Towards an Efficient Topology-Aware Process Mapping for Exascale Systems

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

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

Summary
As we push toward the Exascale time, the quantity of nodes in the high-performance computing and the quantity of cores in every node are quickly expanding. Thus, the physical topology of present day multicore high-performance computing frameworks is winding up increasingly complex both at the inward and inner levels regarding the communication inside and among the nodes. This converts into an expansion in the level of heterogeneity, which results in different purposes of correspondence execution inside the correspondence node. Specifically, between node correspondence is commonly slower than interior node correspondence utilizing shared memory. In this paper, we depict a mapping algorithm that matches parallel application processes as a virtual topology with the physical topology of the target machine to improve the application performance as well as reducing the power consumption. It is also a way to integrate triple models of the parallel programming model as a hybrid tri-model to develop the performance of parallel applications and that combine OpenACC, OpenMP and OpenMPI. We actualized these contributions (the mapping algorithm based on the hybrid model) and accomplished fantastic outcomes to reduce the power consumption and improve the performance of the application.

Key words:
HPC; Exascale; Topology-Aware process Mapping; Parallel Application; Virtual Topology; Physical Topology; OpenACC; OpenMP; OpenMPI

1. Introduction

As we move toward the Exascale era, the number of nodes in the system and the number of cores in each node are rapidly increasing. As a result, the physical topology of modern HPC systems is becoming more complex both at the intra and inter levels [1]. The present trend in HPC is to see a significant upsurge in the consistency of individual nodes in terms of the number of cores, memories and cache levels. For that matter, the applications of high-performance computing need to adapt the heterogeneity platforms to optimum execution. As an illustration, the topology-aware process mapping is a way of carrying out a particular task to enhance parallel application execution by decreasing the communication cost of processes by matching the virtual topology of the parallel application to the target underlying hardware architecture called physical topology. One of the advantages of the topology-aware process mapping is matching the application processes to the processors that are physically close one to the other. In order to do a topology-aware process mapping, it is necessary to choose the parallel programming models that help in this matter. To put it another way, the parallel programming model has a valuable help in parallel application execution, because some of the parallel programming models have a mechanism that helps the application to exploit the underlying hardware to improve communication and the locality. Moreover, it will be helpful for virtual topology management to reorganize the processes according to the target underlying hardware architecture. Therefore, the most important parallel programming model is the Message Passing Interface (MPI) which is the standard model of the parallel programming models. As illustration in this section, we implement an efficient algorithm-based technique to take full advantage of the hardware architecture "Figure 1" of the target machine called physical topology and map the processes "Figure 2" of parallel applications to the hardware which are close to each other physically. This technique is combination from two algorithms and hybrid model of parallel programming model as a tri-model. For the algorithms, the first one is tracing algorithm that can be analysis the parallel application and detect the number of transmit messages between the processes. The second algorithm, it makes the mapping of virtual topology onto the physical topology based on the results of the first algorithm. These algorithms integrated with the hybrid model of the parallel programming model which combine OpenACC, OpenMP and OpenMPI. The proposed technique handles large-scale communications (Collective Communications) rather than peer-to-peer communications including Broadcast "Figure 3", AllGather "Figure 4", Scatter "Figure 5" and Gather "Figure 6". Finally, the following sections will explain all the information referred to in this section in details.

