An efficient Modules for HPC Topologies Mapping

Saad B. Alotaibi[1] and Prof. Fathy Elbouraey[2]

[1,2]King Abdulaziz University, Jeddah, KSA

Summary
Nowadays, as we are moving towards the Exascale era, the topology-aware process mapping is becoming an important approach to improve the performance and reduce the power consumption of Exascale applications. Accordingly, most researchers in this area have proposed many techniques and approaches for finding the best and efficient topology aware process mapping. In this paper, we have proposed the main modules for any high-performance computing (HPC) topologies mapping technique which includes parallel application behavior analyzer module, physical topology generator module for the target machine, virtual topology generator module for the entire parallel application and topologies mapper module for mapping the process to the processor during runtime. All models work as a dynamic modules.

Key words:
HPC, Physical Topology, Virtual Topology, Topologies Mapping, Dynamic Module, Static Module.

1. Introduction

High performance computing (HPC) refers to a computing system and environment that typically uses many processors (as part of a single machine) or several computers organized in a cluster (operating as a single computing resource). There are many types of HPC systems ranging from large clusters of standard computers to highly specialized hardware. Most cluster-based HPC systems use high-performance network interconnects, such as those from InfiniBand or Myrinet. The basic network topology and organization can use a simple bus topology [1] [2]. In high performance environments, the mesh network system provides a shorter latency between hosts, thus improving overall network performance and transmission rate. “Figure 1” shows a mesh HPC system. In a mesh network topology, this architecture accelerates cross-host communication by reducing the physical and logical distance between network nodes. Although network topology, hardware, and processing hardware are important in HPC systems, the core functionality that makes the system so effective is provided by the operating system and application software.

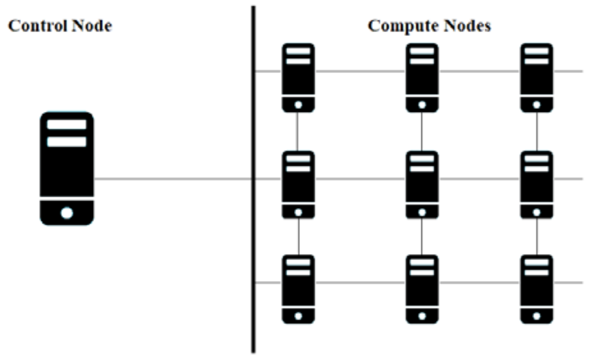


Fig. 1 HPC: Compute Nodes & Control Node

HPC systems use specialized operating systems that are designed to look like a single computing resource. As can be seen from “Figure 1”, there is a control node that forms the interface between the HPC system and the client. The control node also manages the work distribution of the compute nodes [3]. For task execution in a typical HPC environment, there are two models: Single Instruction/Multi Data (SIMD) and Multiple Instruction/Multi Data (MIMD). SIMD performs the same computational instructions and operations across multiple processors, but for different data ranges, it allows the system to compute the same expression using many variables simultaneously. MIMD allows the HPC system to perform different calculations at the same time using different variables, making the entire system look like not just a computing resource without any features (although it is powerful), it can perform many calculations simultaneously. Whether using SIMD or MIMD, the basic principles of a typical HPC are still the same: the entire HPC unit behaves like a single computing resource, spreading the actual requested load to each node [4]. HPC solutions are also dedicated units that are specifically designed and deployed to act as large computing resources. Based on the previous, the matching communication patterns of the parallel application into target machine topology (HPC systems) is an important task in order to improve the performance of the application as well as reduce the power consumption. As we know, the modern machine has multi nodes, cores, memories, etc. Actually,

Manuscript received February 5, 2019
Manuscript revised February 20, 2019

this complexity of the devices needs to be cautious in adapting the application to these devices in order to run the application in the best form. For that matter, we have studied how to make the important steps to do the mapping of the virtual topology of the parallel application onto the physical topology of the target machine. We have built an architecture in order to clarify the details of each module and facilitate constructive. In the following section we described the important technical background related to the main modules of the topologies mapping.

2. Technical Background

2.1 Parallel Programming Models

Parallel programming is a programming method that achieves faster than serial programming by simultaneously executing computer instructions. Parallel programming is a concept that is proposed in relation to traditional serial programming. In serial programming, a program's instructions are executed sequentially on a single CPU, while in parallel programming, a program is divided into separate parts and executed synchronously on one or more CPUs to achieve higher execution of efficiency and performance [5].
According to the underlying memory structure, parallel programming can be divided into the following three programming types:

- Shared memory model “Figure 2”: Multiple threads or processes running simultaneously [6]. They share the same memory resource, and every thread or process can access the memory anywhere. For example, OpenMP uses a shared memory model.
- Distributed Memory Model “Figure 3”: Multiple independent processing nodes work at the same time, and each processing node has a local private memory space. A process executing a program can directly access its private memory space. If a process needs to access a private space at another processing node, the process needs to send information to the process for access. MPI is a distributed memory model [6].
- Distributed shared memory model “Figure 4”: The entire memory space is divided into shared space and private space. Each thread has access to all of the shared space, and each thread has its own separate private space. Unified Parallel C uses a split global address space model [6].

The following figure “Figure 2” explains the shared memory, distributed memory as well as distributed shared memory as a hardware architecture.

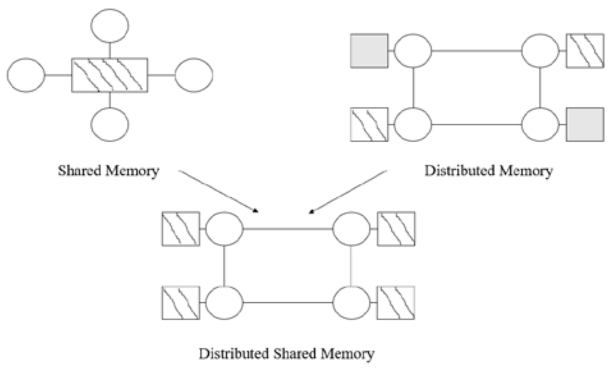


Fig. 2 Shared, Distributed and Shared Distributed Model

These types of parallel programming models can be integrated as a hybrid parallel programming models, the following section illustrates the hybrid model.

