RESEARCH ARTICLE

*ScienceAsia* 37 (2011): **268–276**

doi: 10.2306/scienceasia1513-1874.2011.37.268

# On efficiency improvement of grid applications

**Sirod Sirisup***, **Suriya U-ruekolan**

Large-Scale Simulation Research Laboratory, National Electronics and Computer Technology Center, 112 Thailand Science Park, Phahon Yothin Rd., Klong 1, Klong Luang, Pathumthani 12120 Thailand

*Corresponding author, e-mail: sirod.sirisup@nectec.or.th

*Received 9 Aug 2010*
*Accepted 31 Aug 2011*

**ABSTRACT**: Grid computing technology addresses the growing need for computing resources in simulation-based engineering and science (SBES). However, optimal performance of grid computing technology requires 'virtually tightly coupled' computing resources. This, though, may not be economically available in many research societies. In this work, we propose a new approach to improve performance of a typical parallel application for SBES on a 'virtually not tightly coupled' grid computing environment. To this end, we have proposed, implemented, and investigated the performance of a new grid programming technique based on the Global Arrays toolkit and multi-level topology-aware approach. We have found that performance of the evaluating application implemented with this new technique outperforms both MPICH-G2 and native Global Arrays implementations, but is still comparable to that of the application implemented with OpenMP, the native Symmetric-Multi Processors programming.

**KEYWORDS**: grid computing, Global Arrays toolkit, multi-level parallelism, multi-level topology-aware

## INTRODUCTION

Grid computing[1] enables the development of large scientific applications on large scale computing resources. Most grid applications in simulation-based engineering and science (SBES) make use of coupled computational resources that cannot be replicated at a single site. Thus grids allow scientists to solve large scale problems by involving aspects of scientific computing and data management.

Most grid applications in SBES, e.g., computational fluid dynamics, computational chemistry, and bioinformatics, have been implemented in the form of parallel applications. The algorithms in the applications typically need immense interprocess communications. The typical requirements in grid computing environment include both low latency and high-speed interconnection which usually are not so easy to establish in some research societies. Thus applications that run on grid computing environment may be degraded. A way to resolve this problem is to reduce network use of interprocess communications in the parallel application, thus efficient network use on grid computing environment can be accomplished.

Typical parallel grid applications on grid computing environment are developed through MPI standard libraries which contain a message passing point-to-point communication (Isend/Ircev). The use of point-to-point communications can affect parallel grid application on grid computing environment in terms of interconnect network use. Hence, implementing parallel grid application with one-sided communication can deliver better performance compared to implementing with point-to-point communication[2]. There are very few parallel libraries that provide one-sided communication capability. These are MPI-2 and Global Arrays toolkit. However, the Global Arrays toolkit offers a simpler and lower-level model of one-sided communication than MPI-2 does through the Aggregate Remote Memory Copy Interface (ARMCI)[3]. Moreover, the MPI-2 library is not currently supported in the Globus Grid toolkit. Another approach to improve the efficiency of the grid application is to employ multi-level topology-aware technique[4–6]. In this paper, we propose, implement, and investigate the performance of a new grid programming technique based on the Global Arrays toolkit and multi-level topology-aware approach. Specifically, we investigate the performance of a parallel application implemented with this new technique on a grid computing environment as well as compare the performance of same application implemented with MPICH-G2 library and native Global Arrays toolkit. The grid environment used in the current investigation is "virtually not tightly coupled" i.e., with low bandwidth and high latency network to resemble the current grid computing environment in Thailand. Nonetheless, as a Symmetric Multi Processors (SMP) cluster is included in grid computing environment, one may ask the following question. In order to achieve the peak performance of the application, which approach should one employ for that SMP cluster: the native SMP programming or this newly proposed programming technique? Our current investigation also addresses this situation as well.

## CORE TECHNOLOGIES AND APPROACHES

In this paper, we propose a new technique to improve performance of a typical parallel application for SBES on a "virtually not tightly coupled" grid computing environment. To achieve that, many core technologies and approaches must be concurrently recruited. Those technologies and approaches are grid computing, MPICH-G2, multi-level topology-aware, Global Arrays, ARMCI programming model. In order to provide the comparison for the case that an SMP cluster is included in the grid computing environment, the OpenMP API will be also briefly introduced here. The details of each technologies and approaches are briefly described next.