Manuscript received January 5, 2019
Manuscript revised January 20, 2019

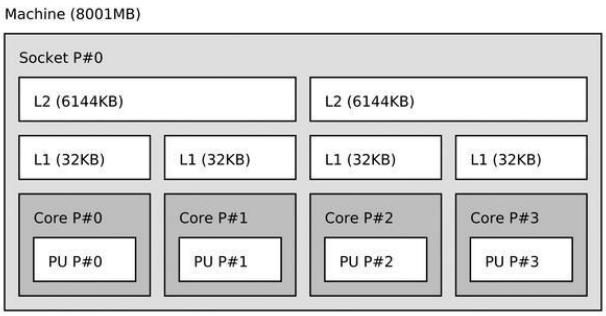

Fig. 1 Hardware information of the target machine

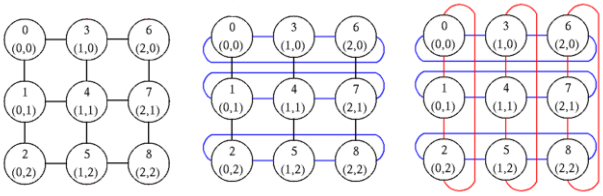

Fig. 2 Virtual topology with 9 processes and has multi direction

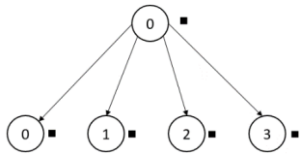

Fig. 3 MPI_Broadcast

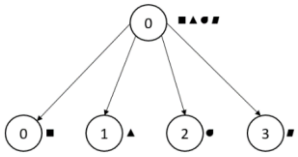

Fig. 4 MPI_Scatter

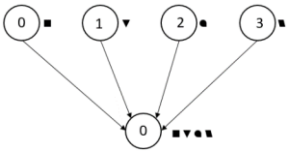

Fig. 5 MPI_Gather

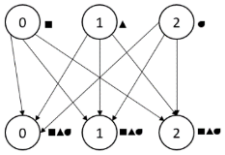

Fig. 6 MPI_Allgather

2. Problem statement

Nowadays, we observe the number of nodes and the number of cores within the node are increasing, as well as the complexity of the memory hierarchy and the higher parallelism. This makes the high-performance computing systems becoming very complex, which leads to the increasing in the heterogeneity levels in many aspects, including the number of nodes, cores, memories, NUMAs, GPUs, CPUs and others. As a consequence, the processes of the parallel application (communication patterns) need to adapt and map carefully to this hardware topology for optimum execution as well as manage the cost of the data movement. Because the cost of the data movement is the important factor in modernistic HPC. So that, its necessary to find an optimal solution for the cost of the data movement and power consumption. As an illustration, the following figure "Figure 7" show the processes of the parallel application that mapped to different processors without considering the processors affinity that can reduce the communication costs as well as the power consumption. Moreover, its very important to develop the parallel application by choose carefully the parallel programming model, because that has a vital role in the application performance. In this paper, we focused on how to build an algorithm based on processors affinity and the number of communications between processes to handle the data movements as well as minimize the power consumption. In addition, we built a hybrid model of parallel programming models to support our mapping algorithm in several aspects.

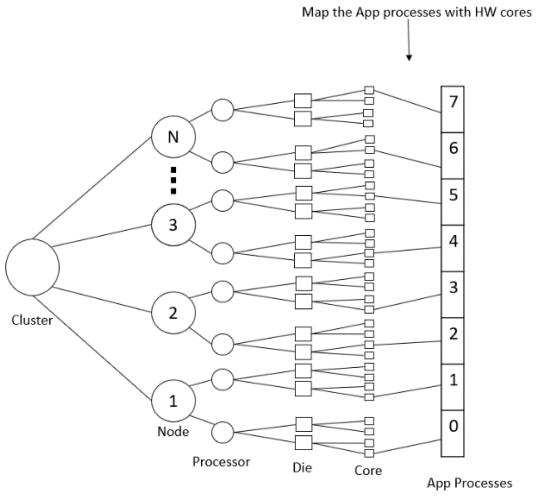

Fig. 7 Match process on processor

3. Paper focus and key contributions

The research and development activity of this paper is focused on the parallel application of the high-performance computing (HPC). Precisely, the topology-aware process mapping that can maps the virtual topology of the parallel application (communication patterns) onto the physical topology of the target machine. Moreover, this paper focused on the parallel programming models and constructed a new hybrid model that combines three parallel programming models including: OpenMPI, OpenMP and OpenACC. We summarize our key contributions of this paper in the following points:

- Develop a tracing algorithm which analyzes the parallel application for identifying all the messages transferred between the

processes. The results of this algorithm help us to know the processes that has a highly communication to map it on the processors that physically closely to each other.

- Develop a dynamic physical topology based on the operating system of the target machine. The proposed topology constructed automatically during the runtime. In addition, this topology has all the necessary information related to the hardware assets including: CPUs, GPUs, NUMAs, Cache Levels, Cores, PUs, …etc.
- Develop a dynamic virtual topology based on the number of processes of the target parallel application. This topology constructed during the runtime. Moreover, the topology has many processes related to the target application. The ordering of this processing depends on the results of the tracing algorithm.
- Develop a topology-aware process mapping algorithm based on the previous points. The results of this algorithm as the following:
    - Process (Task) mapped with target Processor (Processing Unit).
    - Bind this task until the processing is finished.
    - Asking the operation system first to check the processors availability such as: Idle or Busy.
    - All tasks are mapped near each other to avoid the cost of data movement.
- Develop a new hybrid parallel programming model to improve the performance of the parallel application that has the two principal methods of parallel computing: distributed memory computing and shared memory computing. This hybrid model combines the following parallel programming models:
    - We have chosen the OpenMP for dealing with threads on the shared memory. Threading is the most popular shared memory programming technique.
    - For the distributed memory, we use the OpenMPI as a message passing interface.
    - For the accelerator we use the OpenACC. The basic approach is to insert special comments (directives) into the code so as to offload computation onto GPUs and parallelize the code at the level of GPU (CUDA) cores.
    - This hybrid model integrated with mapping algorithm to reach our goals and that are improve the performance of the parallel application and reduce the power consumption as well as reduce the communication costs.

