report_perf</code>	Reporting options	$0^*, 1, 2$
<code>--help</code>	Lists input parameters, and aborts.	

The default error tolerance is  $10^{-8}$  for `REAL8` and  $10^{-4}$  for `REAL4`. Note that all variables are checked, each requiring a global summation, which can significantly impact execution time.

A special problem is configured to enable easier tracking of the diffusion of values across the domain. Setting the runtime parameter `grid_debug` to 1 initializes the `GRID` arrays to 0, inserts a source term in the middle of the global domains, and then tracks the sources as they propagate throughout the arrays.

## 2.3 Output

Output is controlled by the command line option `report_perf`. By default it is set to 0, resulting in the problem configuration and performance results written to a file named `result.yaml`, formatted using `YAML`<sup>2</sup>. By setting this option to 1, this information is also written to a text file named `result.txt`. Setting it to 2 adds per processor communication times to the `result.txt` file.

## 2.4 Code description

MiniGhost is constructed using a modular design, illustrated in Figure 2.3. In particular, the separation of the stencil computation and boundary exchange communication enables experimentation in a variety of ways. For example, new stencils, adding weights to the stencils, or alternative MPI functionality could be configured. Significantly different implementations of the required functionality can also be configured, with some examples described in this report’s summary, Chapter 3.

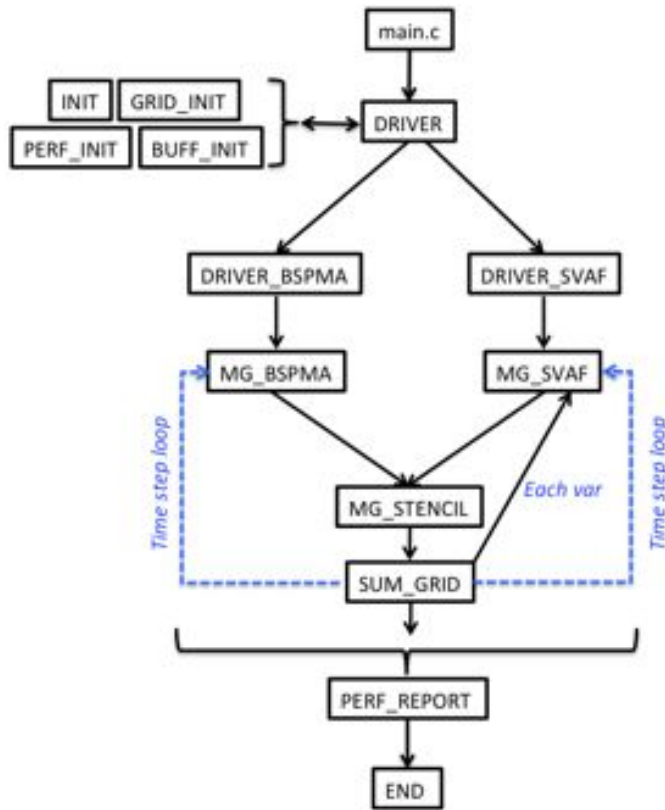
MiniGhost is (mostly) implemented using the Fortran programming language<sup>3</sup>, requiring at least a Fortran 90 compliant compiler. Parallelism, described in Section 2.5, is enabled using functionality defined by the MPI specification [15]. Each variable (representing for example a material state) is stored in a distinct three dimensional Fortran array (named `GRIDx`, for  $x = 1, \dots, 40$ ), across which the stencil is computed using a triply nested `DO` loop. Type precision is configurable as either single (four bytes) or double (eight bytes, the default), managed in module `MG_CONSTANTS`. Pre-processor compiler directives manage the interface with MPI functionality. That is, `MG_MPI_REAL` is set to `MPI_REAL4` or `MPI_REAL8`, depending on the precision requested. Most other variables are declared using the default `INTEGER` or `REAL`, unless otherwise required or recommended. For example, timings are determined at double precision, using `MPI_Wtime` under MPI and the Fortran function `SYSTEM_CLOCK` for serial execution. Accumulation of profiling data could require increased precision, so eight

---

<sup>2</sup><http://yaml.org>

<sup>3</sup>The main program is configured using the C programming language, which enables more flexible parsing of command line input.