### Grid computing and MPICH-G2

High-performance 'computational grids' involve heterogeneous collections of computers that may reside in different administrative domains, run different software, be subject to different access control policies, and be connected by networks with widely varying performance characteristics[1]. By using communication mechanisms for a grid environment, such as MPICH-G2 to use services provided by the Globus toolkit, any application can run across clusters spread over campus-area, metropolitan-area or wide-area networks. The MPICH-G2 uses the Globus toolkit's Resource Specification Language (RSL)[7] to describe the resources required to run an application. User write RSL scripts, which identify resources and specify requirements and parameters for each. An RSL script can be used as the user interface to globusrun, an upper-level Globus service that first authenticates the user by using Grid Security Infrastructure (GSI)[8] and the schedules and monitors the job across the various machines by using two other Globus toolkit services: the Dynamically-Updated Request Online Coallocator and Grid Resource Allocation and Management[7]. RSL is designed to be an easy-to-use language to describe multi resource multi site jobs while hiding all the site-specific details associated with requesting such resources.

### Multi-level topology-aware

The multi-level topology-aware approach minimizes messaging across the slowest links at each level by clustering the processes at the wide-area level into site groups, and then within each site group, clustering processes at the local-area level into machine groups. One benefit of using a multi-level topology-aware tree to implement a collective operation is that we are free to select different subtree topologies at each level. This technique thus allows developer to create a multi-level parallelism application based on a proper parallel tools in each machine group. The MPICH-G2 addresses this issue within the standard MPI framework by using the MPI communicator construct to deliver topology information to an application. It associates attributes with the MPI communicator to communicate this topology information, which is expressed within each process in terms of *topology depths* and *colours* used by MPICH-G2 to represent network topology in a computational grid[4]. MPICH-G2 introduced the concept of colours to describe the available topology. It is limited to at most four levels, that MPICH-G2 call: WAN, LAN, system area and, if available, vendor MPI[9]. Those four levels are usually enough to cover most use cases. However, one can expect finer-grain topology information and more flexibility for large-scale grid systems. Topology-discovery features in Globus have been used to implement a topology-aware hierarchical broadcast algorithm in MPICH-G2. However, complex applications require a larger diversity of collective operations, including reductions, barrier, and sometime all-to-all communications.

### Global Arrays and ARMCI

The Global Arrays[10] was designed to simplify the programming method on distributed memory systems. The most innovative idea of Global Arrays is that it provides an asynchronous one-sided, shared memory programming environment for distributed memory systems. The Global Arrays has included the ARMCI library which provides one-sided communication capabilities for distributed array libraries and compiler run-time systems. ARMCI offers a simpler and lower-level model of one-sided communication than MPI-2[2,3]. Global Arrays reduces the effort required to write parallel program for clusters since they can assume a virtual shared memory. Part of the task of the user is to explicitly define the physical data locality for the virtual shared memory and the appropriate access patterns of the parallel algorithm[11]. There is also another package that is closely related to Global

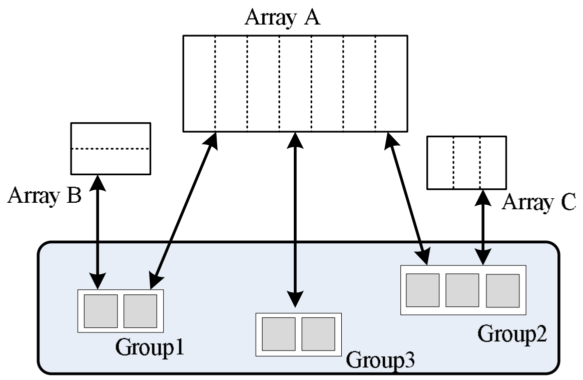

**Fig. 1** An example of multi-level parallelism in Global Arrays.

Arrays in handling global address[12].

The development of multi-level parallel algorithm in SBES with the Global Arrays toolkit has been enabled by introduction of the Global Arrays processor group. Due to the required compatibility of Global Arrays with MPI, the MPI approach to the processor group management was followed as closely as possible. However, in shared memory programming, management of memory and shared data rather than management of processor groups itself is the primary focus. More specifically we need to determine how to create, efficiently access, update, and destroy shared data in the context of the processor management capabilities that MPI already provides. For example, Fig. 1 illustrates the concept of using shared arrays. The three processor groups (Group1, Group2, and Group3 in Fig. 1) execute tasks that operate on three arrays: A, B, and C. Array A is in the scope of all three processor groups. Array B is distributed on processor Group 1. Array C is distributed on processor group 3. All arrays can be accessed using collective (individual and multiple arrays) and one-sided (non-collective) operations.