4. Related Work

Mapping the processes of parallel application to target processors based on physical topology has a critical role in the performance of the parallel application in current and future generation of high-performance computing. Because the current and future generation of high-performance computing has a very complex hardware topology that effect on the application performance and that has very impact on the power consumption. For that matter, much research has been done to map the processes of the parallel application based on the virtual topology and the physical topology. The following studies were reviewed in the previous paper [2], to begin with, Cyril Bordage et al. [4] proposed a Netloc tool for collecting the physical topology that is integrated with a Scotch practitioner for computing the topology-aware MPI process placement. However, their experiments were based on the fat tree machine. Guillaume et al. [5] has modified the function of the MPICH2 implementation of the MPI_Dist_graph_create for reordering the process ranks of MPI. The objective is to create a map between the hardware topology and the application communication pattern. Nonetheless, this modification is achieved through rank reordering. Bhatele et al. [6] have proposed various heuristics that are based on the hop-bytes metrics for mapping the graphs of irregular communication to the mesh topologies. Mercier et al. [7] built the topology-aware mapping, based on the Scotch library. Generally speaking, they used the virtual topology (The application communication pattern) and the physical topology as a complete weighted graph. Rashti et al. [8] have extracted the network topologies and intra-node using the InfiniBand tools and HWLOC library respectively. To develop the undirected graph with edges that represent the performance of the communication between cores depending on their distances. Then, this mapping technique is executed by the Scotch library. Ito et al. [9] have proposed a similar mapping technique but using the existing bandwidth between the nodes measured at the time of execution for assigning the edge weights in the graph of the physical topology. Again, the method of this mapping technique was implemented by the Scotch library. Chung et al. [10] proposed an efficient technique based on the hierarchical mapping which partitions the physical topology graphs and the process into numerous super nodes. Also, the very first mapping assigns process topology graph super nodes to the equivalent peers in the graph of the physical topology. These approaches use a graph partitioning to solve the matching between the physical and virtual

topology. Actually, we use in our algorithm a tree method to map the process to processor during the runtime and no need to use graph partitioning before the execution. Moreover, our algorithm is dynamic and it's work with any application without any modification. In addition, our algorithm integrated with hybrid model as a tri-model which is supportive of the parallel application during runtime, which led to increase the performance and reduce the power consumption. There is another approaches of mapping process such as MPIPP [11] that provides a method of configuration file guidance that automatically finds the best match between MPI procedures and resources, thereby minimizing the cost of peer-to-peer communication for random message migration requests. MPIPP uses tools to automate the mapping process to investigate hardware topology maps, so users can generate optimized maps without knowing the application or board system. This study deals only with peer-to-peer communications. Joshua et al. [12] have proposed a Locality-Aware Mapping algorithm to distribute the parallel application processes across processing resources in the high-performance computing system. This algorithm is capable of dealing with both, heterogeneous and homogeneous hardware systems. In the final analysis, they implemented it on the OpenMPI. K. B. Manwade et al. [13] proposed a novel technique known as a "ClustMap" for mapping the application and system topologies. Jingjin Wu et al. [14] proposed a strategy for the mapping of the hierarchical task that implements inter and intra node mapping. They considered supercomputers with torus network and fat-tree topologies, additionally providing two mapping algorithms. The first can deal with both inter-node and intranode mapping. The second can partition the nodes of the computation regarding its affinity. To compare these studies with our work, we have implemented our algorithm to deal with collective communications instead of peer-to-peer only. Moreover, these studies focus on the MPI only and we have developed hybrid model to deal with OpenACC, OpenMP and OpenMPI. In addition, our technique work with runtime and no need to configure or do specific setups as a static mapping. Finally, we have developed our approach as a library that can be used in any application without code modification.

5. Proposed Technique

Nowadays, we observe the number of nodes and the number of cores within the node are increasing, as well as the complexity of the memory hierarchy and the higher parallelism. This makes the high-performance computing systems becoming very complex, which leads to the increasing in the heterogeneity levels in many aspects, including the number of nodes, cores, memories, NUMAs, GPUs, CPUs and others. As a consequence, the processes of the parallel application (communication patterns) need to adapt and map carefully to this hardware topology for optimum execution as well as manage the cost of the data movement. Because the cost of the data movement is the important factor in modernistic HPC. So that, it's necessary to find an optimal solution for reduce the cost of the data movement and power consumption as well as improve the parallel application performance. For that reason, we have developed a new technique combines the hybrid models of parallel programming models and the mapping technique that map the process to processors. The following subsection explain this technique in details with the results and discussion. The mapping technique is divided into four main sections, including: tracing algorithm, virtual topology (Communication Patterns) of the parallel application, physical topology (Hardware Information) of the target machine and the mapping algorithm (Matching virtual topology onto physical topology), the following sections explain it in details.

5.1 Tracing Algorithm

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 tracing algorithm without editing on the main code for detecting and counting all the messages that happened between all the processes whether on peer-to-peer or collective communication. 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 result of this algorithm as the following table "table1": -

Table 1: Final Tracing Algorithm Result

Sender	Receiver	# of MSGs
P2	P9	120
P4	P1	106
P3	P5	100
P6	P8	98
P8	P7	93
P7	P0	88
P0	P10	85
P11	P10	78
P10	P7	72

The tracing algorithm deal with collective and point-to-point communication between the processes of the parallel application. We have built this algorithm as a library on C language for easier use in any parallel application without any configuration or complicated setups. Actually, just need the name of the library at the top of the program code, “trace.h”. The main idea of tracing algorithm is work as a middleware between application execution and call of MPI communication functions. The following diagram explains the algorithm's steps “figure 8 and figure 9”.