Figure 2.3 shows the runtime flow. For the most part, the names refer to the subrou-



**Table 2.2.** MiniGhost functionality

<i>Functionality</i>	<i>Function</i>	<i>Alternatives</i>
	MINI_GHOST MG_BUFINIT MG_CONSTANTS MG_OPTIONS MG_PROFILING MG_UTILS	
Boundary exchange driver	DRIVER_BSPMA	DRIVER_SVAF
Boundary exchange	MG_BSPMA MG_BSPMA_DIAGS	MG_SVAF MG_SVAF_DIAGS
Driver for stencil option y-point stencil computation in x dimensions	MG_STENCIL MG_STENCIL_xDyPT	
Manages the reduction (sum) across a grid Performs the reduction (sum) across a grid	MG_SUM_GRID MG_ALLREDUCE	
Post non-blocking receives	MG_Irecv	
Pack face into message buffer	MG_PACK	
Send boundaries	MG_SEND_BSPMA MG_SEND_SVAF	
Message completion	MG_UNPACK_BSPMA	MG_UNPACK_SVAF
Unpack message buffer into GRIDs ghost space	MG_GET_FACE	
Captures heat lost to flux. Correctness check functionality.	MG_FLUX_ACCUMULATE	
Driver	main.c	

**Table 2.3.** MPI functionality employed

<i>Subroutine</i>	<i>Use</i>
MPI_IRECV MPI_SEND MPI_WAITANY	Core functionality
MPI_ALLREDUCE MPI_ISEND MPI_RECV	Optional Core functionality
MPI_ABORT MPI_BCAST MPI_COMM_DUP MPI_COMM_RANK MPI_COMM_SIZE MPI_ERRHANDLER_SET MPI_INIT MPI_GATHER MPI_FINALIZE MPI_REDUCE	Support functionality

## 2.5 Parallel Programming Model

MiniGhost is configured using the Single Program Multiple Data (SPMD) parallel programming model, with parallelism enabled using functionality defined in the MPI specification [15]. MPI provides a wealth of mechanisms and configurations for point-to-point interprocess communication. Our choice is motivated by that employed by the widest number of applications in our experience, reinforced by discussions with many MPI implementers. Here, non-blocking receives for all communication partners are posted, followed by all non-blocking sends, followed by completion of these procedures as a whole. Sends are preceded by data copies into message buffers where needed; upon completion, receives are followed by unpacking of data into appropriate data structures where needed. Figure 2.4 illustrates the idea with a code fragment. We anticipate that different configurations might result in meaningfully different (and perhaps better) performance on different platforms with different MPI implementations, an issue we intend to explore as a general study using this and other miniapps.

The OpenMP implementation explicitly enables processor and memory affinity using an explicit first touch algorithm when initializing the `GRIDx` arrays. Parallel loops are enabled in the stencil computations, summation across the `GRIDx` arrays, and the packing and unpacking of the halos.

**Figure 2.4.** Sketch of miniGhost boundary exchange

```
DO I = 1, NUM_RECVS
  CALL MPI_Irecv ( ..., MSG_REQ(I), ... )
END DO

! Perhaps some buffer packing

DO I = 1, NUM_SENDS
  CALL MPI_Isend ( ..., MSG_REQ(I+NUM_RECVS), ... )
END DO

DO I = 1, NUM_RECVS + NUM_SENDS
  CALL MPI_WAITANY ( NUM_RECVS + NUM_SENDS, MSG_REQ, IWHICH, ISTAT, IERR )

  IF ( IWHICH <= NUM_RECVS ) THEN
    ! Perhaps some unbuffer packing
    ! else completed send, no action required.
  END IF
END DO
```

## 2.6 Peer implementations

Mantevo miniapps are designed to serve as a tractable means of describing key performance issues within the context of large scale scientific and engineering application codes. As such, they are purposely written using the most ubiquitous languages (C, C++, Fortran) and parallel programming mechanism (MPI) with an option to use OpenMP [10] within a node, providing what we refer to as the *reference implementation*. This provides a means for exploring alternative, emerging and future architectures. The current distribution includes support for the OpenACC version 1.0 specification<sup>4</sup>. We anticipate implementations based on Fortran co-arrays, as well as alternative and developing programming models and languages, such as Chapel [8] and X10 [9], and perhaps some functional languages. We also anticipate developing an implementation based on the C programming language.