## OpenMP

OpenMP is an industry standard[13] for shared memory programming. OpenMP is based on a combination of compiler directives, library routines and environment variables it is used to specify parallelism on shared memory machines. Communication in the OpenMP application is implicit and OpenMP applications are relatively easy to implement. In theory, OpenMP makes better use of the shared memory architecture. Runtime scheduling is allowed both fine-grained and coarse-grained parallelism. OpenMP codes will however only run on shared memory machines and the placement policy of data may cause problems. Coarse-grained parallelism often requires a parallelization strategy similar to an MPI strategy and explicit synchronization is required.

## SCHEME, EVALUATING APPLICATION, AND IMPLEMENTATION

In order to evaluate the performance of the proposed technique for a typical parallel application for SBES, the structure of the parallel algorithm in the evaluating problem should comprise of interprocess communications significantly and should not fall into the class of embarrassingly parallel category. Besides, the evaluating problem itself should also represent most of the applications for simulation-based science and engineering as well. To this end, in this study, a program that finds steady-state heat distribution over a thin plate is chosen to be the evaluating application because the algorithm used in this application matches the above requirements. The program essentially solves the following governing partial differential equations:

$$u_{xx} + u_{yy} = f(x, y), \qquad 0 \leqslant x \leqslant a, \quad 0 \leqslant y \leqslant b. \tag{1}$$

The finite difference approximation discretizes the computational domain into a set of discrete mesh points $(x_i, y_j)$ with evenly spaced of distance $h$. With a zero source term, the finite difference equation representing (1) is reduced to:

$$\hat{u}_{i,j} = \frac{\hat{u}_{i+1,j} + \hat{u}_{i-1,j} + \hat{u}_{i,j+1} + \hat{u}_{i,j-1}}{4} \tag{2}$$

where $\hat{u}_{i,j}$ represents the approximation of $u(x_i, y_j)$. The solution process starts with initial estimates for all $\hat{u}_{i,j}$ values, then the iterative process must be performed until the values converge; we refer to this process as the Jacobi method. The convergence of the Jacobi method is guaranteed because of the diagonally dominant structure; here, we set the number of the iterations to be 2500. For the detailed description of each implementation, suppose that we are working with an $n \times n$ mesh with $p$ processes, with the row-wise block-stripped decomposition, each of the $p$ processes manages its own mesh of size $n/p \times n$, see Fig. 2. In each iteration, each interior process must send and receive $2n$ values to and from its immediate neighbours in order to update boundary cells in its block, denoted as cells with dashed line in Fig. 2. The detail of each implementation is described as follows.

### MPICH-G2 implementation

For the MPICH-G2 implementation application, in each iteration, each interior process must send and receive $2n$ values to and from its immediate neighbours

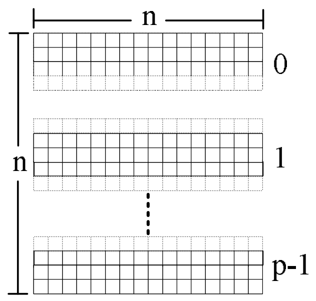

**Fig. 2** A row-wise block-stripped decomposition of $n \times n$ mesh with $p$ processes.

in order to update boundary cells in its block. The implementation of the algorithm with MPI library is quite straight forward by simply using the `MPI_Send` and `MPI_Receive` functions.