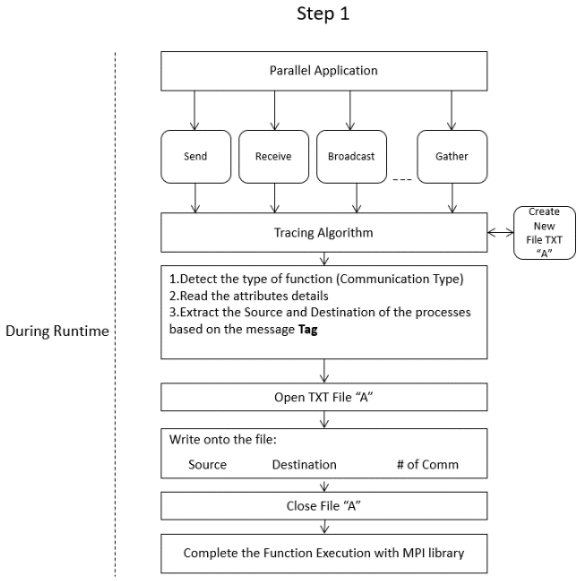

Fig. 8 Step 1 of Tracing Algorithm

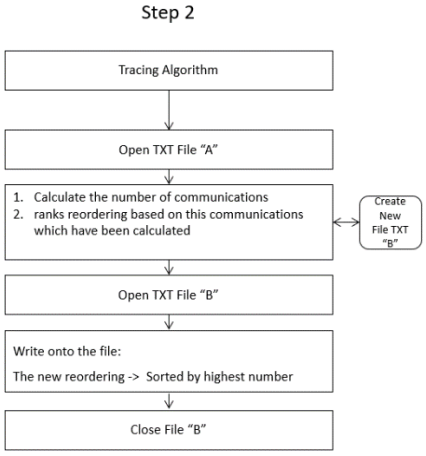

Fig. 9 Step 2 of tracing algorithm

Correspondingly, result of the tracing algorithm is the file with a new ranks reordering. This new ordering used by mapping algorithm for map the processes that has high communication near each other physically.

5.2 Virtual Topology

In reality, most of studies depend 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 section explains the methodology of the proposed virtual topology.

5.2.1 Methodology

First of all, we detect the processes number of the parallel application. Depending on the number of processes, the virtual topology is built with the following feature:

- Periodic = true: A logical matrix of size ndims that indicates whether the grid is periodic, to make the grid more flexible
- Reorder = true: ranks can be reordered, to support the mapping algorithm.

After that, we spilt this virtual topology onto several groups and each group has the processes that has the highest communication. Now, the new groups have a new ordering of ranks. The following points illustrate the steps of creating the virtual topology.

- Detect the number of processes (Parallel Application Processes), EX: 12
- Then, Create the Cartesian Topology as following:
  - MPI_CART_CREATE (comm_old, ndims, dims, periods, reorder, comm_new)
    - comm_old is the input communicator or the current communicator.
    - Ndims is the dimensions of the Cartesian grid, based on the number of processes.
    - Dims is an integer matrix of size ndims, indicating the number of processes per dimension
    - Periods is a logical matrix of size ndims that indicates whether the grid is periodic (true) or aperiodic (false) on each dimension.
    - Reorder is the number of IN reorder identifiers can be reordered (true) or not (false)
    - Comm_new is the new comm of the Cartesian virtual topology
  - MPI_CART_CREATE, return a new communicator and that linked with the information of the cartesian virtual

topology. If reorder = false, then the number of IDs for each process in the new group is the same as the number of IDs in the old group, otherwise, the function will renumber the process (perhaps choosing a good way to embed the virtual topology into the physical machine). If the total size of the grid (Cartesian grid) < size of the COMM group, then some of the processes will returns MPI_COMM_NULL. If its > the size of the COMM group, the result of the call is error.

○ After that, we built the following code to create the Cartesian virtual topology - (Here we take 12 processes as an example), "figure 20" show the virtual topology of 12 processes.

```
MPI_Comm grid_comm;
int dim_sizes[2];
int period[2];
int reorder = 1;
dim_sizes[0] = 4;
dim_sizes[1] = 3;
period[0] = 1;
period[1] = 1;
MPI_Cart_create(MPI_COMM_WORLD, 2, dim_sizes, period , reorder, &grid_comm);
```

0	1	2	3
4	5	6	7
8	9	10	11

Fig. 10 Virtual topology with 12 processes

- Now, we have a virtual topology with 12 processes, after that we read the file of the ranks reordering, because this file contains the processes that have highest communication between each other's.

  ○ Based on the result of tracing algorithm, we divide the virtual topology onto several groups. Each group contains the processes that have highest communication. The following figure "figure 21" explain exactly the method mentioned above after reorder the processes based on tracing algorithm result:

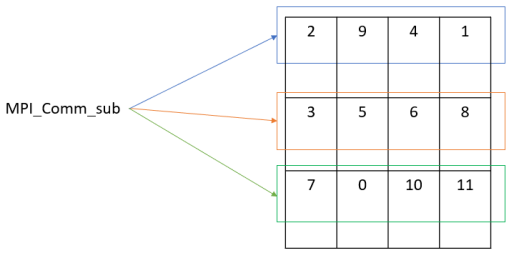

Fig. 11 Subgroups

○ After that, this groups will be ready to map it onto physical topology based on mapping algorithm (Explained in next section). This groups have a new communicator.