2.2 Hybrid Parallel Programming Model

Today, high-performance computer (HPC) systems typically have a hierarchical hardware design: a shared memory multi-core CPU to many nodes connected via a network infrastructure. Parallel programming must parallelize distributed memory parallelism with shared memory in parallel within each node in the node interconnect. This trend is even intended to expand the range of hardware design that will continue in the foreseeable future of its advanced technology system. However, always with pure MPI, each CPU core has a processing option, with its own address space, a separate entity [7]. In the study of computing distribution, the interface of transport is based on the parallel application that extends to numerous algorithms of different kinds of processors. This can experience loss of efficiency when located in such a construction of obligatory applications. The ability of parallel is primarily reduced when many cores of processors are enabled. Otherwise, OpenMP is a shared parallel programming memory paradigm that delivers parallelism of thread-level. Combining OpenMP and MPI parallelization to build a hybrid program offers two levels of similarity. This kind of approach resolves the communication load of MPI and exposes the OpenMP overhead due to destruction and creation. For various applications, the idea of the model which is hybrid model OpenMP and MPI has made it prove better performance than the only method of MPI. The use of OpenACC interface delivers instructions of the compiler, environment variables, library calls to enable programmers to state code of parallel for platforms such as the (GPGPU) Which is referred as general graphics processing unit. Scanning MPAC by using OpenACC can run system of the side in a cluster using the total amount of a number of cores of aggregate allocated between nodes of the group [8]. Finally, the following sections explain the main parallel programming models in each hybrid technique.

2.2.1 OpenACC

A new instruction-based accelerator parallel programming standard was released at the Seattle Supercomputing Conference (SC11). The goal of this development standard is to allow more programmers to use GPU computing, while the results can be used across accelerators and even on multicore CPUs. In short, OpenACC instructions work much the same way as OpenMP instructions, but the former is especially useful for highly data-parallel code [9]. They can be plugged into standard C, C++ and Fortran programs to direct the compiler to parallelize certain code segments. The compiler pays special attention to the logical relationship of data moving back and forth between the CPU and the GPU (or other) and maps the computation to the appropriate processor. This way, developers can make relatively small changes to existing or new code to mark the accelerated parallel region. Since the instruction design is for a general-purpose parallel processor, the same code can run on a multi-core CPU, GPU, or any other type of parallel hardware supported by the compiler. This hardware independence is especially important for HPC users because they are reluctant to accept vendor-restricted, non-portable programming environments. To this end, the OpenACC design team emphasizes hardware independence. Programmers cannot specify or guarantee the use of any hardware-specific features in OpenACC directives because portability and performance portability are prioritized [10].

2.2.2 MPI: OpenMPI

Before the 1990s, programmers were not so lucky. Writing concurrent programs for different computing architectures is a difficult and lengthy task. At the time, many software libraries were able to help write concurrent programs, but there wasn't a standard that everyone accepted to do this. At the time, most concurrent programs only appeared in the fields of science and research [11]. The most widely accepted model is the messaging model. What is a messaging model? It actually refers to a program that accomplishes certain tasks by passing messages between processes (a message can be understood as a data structure with some information and data). In practice, concurrent programs are particularly easy to implement with this model. For example, the master process can assign this work to it by sending a message describing the job to the slave process. Another example is that a concurrent sorter can sort the data visible to the current process (which we call local) in the current process, and then merge the sorted data sent to the neighbor process. Almost all parallel programs can be described using a messaging model. Since many of the software libraries used this messaging model at the time, but there were some minor differences in definitions, the authors of these libraries and others have defined a standard for messaging interfaces at Supercomputing 1992 in order to solve this problem - also It is MPI. This standard interface allows programmers to write concurrent programs that run in all major concurrency frameworks. And allow them to use the features and models of some popular libraries that are already in use at the time [11] [12]. By 1994, a complete interface standard was defined (MPI-1). We have to remember that MPI is just a definition of an interface. Then the programmer needs to implement this interface according to different architectures. Fortunately, a complete MPI implementation has emerged. After the first implementation, MPI was heavily used in the messaging application, and remains such programs written standard (de-facto).

2.2.3 OpenMP

OpenMP (Open Multi-Processing) is a multi-threaded concurrent programming API that supports cross-platform shared memory. It runs on most processor architectures and operating systems using C, C++ and Fortran languages, including Solaris, AIX., HP-UX, GNU/Linux, Mac OS X, and Microsoft Windows. Includes a set of compiler directives, libraries, and some environment variables that can affect the behavior of the run [13] [14]. OpenMP is portable, extensible model provides programmers with a simple and flexible development platform, from standard desktop computers to supercomputer parallel application interfaces. Applications built with a hybrid parallel programming model can use both OpenMP and MPI, or more transparently through a cluster of computers running on OpenMP-extended non-shared memory systems. OpenMP is a widely accepted set of Compiler Directives for multithreaded programming of shared memory parallel systems, led by the OpenMP Architecture Review Board [15]. OpenMP support of programming languages, including C language, C ++ and Fortran; and support for the OpenMP compiler include Sun Studio and Intel Compiler, as well as open source in GCC and Open64 compiler. The programmer specifies his intention by adding a dedicated pragma to the source code, so that the compiler can automatically parallelize the program and add synchronization mutex where necessary. In other word, OpenMP is a cross-platform multi-threaded implementation where the main thread (sequential execution instructions) generates a series of sub-threads and divides the tasks into these sub-threads for execution. These child threads run in parallel, and the runtime environment assigns threads to different processors. The code snippet to be executed in parallel needs to be marked accordingly. The precompiled instruction is used to generate the thread before the code snippet is executed. [16] [17]