## Global Arrays implementation

For Global Arrays implementation, in order to retrieve and update data for each processor's block, in contrast to the previous implementation, we specifically employ the one-sided communication operations provided by the Global Arrays toolkit e.g., the `NGA_Get` and `NGA_Put` functions. Here, each process individually updates and retrieves data needed for calculating the finite difference equation (2), from the Global Arrays blocks by individually calling `NGA_Get` and `NGA_Put` functions; the data validity is handled by ARMCI[3]. To illustrate this, we present the calls made by each processor in Fig. 3. The Global Arrays block updated by `NGA_Put` function called by processor 0 is shown as enclosed area in Fig. 3a which is its own block. However, `NGA_Get` function called by processor 0 retrieves the data from the Global Arrays block shown as enclosed area in Fig. 3b which also includes the data from processor 1. Like processor 0, processor 1 can call `NGA_Put` function to update data in its own block, Fig. 3c. However, being an interior processor, processor 1 needs to retrieve data that also include boundary data from processor 0 and processor 2 by calling `NGA_Get` function, Fig. 3d. Other interior processors can directly employ the same strategy as processor 1 does. However, a similar strategy as processor 0 is employed for the other exterior process, processor $p-1$.

## Global Arrays with multi-level topology-aware implementation

This is our proposed technique, the key idea is to separate processors into groups of processors that belong to the actual cluster sites in the grid computing

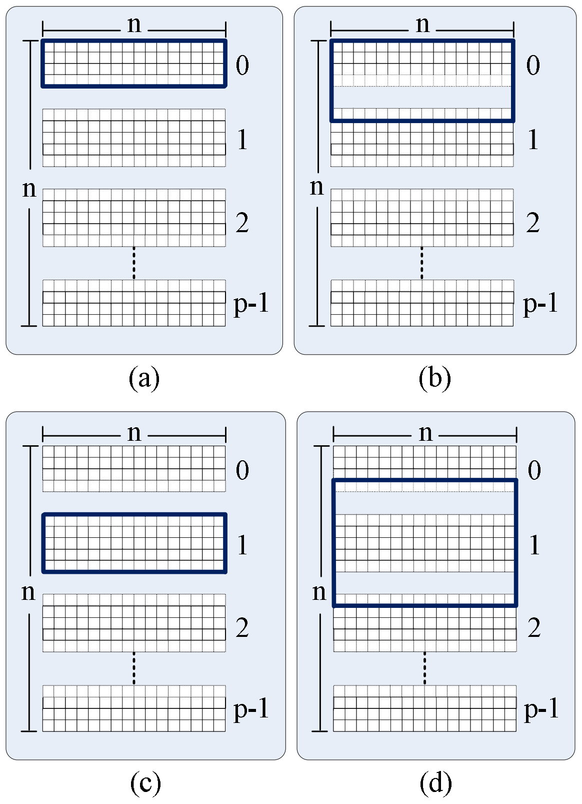

**Fig. 3** A row-wise block-stripped decomposition of $n \times n$ mesh with $p$ processes for Global Arrays implementation.

environment. We can then take advantages of the performance of the Global Arrays toolkit on each cluster individually[3], by creating a Global Arrays block for each group of processors. The group of processors is defined as a process group in the Global Arrays toolkit. In this scenario, we have one process group per one cluster site and thus many process groups in our grid computing environment. The communications for the boundary cells between process groups are solely handled by the master of each process group. Thus the communications between process groups i.e., actual cluster sites can be greatly reduced yielding a much improved performance.

Practically, the separation of processors in our grid computing environment into groups of processors can be accomplished by digesting the topology depths and colors reported by `MPI_Attr_get` function provided by MPICH-G2 library through the `MPICHX_TOPOLOGY_DEPTHS`, `MPICHX_TOPOLOGY_COLORS` variables, see line 10 to 13 in Listing 1. Specifically, the variable `MPICHX_TOPOLOGY_DEPTHS` represents number of network levels and `MPICHX_TOPOLOGY_COLORS`

**Listing 1** A fragment of application code that uses `MPICHX_TOPOLOGY_DEPTHS` and `MPICHX_TOPOLOGY_COLORS` to create group communicators.

```
...
int *depths;
int **colors;
MPI_Status status;
MPI_Comm comm_master, comm_lan;
MPI_Init (&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&me);
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
...
MPI_Attr_get(MPI_COMM_WORLD,        \
MPICHX_TOPOLOGY_DEPTHS,&depths,&flag);
MPI_Attr_get(MPI_COMM_WORLD,        \
MPICHX_TOPOLOGY_COLORS,&colors,&flag);
...
MPI_Finalize();
```