- Now, we need to explain the hardware topology to prepare it for mapping with this virtual topology.

5.3 Physical Topology

Actually, gathering the information of target hardware 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 [3], 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. The proposed topology, depends on the operation system of the target machine to detect the processors are Idle or Busy. The following section explains the methodology.

5.3.1 Methodology

We develop the physical topology as a tree where the processing unit in the leave. 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. We have detected and developed the tree by HWLOC, "figure 12" show the tree structure

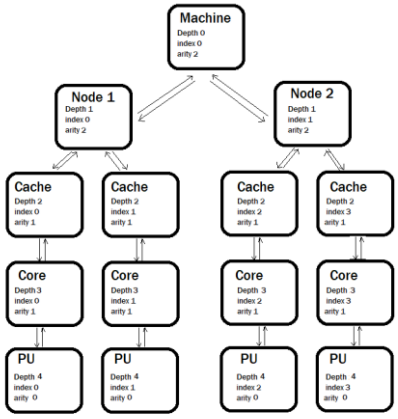

Fig. 12 Physical Topology – Tree

Finally, the mapping algorithm selects the leaves of the physical topology for mapping processes to these leaves. The mapping algorithm is described in the following section.

5.4 Mapping Algorithm

In this paper we present algorithm and practical techniques to accomplish the topologies mapping. In our approach, the first step is to obtain data regarding hardware and the communication pattern of the parallel application and then calculating and creating the matching between the topologies. The above objective can be accomplished by two different methods known as core binding and rank reordering. The algorithm of core binding decides on the physical core where a given MPI process is to be located. Rank reordering algorithm creates a new communicator in which the information specific to the application is attached to the communicator. In the proposed method, we creating a new communicator by reordering the process and mapping these processes to the processors. The function in this approach receives three arguments that make the mapping. These include target, destination, and weight. The objective function and the optimization goal in the study was the reduction in communication costs and power consumption. The following flowchart explains the proposed algorithm "figure 13".

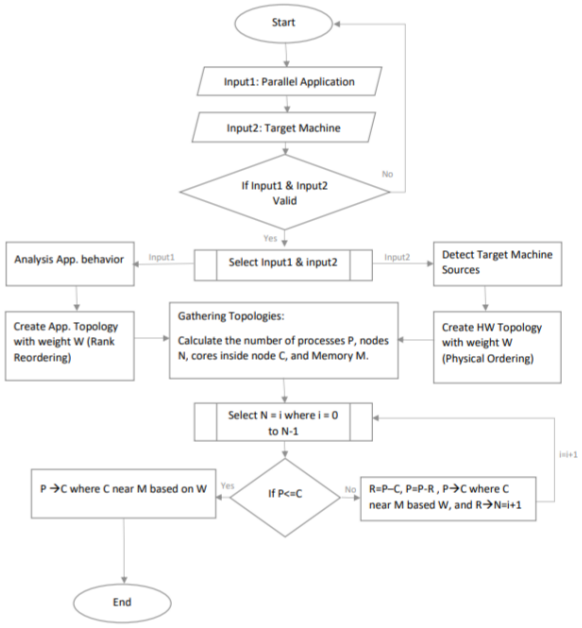

Fig. 13 Mapping Algorithm Flowchart

5.4.1 Methodology

We have developed process-to-processor mapping algorithm as a library based on C language, to facilitate its use in any parallel application without modification or special configuration. Just need to add name of the library in top of the application and use one function as well as one argument to do the mapping process (Function: Mapping() – Argument: Ranks or Process IDs). The proposed mapping algorithm has two important inputs including Process IDs and Processors index. Process IDs will be ready after tracing algorithm that detect the IDs which have a highest communication. Processors will be gathered based on HWLOC and based on its location in the physical topology as a physical ordering or logical distance. The mapping algorithm deals with the physical topology as a tree and leaves are the processing units and their parents are cores which is used by the algorithm for mapping. In addition, deals with the virtual topology as groups like a tree and each group has rank IDs which is used by the algorithm for mapping. The pseudocode of the algorithm as follows:

Algorithm: Processes-To-Processors Mapping

Input: Processes Number p, Core Number C, Node Number N, Cache Index CI n

Output: $\sum_{i=0}^{n} Pi \rightarrow Ci$

```
1   P = 0; //Process ID start with 0
2   C = 0; //Core index start with 0
3   N = 0; //Node index start with 0
4   CI = 0; //Cache index start with 0
5   i = 0, j = 1, r = 0; // Counters
6   Select N = i;
7   While ( i < P )
8       Pi → Cr;
9       if (Pi == Last)
10          Break;
11      End
12      if (Cr == Last)
13          Select N = j;
14          r = 0;
15          j++;
16      End
17      i++;
18  End
```

Finally, the proposed algorithm was tested many times and we obtain a good result in both reduce the power consumption and improve the performance of parallel application. In this proposed technique, we also develop a technique that supporting the application performance which is a hybrid models of the parallel programming models including: OpenMPI, OpenMP and OpenACC. the following subsection explains this technique in details.