2.3 Topologies

Topology is used to refer to a structure attached to a set X that essentially depicts set X as a topological space that can handle the properties of convergence, connectivity, and continuity under transformation. In this dissertation, we focus on two types of topologies including the virtual topology of the parallel application and the physical topology of the target machine. Regarding to virtual topology, the communication model of a process set can be represented by a graph, the nodes represent processes, and the edges are used to connect processes that communicate with each other [18]. MPI provides messaging between any pair of processes in a group, and does not require a channel to be opened explicitly. Therefore, in a user-defined process map, "chain loss" does not prevent messages from being exchanged between processes, which means this connection is ignored in the virtual topology. This strategy implies that the topology does not give a way to name this communication path. Another possible result is that when mapping an automatic mapping tool (if it exists in the runtime environment) will not consider this edge, there is no weighting in the communication graph, so the process is either simple or not connected at all. Using a diagram to illustrate the virtual topology is sufficient for all applications. However, in many applications the graph structure is regular, and the creation of detailed graphs is inconvenient for the user and may lack effectiveness at runtime. Most of the parallel applications used use process topologies like rings, 2D or higher dimensional meshes, and rings. These structures are completely defined by the number of dimensions and the number of processes in each respective coordinate direction. Moreover, grid and ring mapping problems are generally easier than mapping of normal graphs. Finally, a lot of applications use the processes topology [19]. The following figure “Figure 3” show all the virtual topology features.

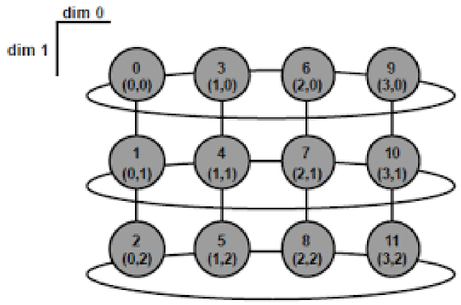


Fig. 3 Virtual Topology Example with features [20]

For the physical topology, the physical topology represents the hardware details whether in the form of a tree or graph. The hardware maybe networks details, processors, memories, caches, storages, and so on. In the following figure “Figure 4” An example of the physical topology (Hardware Information).

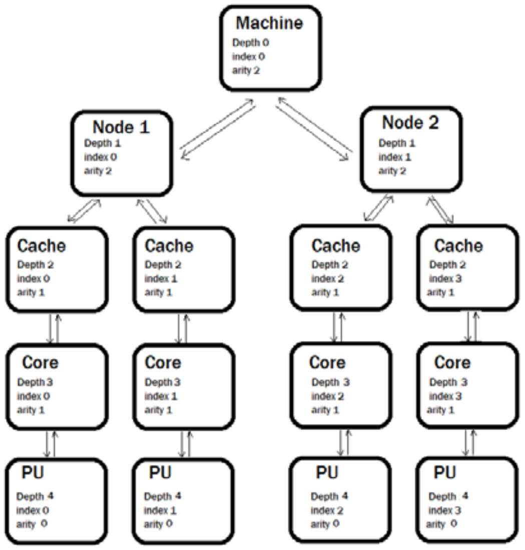

Machine Depth 0 index 0 arity 2
Node 1 Depth 1 index 0 arity 2
Node 2 Depth 1 index 1 arity 2
Cache Depth 2 index 0 arity 1
Cache Depth 2 index 1 arity 1
Cache Depth 2 index 2 arity 1
Cache Depth 2 index 3 arity 1
Core Depth 3 index 0 arity 1
Core Depth 3 index 1 arity 1
Core Depth 3 index 2 arity 1
Core Depth 3 index 3 arity 1
PU Depth 4 index 0 arity 0
PU Depth 4 index 1 arity 0
PU Depth 4 index 2 arity 0
PU Depth 4 index 3 arity 0

Fig. 4 Physical Topology

2.4 Parallel Application Communications

Parallel application has two important types of communication. The first one is peer-to-peer communication or point-to-point communication which involves only two different processes at the same time. The second one is collective communication which refers to a method involving all processes in a communicator. To begin with first type in the following section.

2.4.1 Point-to-Point Communication (MPI Send and Receive)

Sending and receiving are two basic concepts in MPI. Almost all individual methods in MPI can be implemented using the underlying send and receive APIs. The method of sending and receiving MPI is as follows: At the beginning, the A process decides to send some messages to the B process. The A process will package all the data that needs to be sent to the B process and put it in a cache. Because all data is packaged into a large message, the cache is often compared to an envelope (just like we pack a lot of stationery into an envelope and then send it to the post office). After the data is packaged into the cache, the communication device (usually the network) is responsible for passing the information to the right place. The correct place is the process that is determined by a particular rank. Although the data has been sent to B, Process B still needs to confirm that it wants to receive A's data. Once it determines this, the data is transmitted successfully. Process A receives the information that the data is delivered successfully, and then goes to other things. Sometimes A needs to pass a lot of different messages to B. In order to make it easier for B to distinguish between different messages, the MPI runtime sender and receiver additionally specify some information IDs (official names are tags). When B only asks for information about a particular tag, other information that is not the tag will be cached first, and will be given to B when B needs it

2.4.2 Collective Communications

We have explained peer-to-peer communication, which involves only two different processes at the same time. Collective communication is a method of communication which involves participation of all processes in a communicator. The most important thing with collective communication is the idea of synchronization points between the parallel application processes. This means that all processes of parallel application in the implementation of the code must first have to reach a synchronization point to continue the implementation of the code behind. The example of collective communications is Broadcast, Allgather, Gather and Scatter. Now, we've covered all the important information about topologies mapping, for that matter, we will explain the main modules of any HPC topologies mapping in the following sections which combines "Application Behavior Analyzer Module", "Physical Topology Generator Module", "Virtual Topology Generator Module", and finally the "Topologies Mapper Module".