## 2.7 Checkpointing

Checkpointing is a common resilience technique used to recover from program or system failures. A checkpoint contains enough program state to restart execution from the current time step as opposed to starting the run from time step zero. Depending on the checkpoint

---

<sup>4</sup><http://www.openacc-standard.org/>

<i>Subroutine</i>	<i>Use</i>
MPI_FILE_OPEN MPI_FILE_CLOSE	File management
MPI_TYPE_EXTENT MPI_TYPE_GET_EXTENT MPI_TYPE_CONTIGUOUS MPI_TYPE_CREATE_STRUCT MPI_TYPE_CREATE_SUBARRAY MPI_TYPE_COMMIT MPI_TYPE_FREE	Derived type construction
MPI_FILE_SET_VIEW MPI_FILE_WRITE MPI_FILE_WRITE_ALL MPI_FILE_READ MPI_FILE_READ_ALL	Data movement

**Table 2.4.** MPI-IO functionality employed for checkpointing

size and host system performance, checkpointing can be an expensive operation.

MiniGhost includes an MPI-IO based checkpoint module that lets users study checkpoint performance on targeted platforms. (A list of MPI-IO functionality employed is shown in Table 2.4.) At the end of every time step, there is an opportunity to checkpoint the current state of miniGhost. If the checkpoint interval is greater than zero and the current time step matches the interval, a checkpoint is performed. Each checkpoint appends a small header plus the problem variables (`GRIDx`) to the checkpoint file. The first checkpoint has some additional overhead including file creation and the writing of a global header. At the end of each checkpoint, a checkpoint counter in the global header is incremented and the file is closed to ensure a consistent file state.

Checkpoint file I/O is implemented using the parallel I/O API from the MPI 2.2 specification [14]. The miniGhost checkpoint module uses MPI derived datatypes to describe the relationship between the in-memory data representation and the file representation. The simplest datatypes are arrays of integers constructed with `MPI_TYPE_CONTIGUOUS`. These datatypes are used to write the list of problem variables active in this run (`GRIDS_TO_SUM`). The active problem variable list is fixed after startup and common to all the PEs, so the root PE writes the array with `MPI_FILE_WRITE` as part of the global header, while the other PEs are idle. More complicated datatypes for writing the problem variables (`GRIDx`) are constructed using `MPI_TYPE_CREATE_SUBARRAY`. Writing the problem variables requires two derived datatypes. The first datatype (`CP_NOGHOST_TYPE`) describes the local in-memory grid without the ghost cells. Because the ghost cells are copied from other PEs, there is no reason

to save the ghost cells. The second datatype (`CP_TSGRID_TYPE`) is a compound type composed of `CP_NOGHOST_TYPE` elements that describes the distribution of each problem variable across the PEs. Every PE calls `MPI_FILE_WRITE_ALL` to write the entire grid to disk. The combination of these datatypes results in a complete contiguous grid in the checkpoint file without ghost cells.



# Chapter 3

## Summary

MiniGhost is a miniapp developed within the scope of the Mantevo project. It is designed to provide a means to explore the Bulk Synchronous Parallel programming model, supplemented with message aggregation, in the context of exchanging inter-process boundary data typically seen in finite difference and finite volume computations. This programming model is employed across a breadth of science domains, typically for solving partial differential equations. MiniGhost was inspired by the multi-decades experiences by the authors with these sorts of parallel programs, and the desire to explore alternative configurations on current, emerging, and future computing environments. It also provides a means for exploring alternative programming languages as well as alternative semantics of MPI.

An alternative boundary communication strategies are included for the boundary exchange, designed to explore the capabilities of computer node inter-connects. Additionally, collective communication may be inserted throughout the time steps, adding an additional level of realism for many application programs. Further, computation may be “turned off”, providing a skeleton app capability whereby inter-process communication requirements may be used as a “stress test” to explore inter-connect capabilities external of computation.

A methodology for determining how a miniapp is predictive of a full application is presented in [5]. Some results from the use of miniGhost within the context of a full application has been presented in [6].