5.5 Tri-Model

The development of science and technology has greatly promoted the progress of computational science. A new generation of computers is superior in computing power and computing speed than previous computers, but human requirements for computing are constantly increasing. In practice, some single processors do not meet the

requirements of some engineering calculations well, so in addition to increasing the computational performance of the processor itself, parallel computing is an effective way to improve computing power. Computer clusters provide a good platform for such parallel computing. A computer cluster system is a cluster computing system that combines multiple computers to work together to provide powerful parallel computing capabilities. It uses existing computer resources to allocate heavy tasks to various computing nodes for processing. The current mainstream cluster system is the SMP (Symmetric Multiprocessor) cluster system. They are multi-level architectures consisting of SMP nodes with multiple processors and a fast network connecting the nodes. However, at present, the parallelism between SMP nodes and the parallelism between SMP Cluster nodes and the influence of hyperthreading on SMP Cluster system parallel programming should not be very thorough, and this is precisely It is the basis for us to effectively parallel programming and effectively improve the efficiency of parallel program execution. The hybrid programming is considered to be the most suitable programming model for most types of high-performance computing in terms of increasing performance. In this paper, we have integrated and configured three types of parallel programming models to support the mapping algorithm in terms of increasing the performance and reduce the power consumption. So that, to better utilize the hardware performance of multi-core processors, the (tri-model) hybrid parallel programming model is proposed in terms of distributed and shared memory computing. We have chosen the OpenMP for dealing with threads on the shared memory. Threading is the most popular shared memory programming technique. In the threading model, all the resources belong to the same process. Threads, they share a common address space and system resources. The common shared memory access makes it easy for a developer to divide up work, tasks, and data. The following figure "figure 14" explain the threading with shared memory by OpenMP.

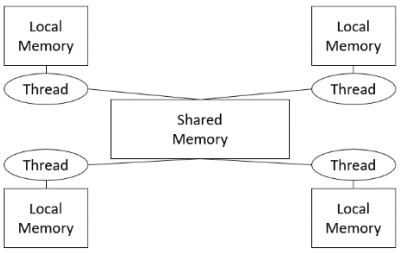

Fig. 14 OpenMP threading with shared memory

For the distributed memory, we use the implementation of the standard MPI which is OpenMPI. MPI is the common messaging programming standard. It is not a programming language. It is a messaging library and API interface bound to Fortran or C language. Its purpose is to serve interprocess communication. The more details in the methodology in the following section. The following figure "figure 15" explain the message passing between machines or nodes by OpenMPI.

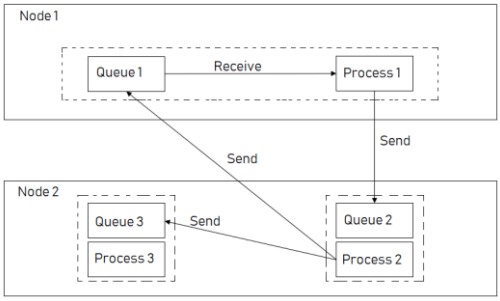

Fig. 15 exchange messages between nodes

For the accelerator we use the OpenACC. The OpenACC interface offloads code from the host CPU to an accelerator device. The programming model presented here are portable for a variety of operating systems, various types of host CPUs, and various types of accelerators. In other words, OpenACC is a set of compiler directives that specify the portion of the C/C++ or Fortran code that is accelerated by the connected accelerator as GPU. It follows the same philosophy as OpenMP and creates an advanced host + accelerator program without managing the accelerator programming language. The following figure "figure 16" explain the OpenACC directive that we used.

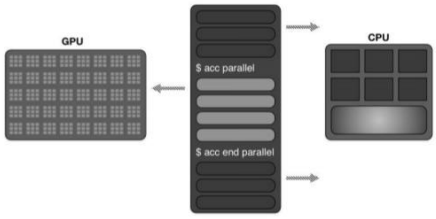

Fig. 16 OpenACC execution model

5.5.1 Methodology

A hybrid MPI+OpenMP+OpenACC parallel programming model for supporting mapping algorithm in heterogeneous high-performance computing hardware architecture is proposed. The model combines the advantages of three programming models of messaging, shared memory and the compiler directives that deals with a specific hardware to achieve better performance. The main step in the integration phase is the configuration of the OpenACC with MPI and OpenMP through the terminal of the Linux OS. These three parallel programming models working together based on the GCC and PGI compilers. The MPI is the standard code

for any parallel application, for that matter we have integrated the OpenACC and the OpenMP inside the OpenMPI code. In addition, any repeat operations such as: While, For, ...etc. will be performed by OpenACC directives, also any threads operations will be performed by OpenMP. For exchanges messages will be performed by OpenMPI. The following pseudocode explains the steps of work for hybrid model.