3. HPC Topologies Mapping Modules

To facilitate the idea we will start with the high level design of any HPC topologies mapping (Proposed Technique) based on our research in this area "Figure 5".

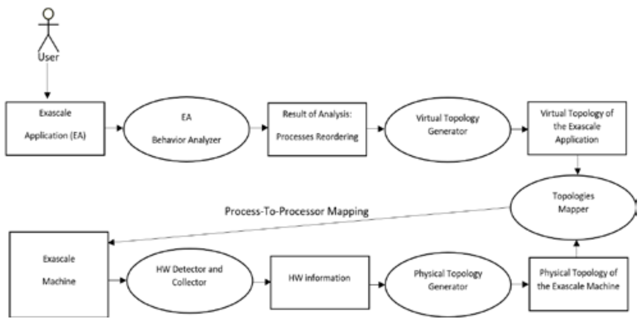

User
Exascale Application (EA)
EA Behavior Analyzer
Result of Analysis: Processes Reordering
Virtual Topology Generator
Virtual Topology of the Exascale Application
Topologies Mapper
Process-To-Processor Mapping
Exascale Machine
HW Detector and Collector
HW information
Physical Topology Generator
Physical Topology of the Exascale Machine

Fig. 5 Proposed Technique for any HPC Topologies Mapping

The proposed technique architecture shows the exact steps of the mapping algorithm. In the beginning, the proposed technique analyzes the entire parallel application by the "Behavior Analyzer algorithm, will be explained later in details", and then detect the processes that have the highest communication between each other. Based on this result, the "Virtual Topology Generator" builds the virtual topology. In addition, the virtual topology pass in several stages, starting from detect the number of parallel application processes to generate the grid of the virtual topology. In same time and as parallel step, the proposed technique detects the target machine and collect all the necessary information such as cores, GPUs, CPUs, memories etc. and then create the physical topology by "Physical Topology Generator". After that and based on the previous steps, the mapping algorithm takes the virtual topology and physical topology in order to initialize the mapping process. Mapping algorithm read the virtual topology as a grid with multi dimension and this grid was ordered based on the highest communication between the processes. After that, takes the processes one by one based on the grid ordered and bind it on the processor. This processor was extracted from the physical topology and this topology was built in the form of a tree with multi levels. The mapping algorithm reads this tree and get the exact levels of processor ID and then map this processor with process. Moreover, the mapping algorithm ensures the process implementation on this processor until the end. Finally, the proposed technique has four main modules including: Application Behavior Analyzer Module, Physical Topology Generator Module, Virtual Topology Generator Module, Topologies Mapper Module. The following sections will explain all the modules in details.

3.1 Parallel Application Behavior Analyzer Module

Parallel application has two important types of communication. The first one is peer-to-peer communication or point-to-point communication which involves only two different processes at the same time. The second one is collective communication which refers to a method involving all processes in a communicator. In addition, the main model of parallel application is MPI. MPI doesn't have a direct way to detect the number of messages that transmitted between the processes. For that matter, we have developed a parallel application behavior analyzer without editing on the main code for detecting and counting all the messages that happened between all the processes. After that, we used this number of messages between the processes as a weight for reordering the processes in the virtual topology to prepare it for the mapping process. The following figure "Figure 6" illustrates the architecture of the application behavior analyzer module.

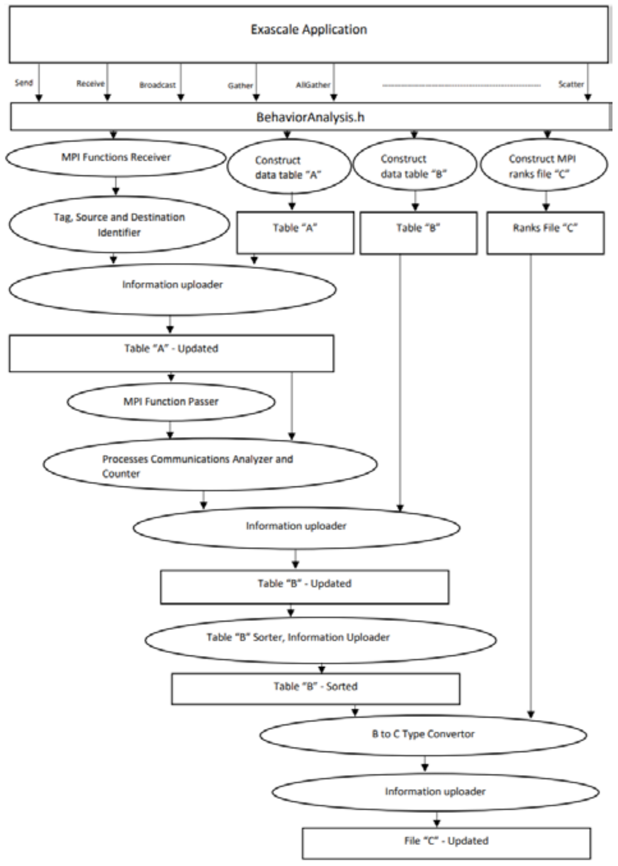


Fig. 6 Architecture of the application behavior analyzer module

The application behavior analyzer works as a middle between the MPI functions calls and the functions execution as shown in the architecture "Figure 6". To begin with first step, the module works with receive the MPI communication functions only such as Send, Receive, Broadcast, ...etc. before execution. After that, the analyzer uses Tag of the messages to detect and classify the message, then identify the Id of process source and destination for consequence action which is save all communications that happens between the processes with IDs in table A, then allow original MPI communication function to complete execution. We have now data that represents the communications between all the processes in the parallel application. The second step of the analyzer, takes this data and make a mathematical calculations to provide how many times process X communicate with process Y, such as (P1 $\rightarrow$ P2 = 4) and then save this results in the table B. The third step of the analyzer module is construct file "C" type of MPI ranks file for saving the final result which is sorting the processes based on the highest communication that's happened between the processes. Finally, the "Virtual Topology Generator" access this file "File C" to reordering the processes in the virtual topology to put the processes that has highest communications near of each other. The "Virtual Topology Generator Module" described in the following section.