# References

- [1] S. Ahern, S.R. Alam, M.R. Fahey, R. Hartman-Baker, R.F. Barrett, R. Kendall, D. Kothe, O.E. Messer, R. Mills, R. Sankaran, A. Tharrington, and J.B. White III. Scientific Application Requirements for Leadership Computing at the Exascale. Technical Report TM-2007/238, Oak Ridge National Laboratory, December 2007.
- [2] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. S. Vetter, P. Worley, and W. Yu. Early Evaluation of IBM BlueGene/P. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08*, pages 23:1–23:12, Piscataway, NJ, USA, 2008. IEEE Press.
- [3] K. Alvin, R. Brightwell, and S. Dosanjh et al. On the Path to Exascale. *International Journal of Distributed Systems and Technologies*, 1(2), 2010.
- [4] R.F. Barrett, S. Ahern, M.R. Fahey, R. Hartman-Baker, J.K. Horner, S.W. Poole, and R. Sankaran. A Taxonomy of MPI-Oriented Usage Models in Parallelized Scientific Codes. In *The International Conference on Software Engineering Research and Practice*, 2009.
- [5] R.F. Barrett, P.S. Crozier, S.D. Hammond, M.A. Heroux, P.T. Lin, T.G. Trucano, and C. Vaughan. Summary of Work for ASC L2 Milestone 4465: Characterize the Role of the Mini-Application in Predicting Key Performance Characteristics of Real Applications. Technical Report SAND2012-4667, Sandia National Laboratories, 2012. In preparation.
- [6] R.F. Barrett, S.D. Hammond, C.T. Vaughan, D.W. Doerfler, M.A. Heroux, J.P. Luitjens, and D. Roweth. Navigating An Evolutionary Fast Path to Exascale. In *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS12)*, 2012.
- [7] B. Bland. Jaguar: The World’s Most Powerful Computer System. In *The 52nd Cray User Group meeting*, 2010.
- [8] B.L. Chamberlain, D.Callahan, and H.P. Zima. Parallel Programming and the Chapel Language. *International Journal on High Performance Computer Applications*, 21(3):291–312, 2007.
- [9] P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. von Praun, V. Saraswat, and V. Sarkar. X10: An Object-Oriented Approach to Non-Uniform Cluster Computing. In *Proceedings of Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA)*, October 2005.
- [10] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5(1):46 –55, 1998.

- [11] D. Doerfler, M. Rajan, C. Nuss, C. Wright, and T. Spelce. Application-Driven Acceptance of Cielo, an XE6 Petascale Capability Platform. In *Proc. 53rd Cray User Group Meeting*, 2011.
- [12] A. Geist and S.S. Dosanjh. IESP Exascale Challenge: Co-Design of Architectures and Algorithms. *Int. J. High Perform. Comput. Appl.*, 23:401–402, November 2009.
- [13] M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.W. Willenbring, H.C. Edwards, A.B. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, and R.W. Numrich. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009. <https://software.sandia.gov/mantevo/>.
- [14] MPI Forum. *MPI: A Message Passing Interface Standard, Version 2.2*, 2009. <http://www.mpi-forum.org/docs/mpi22-report.pdf>.
- [15] MPI Forum. *MPI: A Message Passing Interface Standard, Version 3.0*, 2012. <http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>.
- [16] University of Southern California Center for Software Engineering. CodeCount<sup>TM</sup>toolset. [www.sunset.usc.edu/research/CODECOUNT](http://www.sunset.usc.edu/research/CODECOUNT), 1998.
- [17] M. Rajan, C.T. Vaughan, D.W. Doerfler, R.F. Barrett, P.T. Lin, K.T. Pedretti, and K.S. Hemmert. Application-driven Analysis of Two Generations of Capability Computing Platforms: Purple and Cielo. *Computation and Concurrency: Practice and Experience*, 2012. To appear.
- [18] H. Simon, T. Zacharia, and R. Stevens. Modeling and Simulation at the Exascale for Energy and the Environment: Report on the Advanced Scientific Computing Research Town Hall Meetings on Simulation and Modeling at the Exascale for Energy, Ecological Sustainability and Global Security (E3). Technical report, Office of Science, The U.S. Department of Energy, 2007.
- [19] L.G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33:103–111, August 1990.
- [20] C.T. Vaughan, M. Rajan, R.F. Barrett, D.W. Doerfler, and K.T. Pedretti. Investigating the Impact of the Cielo Cray XE6 Architecture on Scientific Application Codes. In *Workshop on Large Scale Parallel Processing, at the IEEE International Parallel & Distributed Processing Symposium (IPDPS) Meeting*, 2011. SAND 2010-8925C.