Algorithm: Hybrid Model Steps

Input: MPI parallel programming model, file of parallel application code

Output: Tri-model (Hybrid of parallel programming model)

```
1   i = 0; //Counter
2   While (file! = end)
3       Get line(i);
4       if (line == for || line == while)
5           Add OpenACC Directive of loop;
6           Compile this code on GPUs;
7       end
8       else if (line == thread)
9           Add OpenMP instruction of threads;
10      end
11      else
12          Compile this code on CPUs;
13      end
14      i++;
15  end
```

Finally, we have obtained a good result after combining the previous hybrid programming model (Tri-model) and mapping algorithm in terms of improve the performance and reduce the power consumption, and we explain the results in the following section.

6. Experimental Results and Discussion

The experiments in this paper were performed on a cluster with multicores platform (12 processors). We have tested the proposed algorithms and the hybrid model more than 50 times, and the algorithms 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 70 percent. All the parallel programming was tested with 12 and 6 processes. We observed big reduction in the power consumption as well as time and that refer to four reasons. The first one, the processes of the parallel application were executed near together because the mapping algorithm maps the processes physically close to each other. 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 exchange the data. The third reason, the powerful that coming from the hybrid model after integrate the three parallel programming models which combine: OpenMPI, OpenMP and OpenACC. The last reason, the algorithm ensures the processing of the parallel application process was bound on the specific processor until the processing accomplished. The following tables and graphs shows the experimental results with six different tests “table2” and “figure 17”.

Table 2: Experiments Results

Testing Number	Parallel Programming	Technique Based
# of Processes	12	12
Testing #1		
Execution Time	92.55 s	14.02 s
Power Consumption	31 w	6 w
Testing #2		
Execution Time	95.29s	13.7s
Power Consumption	32 w	9 w
Testing #3		
Execution Time	96.85 s	14.20 s
Power Consumption	34 w	8w
Testing #4		
Execution Time	95.51 s	14.03 s
Power Consumption	37 w	8 w
Testing #5		
Execution Time	95.28 s	13.96 s
Power Consumption	35 w	7 w
Testing #6		
Execution Time	95.53 s	13.87 s
Power Consumption	34 w	8 w

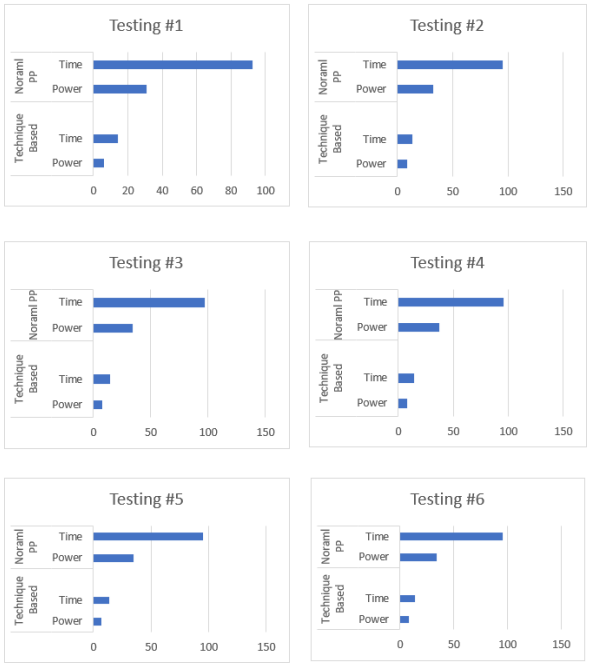

Fig. 17 Results

We have developed another parallel application and we have checked the execution time with mapping algorithm only, and second time with hybrid-model only, and third without any technique the pure parallel application, and the last time with the proposed technique (mapping algorithm based on the tri-model) and we have got excellent results. The following figures shows the parallel application execution with the four previous tests “figure 18”, “figure 19” and “figure 20”.