3.2 Virtual Topology Generator Module

In reality, most of studies depends on the technology of graph for creating the virtual topology. We have built our virtual topology as a grid to facilitate the communication between processes. So that, the MPI provide two important types of virtual topology including Graph and Cartesian. The proposed virtual topology based on the Cartesian virtual topology. The following figure "Figure 7" describes the architecture of the virtual topology generator module.

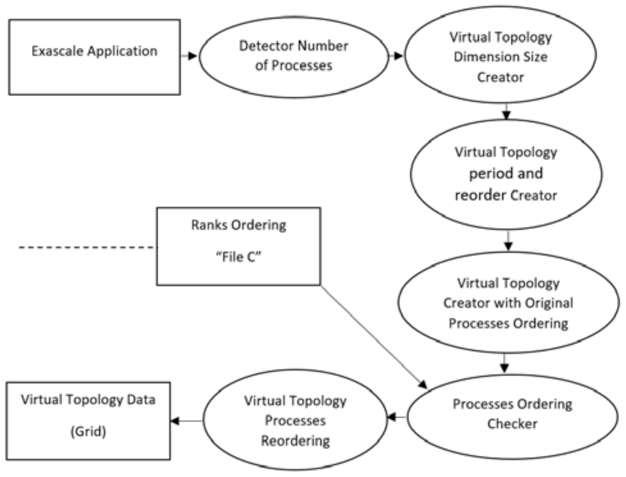

Exascale Application
Detector Number of Processes
Virtual Topology Dimension Size Creator
Virtual Topology period and reorder Creator
Ranks Ordering "File C"
Virtual Topology Creator with Original Processes Ordering
Processes Ordering Checker
Virtual Topology Processes Reordering
Virtual Topology Data (Grid)

Fig. 7 Virtual Topology Generator Module Architecture

First of all, the “virtual topology generator module” detect the processes number of the Exascale application, based on this number the dimension of the virtual topology will be created (ex: 6 processes – 2-Dimensions 2*3). Then, the generator makes the virtual topology accepted the processes reordering to allow the mapping algorithm match the virtual topology onto the physical topology in the right way. Moreover, the generator makes this topology as a periodic for a cyclic boundary. Then, it collects all the processes of the Exascale application and export it into the virtual topology with numeric ordering. At this step, we generate the virtual topology that contains all the Exascale processes, these processes not ordered based on any wright, for that matter we have developed the “Application Behavior Analyzer Module” to provide us the new processes ordering based on the highest communication between each other. The “virtual topology generator module” takes this weight for reordering the processes in the grid and put the processes that has a highest communication near to each other. Finally, we have the virtual topology for the Exascale application and ready to map it with the physical topology. The physical topology generator module explained in the following section.

3.3 Physical Topology Generator Module

Actually, gathering the information of target HW not easy task. We have tried a lot of methods and technique to get the hardware information but we can’t do it as a dynamic hardware topology. With HWLOC [21], we can gather all the necessary information of the target hardware and we have developed our physical topology as a dynamic physical topology based on HWLOC library. Our proposed topology, depends on the operating system of the target machine to detect the processors are Idle or Busy. The following figure “Figure 8” explains the architecture of the physical topology generator.

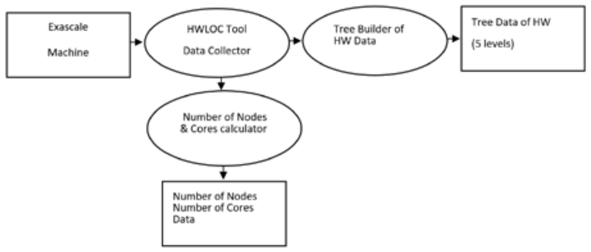

Exascale Machine
HWLOC Tool Data Collector
Tree Builder of HW Data
Tree Data of HW (5 levels)
Number of Nodes & Cores calculator
Number of Nodes Number of Cores Data

Fig. 8 Physical topology generator architecture

The physical topology generator module begins with detect the target hardware and then aggregate all the hardware information including NUMA memory nodes, shared caches, processor sockets, processor cores, PCI devices/bridges and simultaneous multithreading …etc. After that, the generator built a tree with 5 levels to export this information, the root will be the machine and the levels are processing units. In the parallel step, the generator constructs a data table to save the number of nodes and the number of cores because this information is very important during the mapping phase, to divide the communication patterns of the Exascale application based on these numbers. Actually, the benefit of a tree approach over a topology matrix is that there is no need for considering the aspects of the latency and speed of the cache hierarchy. In the algorithm, upward processing is carried out for the topology tree. The processes are grouped recursively based on the next level's arity. Finally, the mapping algorithm uses this tree to prepare the process-to-processor mapping stage. the “Topologies Mapper Module” described in the following section.

3.4 Topologies Mapper Module

The mapper module depends entirely on the modules of virtual topology and physical topology generator. The following architecture explains the module in details “Figure 9”.