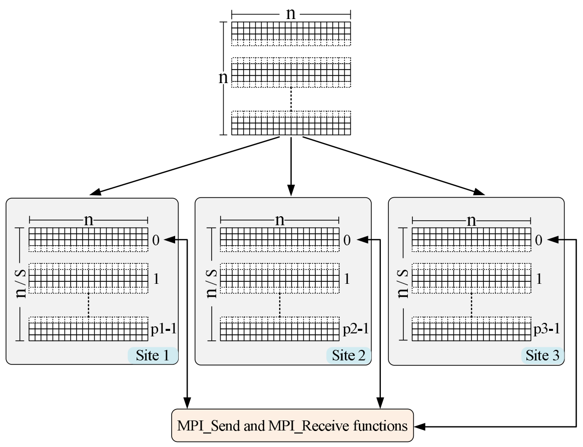

**Fig. 4** A row wise block-stripped decomposition of ($n$/number of sites($S$)) $\times$ $n$ mesh with $p$ processes in the current technique.

represents information indicating that any two processors with the same color can communicate with each other in that network level i.e., processors located on the same cluster site will have the same color. For more information on these variables see Ref. 9. We also assign the master processor of each group of processors e.g., process group to be the processor with the lowest world rank in each process group. The specific communicator is then created for these master processors in order to enable the communication between them.

For our evaluating problem, the original mesh of size $n \times n$ is reduced to a mesh of size ($n$/number of sites($S$)) $\times$ $n$ which is handled individually by a group of processors of size $p$ on each cluster site, see Fig. 4. It is noted that the number of processors $p$ on each site is not necessary the same. However, for our proposed technique to be applicable, the number of processors $p$ on each site must be at least two. This restriction directly comes from the technical issue of the Global Arrays toolkit for creating process group.

In each site, the Global Arrays is employed to handle the mesh by creating Global Arrays blocks handling exclusively by those processes in each site. The exclusively handling mechanism is possible through those Global Arrays process group APIs, see line 6, 7, and 14 in Listing 2. The processes in each site can thus use the one-sided communication operations `NGA_Get` and `NGA_Put` locally in each process group. In order to send and receive $2n$ values to and from its immediate neighbour sites, a master processor (processor with lowest rank in each site) is designated to perform this task through the `MPI_Send` and `MPI_Receive` functions. After the information has been exchanged, the master processor thus effectively transfers the boundary data to the proper processor in its site through Global Arrays block by calling `NGA_Put` function. In order to retrieve the boundary data, that corresponding processor needs to call `NGA_Get` function.

## OpenMP with multi-level topology-aware implementation

For comparison, we also perform the study for the inclusion of an SMP cluster into the grid computing environment. Here, we want to investigate the performance of the evaluating application implemented with our proposed technique compared to that implemented with the SMP native implementation in such scenario. In this case, instead of creating a Global Arrays block on the SMP cluster, we rather implement the evaluating application for that specific cluster based on OpenMP. Specifically, complier directives e.g., **`#pragma`** `omp parallel` and **`#pragma`** `omp` **`for`** have been specified in order to invoke and direct OpenMP threads to perform calculations, see line 1, 3, and 9 in Listing 3. However, the communications between neighbour processor groups are still handled by the master processor of this SMP group with the `MPI_Send` and `MPI_Receive` functions.

It is also noted that these algorithms also form a multi-level parallelism because they are composed of inter-site MPICH-G2 calls and intra-site Global Arrays calls or inter-site MPICH-G2 calls and OpenMP

**Listing 2** A fragment of application code that uses Global Arrays process group APIs.

```
...
MPI_Init (&argc, &argv);
...
GA_Initialize();
int group;
group=                          \
GA_Pgroup_create(site,numsite[0]);
...
gu=GA_Create_handle();
GA_Set_array_name(gu, "Array_U");
GA_Set_data(gu, 2, dims, C_DBL);
GA_Set_irreg_distr_(gu,map,nblock);
...
GA_Set_pgroup(gu, group);
GA_Allocate(gu);
...
GA_Pgroup_sync(group);
...
if(my_group_id==0)
NGA_Put(GA_ghost,lo,hi,ghostR,ld);
GA_Pgroup_sync(group);
if(my_group_id==groupsize-1)
NGA_Put(GA_ghost,lo,hi,ghostR,ld);
...
GA_Destroy(gu);
GA_Pgroup_destroy(group);
GA_Terminate();
...
MPI_Finalize();
```

**Listing 3** A fragment of application code that uses OpenMP directives.

```
#pragma omp parallel private(i,j)
         {
#pragma omp for
          for(i=starti; i<endi; i++)
          for(j=1; j<N-1; j++)
            {
             ...
            }
#pragma omp for nowait
          for(i=starti; i<endi; i++)
          for(j=1; j<N-1; j++)
            {
             ...
            }
         }
```

directives.