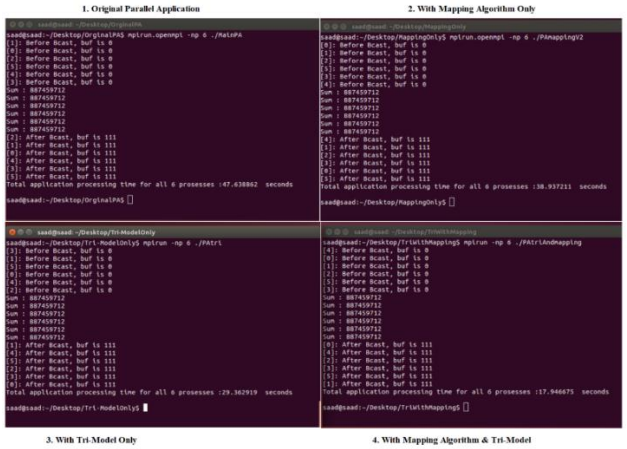

Fig. 17 Test 1 for each type

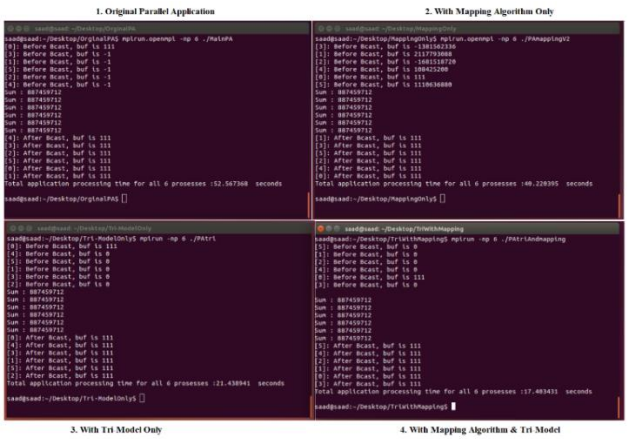

Fig. 19 Test 2 for each type

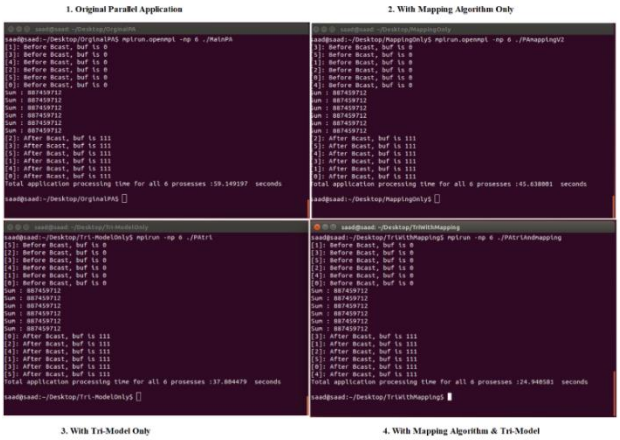

Fig. 20 Test 3 for each type

As a matter of fact, we have got valuable results after include the proposed technique in the parallel application in terms of reduce the power consumption and increase the application performance.

7. Conclusion

The continuous increase of the scale of high-performance computer systems has made the problem of parallel application adaptation very difficult in terms of processes communications. The mapping optimization method between virtual topology of the parallel application and physical topology of the target machine can improve the communication efficiency of applications, and has become one of the research hot topics of high-performance computing. The traditional process mapping optimization model has low mapping efficiency, and it is easy to destroy the integrity of communication-intensive process clusters. To this end, a mapping technique model is proposed in this paper. Based on this model, a novel process mapping algorithm based on tri-model is proposed. Firstly, the tracing algorithm is used to analysis the process communications, then the virtual topology and the physical topology is created. Finally, the mapping algorithm make the process-to-processors matching. this mapping algorithm works with the tri-model to reach our goals which are reduce the power consumption and improve the parallel application performance. We have implemented this technique and we got very impressive results.

References

[1] S. H. Mirsadeghi and A. Afsahi, "Topology-Aware Rank Reordering for MPI Collectives," 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, 2016, pp. 1759-1768.

[2] Saad B. Alotaibi and Fathy Alboraei, (2018) Topology-Aware Mapping Techniques for Heterogeneous HPC Systems: A Systematic Survey, International Journal of

Advanced Computer Science and Applications, Vol. 9, No. 10.
[3] F. Broquedis et al., "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, Pisa, 2010, pp. 180-186.
[4] Cyril Bordage, Clément Foyer, Brice Goglin. Netloc: a Tool for Topology-Aware Process Mapping. Euro-Par 2017: Parallel Processing Workshops, Aug 2017, Santiago de Compostela, Spain.
[5] G. Mercier and E. Jeannot, Improving MPI Applications Performance on Multicore Clusters with Rank Reordering, EuroMPI, p.3949, 2011.
[6] Bhatel´e and L. V. Kal´e. Heuristic-based techniques for mapping irregular communication graphs to mesh topologies. In International Conference on High Performance Computing and Communications (HPCC),pages.765–771,2011.
[7] G. Mercier and J. Clet-Ortega. Towards an efficient process placement policy for MPI applications in multicore environments. In Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI), pages 104–115. 2009.
[8] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp. Multi-core and network aware MPI topology functions. In Proc. European MPI Users' Group Meeting (EuroMPI), pages 50–60, 2011.
[9] S. Ito, K. Goto, and K. Ono. Automatically optimized core mapping to subdomains of domain decomposition method on multicore parallel environments. Computers & Fluids, 80(0):88–93, 2013.
[10] H. Chung, C.-R. Lee, J. Zhou, and Y.-C. Chung, "Hierarchical mapping for HPC applications," in Proc Workshop Large-Scale Parallel Processing, 2011, pp. 1810–1818.
[11] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, "MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters," in Proceedings of the 20th annual international conference on Supercomputing, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 353–360.
[12] J. Hursey , J. M. Squyres , T. Dontje, Locality-Aware Parallel Process Mapping for Multi-core HPC Systems, Proceedings of the 2011 IEEE International Conference on Cluster Computing, p.527- 531, September 26-30, 2011
[13] K. B. Manwade and D. B. Kulkarni, "ClustMap: A Topology-Aware MPI Process Placement Algorithm for Multi-core Clusters", in Intelligent Computing and Information and Communication, Jan 2018, pp. 67-76
[14] Wu, Jingjin and Xiong, Xuanxing and Lan, Zhiling "Hierarchical Task Mapping for Parallel Applications on Supercomputers", in J. Supercomput, 2015, pp. 1776-1802