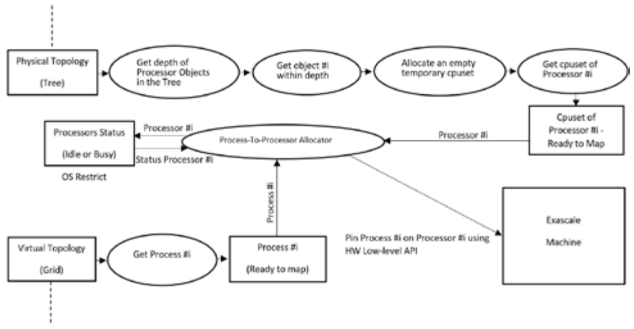

Physical Topology (Tree)
Get depth of Processor Objects in the Tree
Get object #i within depth
Allocate an empty temporary cpuset
Get cpuset of Processor #i
Cpuset of Processor #i - Ready to Map
Processor #i
Process-To-Processor Allocator
Processor #i
Processors Status (Idle or Busy)
Status Processor #i
OS Restrict
Process #i
Pin Process #i on Processor #i using HW Low-level API
Exascale Machine
Virtual Topology (Grid)
Get Process #i
Process #i (Ready to map)

Fig. 9 Mapper Architecture

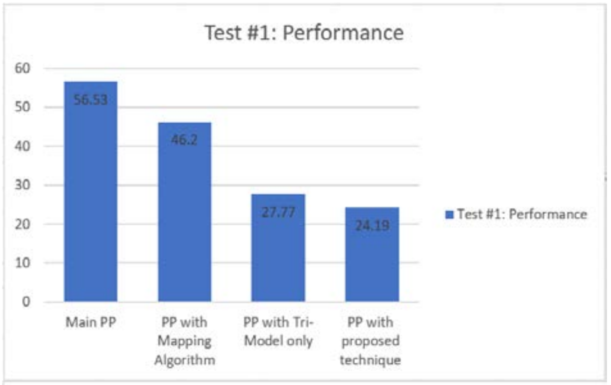
The topologies mapper module works after generate the physical topology of the target machine and the virtual topology of the parallel application. The first step, the mapper takes the physical topology and reads the tree from the first level (Level 0 = Machine) and go to the processor depth (level 3 = core), and then get the index of this object as a physical index. After that, the mapper module allocates cpuset as an empty temporary set regarding to specific processor in specific depth, this cpuset has the target processor index and then check if this index busy or idle based on the target machine operation system. The second step, the mapper takes the virtual topology and reads the grids and start taking the first process Id. The last step, the mapper takes this process id and processor index and pin it together by hardware low level function using hardware API based on C language, and then bind this process with target processor until end the execution to prevent the process migration. Finally, we've got fantastic results after implementing and testing these models with each other, the following section cover the implementation and testing of these modules.

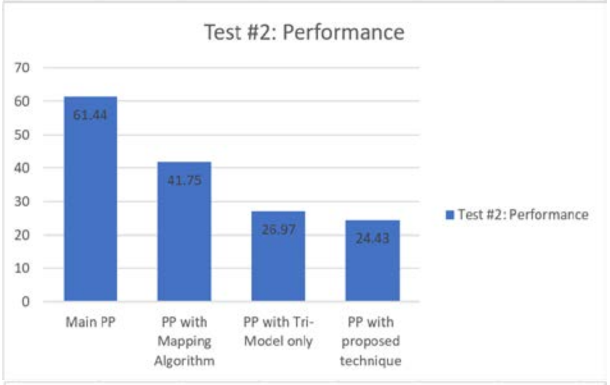
4. Implementation and Testing

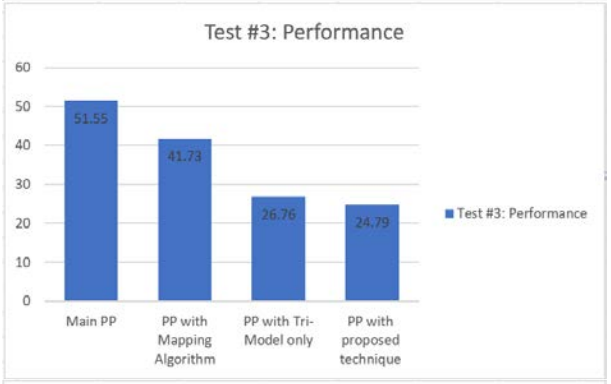
The explained implementation and testing in this paper were performed on a cluster with multicores platform (12 processors). We have tested the proposed modules more than 50 times, and these modules demonstrated impressive performance when there was a higher level of irregularity in the communication pattern. Gains shown in some cases were even up to the extent of 50 percent. All the parallel programming was tested with 6 processes. These modules tested with many parallel applications and we have got very impressive results. To begin with the first implementation, we have developed a more complicated parallel application to test power consumption and application performance. This application contains 100000000 calculations for each process and we have developed this application with 6 processes, then we have in this application (6*100000000) calculations. Moreover, this application contains 10000000 communications as a Collective communications not point-to-point communication, this collective communication as a Broadcast for each process, this means every process send one message to all processes 10000000 times, then we have (6*6*10000000) communications in this parallel application. We have implemented this application without and with our proposed modules Actually, we have got excellent results in terms of reduce power consumption and improve the performance of parallel application. The following table "Table 1" and graphs as a result of the implementation of the above-mentioned parallel application.

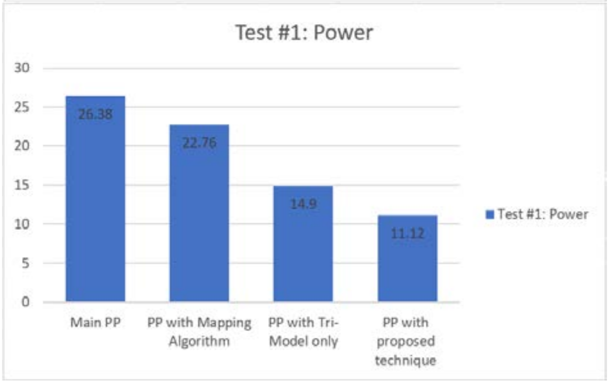
Table 1: all testing results of collective communications