# Appendix A

## Code categorization

This section provides a breakdown of the source code, as reported by CodeCount [16]. Cross referencing to Figure 2.3 should provide a general understanding of the source code for miniGhost. The distinction between a physical and logical line of code is discussed in the READ file (USC\_Read\_Me-TXT.txt) distributed with the CodeCount package:

**PHYSICAL SLOC :** The number of physical SLOCs within a source file is defined to be the sum of the number of physical SLOCs (terminated by a carriage return or EOLN character) which contain program instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, interpreters, and/of assemblers. It excludes comment cards and unmodified utility software. It includes job control language (compiler directive), format statements, and data declarations (data lines). Instructions are defined as lines of code or card images. Thus, a line containing two or more source statements count as one physical SLOC; a five line data declaration counts as five physical SLOCs. The physical SLOC definition was selected due to (1) compatibility with parametric software cost modeling tools, (2) ability to support software metrics collection, and (3) programming language syntax independence.

**LOGICAL SLOC :** The number of logical SLOC within a source file is defined to be the sum of the number of logical SLOCs classified as compiler directives, data lines, or executable lines. It excludes comments (whole or embedded) and blank lines. Thus, a line containing two or more source statements count as multiple logical SLOCs; a single logical statement that extends over five physical lines count as one logical SLOC. Specifically, the logical SLOC found within a file containing software written in the PL/I programming language may be computed by summing together the count of (1) the number of terminal semicolons, (2) the number of terminal commas contained within a DECLARE (DCL) statement, and (3) the number of logical compiler directives that do not terminate with a terminal semicolon, i.e., JCL directives. The logical SLOC definition was selected due to (1) compatibility with parametric software cost modeling tools, and (2) ability to support software metrics collection. The logical SLOC count is susceptible to erroneous output when the analyzed source code file contains software that uses overloading or replacement characters for a few key symbols, e.g., ';' .

– End quoted text.

Table 2.3 above listed the MPI functionality employed by miniGhost. Figure A.1 lists the Fortran SLOC. Figure A.2 lists the Fortran SLOC required for the BSPMA implementation. The latter is shown with two caveats. First, a significant amount code is shared with the SVAF implementation, and second, this includes code needed only by the SVAF implementation.

File `main.c` adds 388 physical lines and 264 logical lines.

Figure A.1. miniGhost SLOC summary by file

Total Lines	Blank Lines	Comments		Compiler	Data	Exec.	Physical	File	Module
		Whole	Embedded	Direct.	Decl.	Instr.	SLOC	Type	Name
209	36	43	4	0	4	126	130	F77	DRIVER.F
141	28	41	7	0	4	68	72	F77	DRIVER_BSPMA.F
140	26	41	8	0	4	69	73	F77	DRIVER_SVAF.F
113	26	41	5	0	8	38	46	F77	MG_ALLREDUCE.F
166	22	41	1	0	1	102	103	F77	MG_BSPMA.F
196	27	40	4	0	2	127	129	F77	MG_BSPMA_DIAGS.F
382	120	79	9	0	6	177	183	F77	MG_BUFINIT.F
934	209	127	14	0	87	511	598	F77	MG_CHECKPOINT.F
237	52	38	37	0	0	147	147	F77	MG_CONSTANTS.F
130	24	37	2	0	5	64	69	F77	MG_FLUX_ACCUMULATE.F
341	124	60	5	0	9	148	157	F77	MG_GET_FACE.F
227	54	48	1	0	2	123	125	F77	MG_IRECV.F
82	21	32	12	0	0	29	29	F77	MG_OPTIONS.F
180	31	61	2	0	4	84	88	F77	MG_PACK.F
1915	313	80	17	0	0	1522	1522	F77	MG_PROFILING.F
237	50	52	2	0	3	132	135	F77	MG_SEND_BSPMA.F
240	51	52	2	0	4	133	137	F77	MG_SEND_SVAF.F
424	32	43	2	0	2	347	349	F77	MG_STENCIL.F
335	90	91	12	0	18	136	154	F77	MG_STENCIL_COMPS.F
154	18	39	2	0	3	94	97	F77	MG_SUM_GRID.F
308	25	44	6	0	3	236	239	F77	MG_SVAF.F
253	40	57	10	0	6	150	156	F77	MG_SVAF_DIAGS.F
169	34	42	4	0	4	89	93	F77	MG_UNPACK_BSPMA.F
177	38	46	4	0	5	88	93	F77	MG_UNPACK_SVAF.F
1112	284	97	23	1	36	694	731	F77	MG_UTILS.F