We want to emphasize that even though the selected evaluating application is quite straightforward however its linear solver algorithm and related relatives are actually the cores of many SBES applications[14].

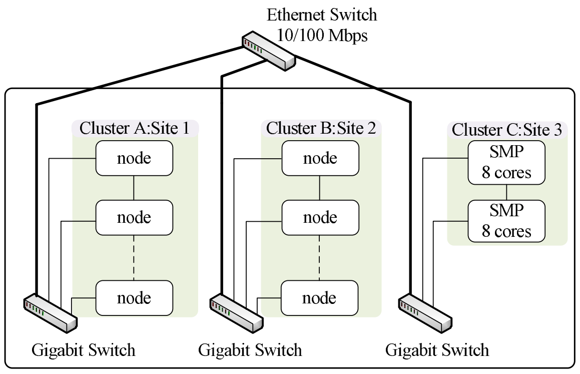

**Fig. 5** Experimental setting for the grid environment used in this study.

## EXPERIMENTAL SETTING

In this section, we provide a summary of experimental setting description for the grid computing environment used in the current investigation. The grid computing environment is composed of three clusters consisting of two eight-host clusters and a two-host SMP cluster connected through the ThaiSarn network with the bandwidth of approximately 30 Mbit/s, see Fig. 5 for full network map. We would like to also emphasize that the current setting does reflect the 'virtually not tightly coupled' grid environment i.e., a grid environment with low bandwidth and high latency network.

The specification of each individual node and its interconnect network communication are as follows. For cluster A and B (Non-SMP clusters), two Itanium 2 (IA64) 1.3 GHz CPUs per node with 4 GBytes of RAM with Gigabit Ethernet network interface card on Linux Kernel 2.6.9, MPICH 1.2.7 and GNU C and Fortran compilers 3.2.3. For Cluster C (SMP cluster), eight Xeon 2.0 GHz CPUs per node with 16 GBytes of RAM with the exact same running environment and interconnect network communication.

In the current study, the Globus toolkit (4.0.5) together with MPICH-G2 are used in the current grid computing environment. The Global Arrays does not require any specific configuration for compiling the Global Arrays on Globus toolkit.

## RESULTS AND DISCUSSION

We have implemented the evaluating application on the grid computing environment in various implementation techniques described above. In order to

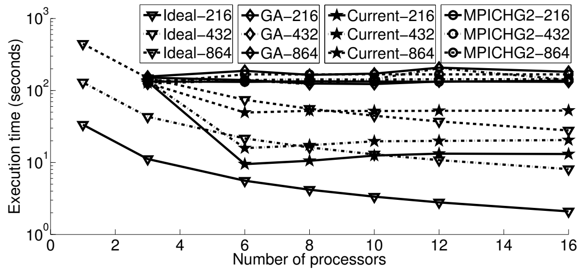

**Fig. 6** Execution time of each implementation on the grid computing environment.

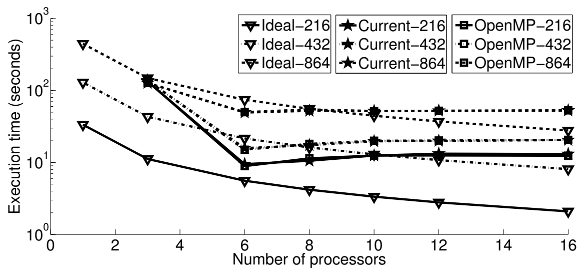

**Fig. 7** Execution time of each implementation on the grid computing environment focusing on SMP native implementation (OpenMP) and the current technique.

understand the characteristic of the proposed technique, we separate the performance analysis into two parts: first, the comparison of the proposed technique to the other typical implementations and second, the comparison of the proposed technique to the SMP native implementation for the case of the inclusion of an SMP cluster.