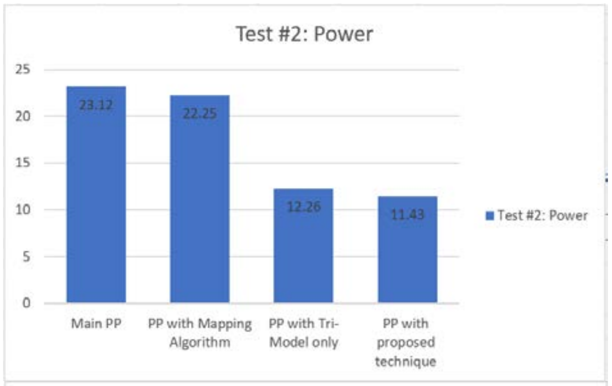
Performance Measurement		
	Main Parallel Application	Parallel Application with Proposed Modules
Test #1	56.53 s	24.19 s
Test #2	61.44 s	24.43 s
Test #3	51.55 s	24.79 s
Power Measurement		
	Main Parallel Application	Parallel Application with Proposed Modules
Test #1	26.38 w	11.12 w
Test #2	23.12 w	11.43 w
Test #3	22.99 w	11.24 w

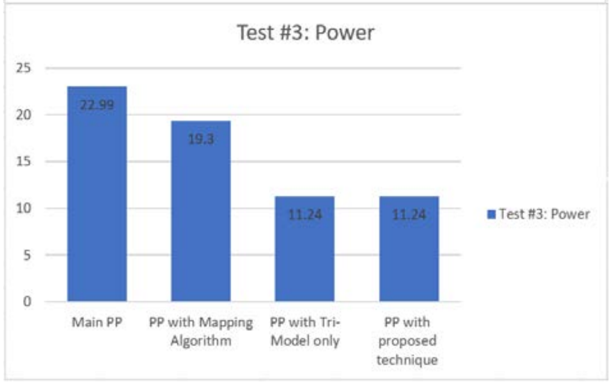


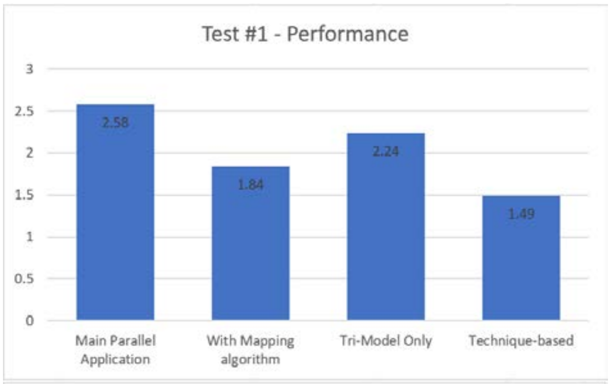


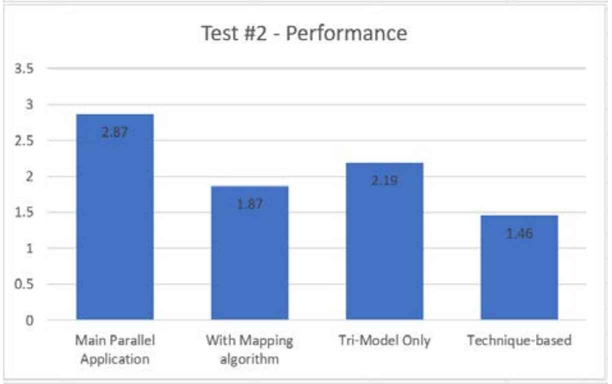


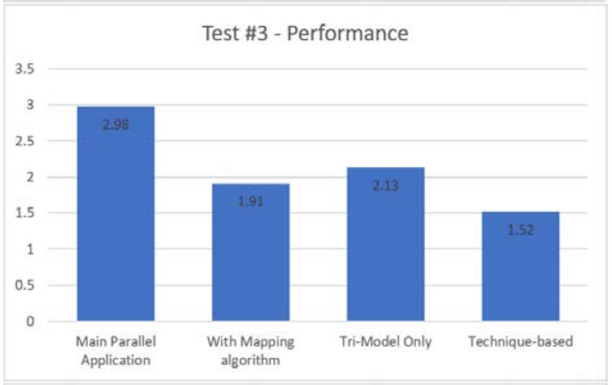


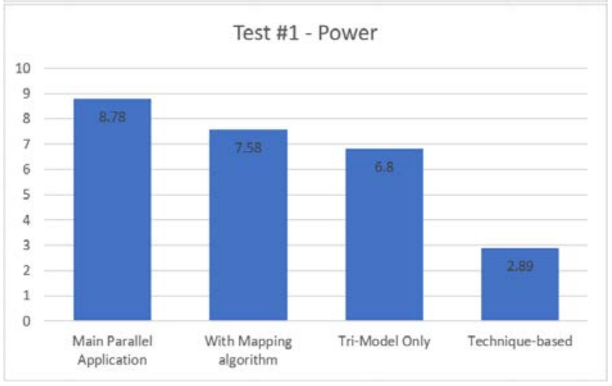








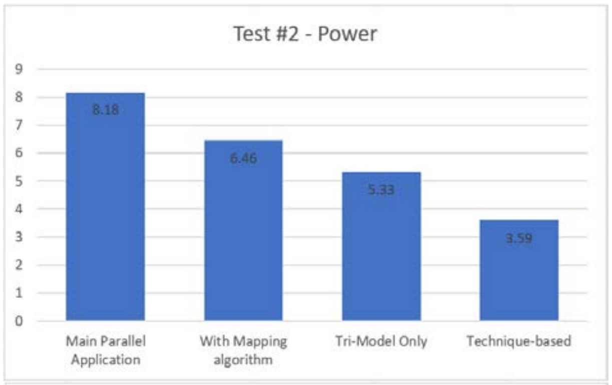


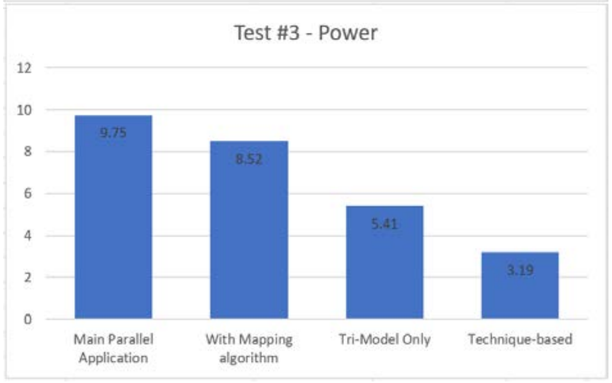


Finally, we have developed a point-to-point parallel application, also this parallel application is very complicated because it has a 6M communications between the processes. We have tested this application with and without our proposed technique and got an excellent result in terms of reduce the power consumption as well as improve the application performance. The following graphs and table “Table 2” show the experiments results.