FORTRAN SOURCE LINES OF CODE COUNTING PROGRAM  
 ( c ) Copyright 1998 - 2000 University of Southern California, CodeCount (TM)

(a) File listings

Total Lines	Blank Lines	Comments		Compiler	Data	Exec.	Number		File	SLOC
		Whole	Embedded	Direct.	Decl.	Instr.	of Files		Type	Definition
8802	1775	1372	195	1	220	5434	25	5655	F77	Physical
0	0	0	0	0	0	0	0	0	F90	Physical
0	0	0	0	0	0	0	0	0	HPF	Physical
0	0	0	0	0	0	0	0	0	DATA	Physical
								5655	<---Total Physical SLOCs	
8802	1775	1372	195	1	196	3750	25	3947	F77	Logical
0	0	0	0	0	0	0	0	0	F90	Logical
0	0	0	0	0	0	0	0	0	HPF	Logical
0	0	0	0	0	0	0	0	0	DATA	Logical
								3947	<---Total Logical SLOCs	

Ratio of Physical to Logical SLOC (FORTRAN-77)..... 1.43

(b) Summary



Figure A.2. miniGhost BSPMA SLOC summary by file

Total Lines	Blank Lines	Comments		Compiler Direct.	Data Decl.	Exec. Instr.	Physical SLOC	File Type	Module Name
Whole	Embedded								
209	36	43	4	0	4	126	130	F77	DRIVER.F
141	28	41	7	0	4	68	72	F77	DRIVER_BSPMA.F
113	26	41	5	0	8	38	46	F77	MG_ALLREDUCE.F
166	22	41	1	0	1	102	103	F77	MG_BSPMA.F
196	27	40	4	0	2	127	129	F77	MG_BSPMA_DIAGS.F
382	120	79	9	0	6	177	183	F77	MG_BUFINIT.F
237	52	38	37	0	0	147	147	F77	MG_CONSTANTS.F
130	24	37	2	0	5	64	69	F77	MG_FLUX_ACCUMULATE.F
341	124	60	5	0	9	148	157	F77	MG_GET_FACE.F
227	54	48	1	0	2	123	125	F77	MG_IRECV.F
180	31	61	2	0	4	84	88	F77	MG_PACK.F
237	50	52	2	0	3	132	135	F77	MG_SEND_BSPMA.F
424	32	43	2	0	2	347	349	F77	MG_STENCIL.F
335	90	91	12	0	18	136	154	F77	MG_STENCIL_COMPS.F
154	18	39	2	0	3	94	97	F77	MG_SUM_GRID.F
169	34	42	4	0	4	89	93	F77	MG_UNPACK_BSPMA.F
1112	284	97	23	1	36	694	731	F77	MG_UTILS.F

(a) File listings

Total Lines	Blank Lines	Comments		Compiler Direct.	Data Decl.	Exec. Instr.	Number of Files	SLOC	File Type	SLOC Definition
Whole	Embedded									
4753	1052	893	122	1	111	2696	17	2808	F77	Physical
0	0	0	0	0	0	0	0	0	F90	Physical
0	0	0	0	0	0	0	0	0	HPF	Physical
0	0	0	0	0	0	0	0	0	DATA	Physical
									2808 <---Total Physical SLOCs	
4753	1052	893	122	1	111	2455	17	2567	F77	Logical
0	0	0	0	0	0	0	0	0	F90	Logical
0	0	0	0	0	0	0	0	0	HPF	Logical
0	0	0	0	0	0	0	0	0	DATA	Logical
									2567 <---Total Logical SLOCs	

(b) Summary



## DISTRIBUTION:

1	MS 1319	Robert A. Ballance, 1422
1	MS 1318	Robert J. Hoekstra, 1424
1	MS 1319	James A. Ang, 1420
1	MS 1322	Bruce A. Hendrickson, 1440
1	MS 1324	Robert W. Leland, 1400
		,
1	MS 0899	Technical Library, 9536 (electronic copy)