## Comparison to typical parallel implementations

In this part, we focus on the performance comparison of the evaluating application with the exact same algorithm in three types of implementations: native Global Arrays, MPICH-G2 and the proposed technique. All implementations have been executed on various resolutions ranging from $216 \times 216$, $432 \times 432$, to $864 \times 864$ mesh resolutions. The numbers of processors used here are 3(1–1–1), 6(2–2–2), 8(4–2–2), 10(4–4–2), 12(4–4–4), and 16(4–4–8) where ($np_1$–$np_2$–$np_3$) represents numbers of processors from site 1, site 2, and site 3, respectively.

The results of execution time for all implementations on the grid computing environment and the ideal performance is shown in Fig. 6. It is noted here that in the case of three processors i.e., one processor per site, implementations based on MPICH-G2 and the proposed technique are actually identical. From the figure, we can see clearly that the implementation based on multi-level topology-aware implementation, the proposed technique, greatly outperforms the other two implementations. We also see that the native Global Arrays implementation performs poorly compared to MPICH-G2 implementation. This is the expected result because there are large amount of uncontrollable network communications between processor $p$ and other processors which are not the immediate neighbours of processor $p$. These uncontrollable communications can lead to a significant performance drop if those processors are "virtually not tightly coupled" as in this grid computing environment. In the case of MPICH-G2 implementation, even though we can explicitly control the communication pattern in this implementation technique, the MPICH-G2 Globus I/O uses SSL for securing messages and this does take some overhead use. When we add that to the cost from the synchronous function calls, it can result in the performance degradation as well. Here, we see that only multi-level topology-aware implementation, the proposed technique, can provide a close agreement to the ideal performance. However, this is only valid for the cases that we have large enough data size for each processor.

## Comparison to SMP native implementation

We now move to the scenario that there is at least one SMP cluster has been included in the grid environment. We also want to examine the efficiency of our proposed technique, Global Arrays with multi-level topology-aware implementation, compared to that of the SMP native implementation on that SMP cluster as well. To this end, we have also implemented the evaluating application in OpenMP with multi-level topology-aware technique for this case as well. The detail of the implementation has already been described above.

The results of execution time of both implementations is shown in Fig. 7. From the figure, we can see that their performances are comparable. Besides, as compared to the ideal performance, both implementations show super-linear speed up in 6,8 processors where a proper data size is met.

In order to provide a strategy on the proposed technique for choosing an optimal combination of numbers of processors in each site, we also investigate the application performance with various combination of numbers of processors in each site on this type of grid environment as well. The results are presented in Table 1. From Table 1, for the OpenMP implementation (i.e., the OpenMP with multi-level topology-aware technique), we see that an increase

**Table 1** Execution time (seconds) for various combination of ($np_1$–$np_2$–$np_3$) for implementations with the current technique (A) and OpenMP (B).

| $p$ | $n = 216$ | | $n = 432$ | | $n = 864$ | |
|---|---|---|---|---|---|---|
| | A | B | A | B | A | B |
| 6(2–2–2) | 9.49 | 9.90 | 15.84 | 15.83 | 49.41 | 49.48 |
| 8(4–2–2) | 10.55 | 11.42 | 17.36 | 18.15 | 52.12 | 54.52 |
| 8(2–2–4) | 9.95 | 9.59 | 18.27 | 15.64 | 53.15 | 49.32 |
| 10(4–4–2) | 12.48 | 12.48 | 19.69 | 20.04 | 51.84 | 51.86 |
| 10(2–2–6) | 11.79 | 10.38 | 19.15 | 15.78 | 53.67 | 49.57 |
| 12(4–4–4) | 13.26 | 12.59 | 19.91 | 20.19 | 52.29 | 52.27 |
| 12(2–2–8) | 12.71 | 9.90 | 19.54 | 16.44 | 54.39 | 49.67 |

in number of processors in SMP node results in a decrease of the execution time. However, the decrease of the execution time is not that significant (within 3%) and including even more threads does not help improve application performance that much either. Besides, a higher ratio of the number of processors from non-SMP sites can bring unfavourable impact to the overall performance as well. For the Global Arrays with multi-level topology-aware implementation, the proposed technique, although the execution time of the application is really close to that of OpenMP implementation, still shows strong favour of performing intra-machine computation to inter-machine computation. Also, from Table 1, we can see that the execution time of the proposed technique gradually increases as more processors from other nodes are added. This implies that to effectively use the proposed technique on SMP architecture, we still need some special techniques for properly handling global address provided by Global Arrays. In the case of availability of larger improvement margin which is not our case here, one can include an SMP-aware technique[15] in the proposed technique to achieve that improvement.