Table 2: all testing results of point-to-point communications

Performance Measurement		
	Main Parallel Application	Parallel Application with Proposed Modules
Test #1	2.58 s	1.49 s
Test #2	2.87 s	1.46 s
Test #3	2.98 s	1.52 s
Power Measurement		
	Main Parallel Application	Parallel Application with Proposed Modules
Test #1	8.78 w	2.89 w
Test #2	8.18 w	3.59 w
Test #3	9.75 w	3.19 w





After the implementation and testing of our proposed modules, we have discussed the results in the following section.

5. Discussion

The excellent results that we have got after add the proposed technique onto the target parallel application it indicates that the proposed technique works in an excellent way and achieved the desired objectives in terms of reduce the power consumption and improve the parallel application performance. Moreover, in each result we observed big reduction in the power consumption as well as time and that refer to three reasons.

1. The first one, the processes of the parallel application were executed near each other because the mapping algorithm maps the processes physically close to each other.
2. The second reason, the volume of exchange data between the processors is huge and that need to map the processes physically close to others to make the processing nearby from others to reduce the time as well as the traffic of data exchange.
3. The last reason, the algorithm ensures the processing of the parallel application process was bound on the specific processor until the processing accomplished.

Actually, the execution way of the proposed algorithm helps the parallel application to execution without any obstacles or code modification. This way including: detect the number of parallel application processes and after that, the virtual topology was created during the runtime based on the number of processes, and then using the tracing algorithm to trace all the messages or communications that happened between these processes and after that used this analysis to ranks reordering to make the processes that contain highest communications near each other. The virtual topology was modifies based on the results of the application behavior analyzer algorithm, to generate the weights of the processes. The weights are the volume of messages which are between the parallel application processes. In same time, the physical topology was created with weights. the weights are the physical ordering or logical distance. Once both topologies are ready, it’s the time to the mapping algorithm. The mapping algorithm takes the processes of the virtual topology and the processors of the physical topology and map it together. Moreover, the mapping algorithm bind this process on the processor unit the execution finish, so as to ensure that it is not transferred to another processor.

Through many tests on the proposed technique, we have observed several points on the results and divided into two main parts are as follows:

5.1 Collective Communication

In this part, the parallel application after using our proposed technique has improvement near 43% of normal parallel application. As for power consumption, we have seen the reduction is near 42% after using our proposed technique, the following table show the summary of collective communication results after using our proposed technique.

	Performance improvement	Power Consumption Improvement
Parallel Application: Collective Communication	43%	42%

5.2 Point-To-Point Communiction

Regarding to this type of parallel application, we have observed the improvement of the performance after using our proposed technique is near 48% of normal parallel application. As for power consumption, we have seen the reduction is near 47% after using our proposed technique. the following table show the summary of collective communication results after using our proposed technique.

	Performance improvement	Power Consumption Improvement
Parallel Application: Point-To-Point Communication	48%	47%

This modules will be help the parallel applications in terms of increased performance, as well as help high performance computing reduce energy usage.

6. Conclusion

As a matter of fact, after developing these modules which is parallel application behavior analyzer module, virtual topology generator module, physical topology generator module and topologies mapper module we have improved the parallel application performance as well as reduce the power consumption. These models have been tested a lot, and has proven itself in terms of improving the application performance and reducing the power consumption. Acutally, these module doesn’t need any special configuration or modicfication on the appliction code, its work like a library used by any parallel application. In the future works, we will add a new module to use the the compiler directives that deals with a specific hardware to achieve better performance.

References

[1] A. c. n. o. expertise, "Introduction to High-Performance Computing".

[2] S. Wind, "Optimization of cpu topology and ceph use," 2017.

[3] I. HPC, "What is high performance computing?," Insidehpc.

[4] C. S. Communication, "High performance computing," China, 2018.

[5] National Energy Research Scientific Computing Center, "OpenMP Resources, 2016.

[6] T. Shan, "A programming model supports all architectures," 2017.

[7] landcareweb, "What is the difference between openacc and openmp and mpi?," 2017.

[8] P. Springer, "OpenACC - A Step Towards Heterogeneous Computing," German Research School for Simulation Sciences GmbH Laboratory for Parallel Programming.

[9] Easy GPU Parallelism with OpenACC, Rob Farber, drdobbs, 2012

[10] Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation , Edgar Gabriel et al., 2004

[11] W. Kendall, MPJExpress Tutorials, "MPI tutorial introduction”, 2017.

[12] G. W. Amy Brown, "OpenMPI," in The Architecture of Open Source Applications , 2007.

[13] T. H. G. B. J. J. D. Teng Ma, "Process Distance-aware Adaptive MPI Collective Communications," 2011 IEEE International Conference on Cluster Computing, 2011.

[14] A. Sen, "Learning the OpenMP framework with GCC," 2012.

[15] Wikipedia, OpenMP Architecture Review Board "OpenMP," 2013.

[16] L. M. Tim Mattson, "A “Hands-on” Introduction to OpenMP," OpenMP.\

[17] "OpenMP directives," Microsoft, 2018

[18] Wikipedia, "Topology"

[19] T. Hatazaki, "Rank Reordering Strategy for MPI Topology Creation Functions".

[20] Npac, "Cartesian and graph topologies," 1999

[21] Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R.: hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In: IEEE (ed.) PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing. Pisa, Italie (Feb 2010)