## CONCLUSIONS

In this paper, we propose a new approach to improve performance of a typical parallel SBES application on a low bandwidth and high latency grid computing environment. The newly proposed technique consists of using Global Arrays toolkit with multi-level topology-aware technique for implementing a parallel application on such grid computing environment. We have implemented the proposed technique for the evaluating application based on an iterative solver for a large system of linear equations. The performance comparisons are done in twofold: comparison to typical parallel implementations and comparison to the system specific implementation. In the current study, the OpenMP API is used for the system specific implementation. From the first comparison, it is found that the evaluating application implemented with the newly proposed technique outperforms the other two typical parallel implementations, the MPICH-G2 and native Global Arrays implementations, in all studied cases. This proves the efficiency of the proposed technique. We have also found that, in the second comparison, the evaluating application implemented with this new proposed technique can perform very close to that implemented with the system specific implementation. It should be also noted that the application implemented with the proposed technique can be directly ported to a grid computing environment that includes SMP clusters and its performance is almost the same to that of the system specific programming. This also proves the robustness of the proposed technique. However, to effectively use the Global Arrays with the proposed technique, the developer must make sure that the algorithm of the application is stable. This is because the explicit control of the Global Arrays (process groups) operations may not be easily done. Moreover, the initial investment in re-coding of the application based on Global Arrays with multi-level topology-aware technique is quite considerable compared to the MPICH-G2 implementation.

## REFERENCES

1. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the Grid: Enabling scalable virtual organizations. *Int J High Perform Comput Appl* **15**, 200–22.
2. Thakur R, Gropp W, Toonen B (2005) Optimizing the synchronization operations in message passing interface one-sided communication. *Int J High Perform Comput Appl* **19**, 119–28.
3. Nieplocha J, Carpenter B (1999) ARMCI: A portable remote memory copy library for distributed array libraries and complier run-time systems. In: *Proceedings of the 11th IPPS/SPDP'99 Workshops*, Heidelberg, pp 533–46.
4. Karonis N, Foster I, Gropp W, Lusk E, Bresnahan J (2000) Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In: *Proceedings of the International Parallel Processing Symposium*, pp 377–84.
5. Xavier C, Sachetto R, Vieira V, Weber dos Santos R, Meira W (2007) Multi-level parallelism in the computational modeling of the heart. In: *Proceedings 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*, pp 3–10.
6. Dong S, Lucor D, Karniadakis GE (2003) Multi-level parallel paradigms for flow-induced vibrations. *NAVO MSRC Navigator*, Fall 2003, 5–8.

7. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S (1998) A resource management architecture for MetaComputing systems. *The Fourth Workshop on Job Scheduling Strategies for Parallel Processing*, pp 62–82.
8. Butler R, Welch V, Engert D, Foster I, Tuecke S, Volmer J, Kesselman C (2000) A national-scale authentication infrastructure. *IEEE Computer* **33**, 60–6.
9. Karonis N, Toonen B, Foster I (2003) MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *J Parallel Distr Comput* **63**, 551–63.
10. Nieplocha J, Harrison RJ, Littlefield RJ (1996) Global Arrays: A nonuniform memory access programming model for high-performance computers. *J Supercomput* **10**, 169–89.
11. Nieplocha J, Palmr B, Tipparaju V, Krishnan M, Trease H, Apra E (2005) Advances, applications and performance of the Global Arrays shared memory programming toolkit. *Int J High Perform Comput Appl* **20**, 203–31.
12. Nomoto A, Watanabe Y, Kaneko W, Nakamura S, Shimizu K (2004) Distributed Shared Arrays: Portable shared-memory programming interface for multiple computer systems. *Cluster Comput* **7**, 65–72.
13. Dagum L, Menon R (1998) OpenMP: An industry standard API for shared-memory programming. *IEEE Comput Sci Eng* **5**, 46–55.
14. Strang G (2007) *Computational Science and Engineering*. Wellesley-Cambridge Press.
15. Träff JL (2003) SMP-aware message passing programming. In: *Eighth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'03)*, pp 56–65.