The Journal of Supercomputing (2023) 79:10423–10441
https://doi.org/10.1007/s11227-023-05085-7

PAARes: an efficient process allocation based on the available resources of cluster nodes

J. L. Quiroz-Fabián[1] · G. Román-Alonso[1] · M. A. Castro-García[1] · M. Aguilar-Cornejo[1]

Accepted: 27 January 2023 / Published online: 8 February 2023

Abstract

The process allocation is a problem in high performance computing, especially when using heterogeneous architectures involving diverse performance characteristics such as number of cores and their frequencies, multithreading technologies, cache memory etc. In order to improve the application performance, it is necessary to consider which processing units are the most suitable to execute the application processes. In this paper, PAARes (Process Allocation based on the Available Resources) strategy is implemented that automatically collects the system information for the process distribution by considering the processing capacity of each node in a cluster and their available resources. To demonstrate the efficiency and efficacy of the proposed strategy, the NAS (NASA Advanced Supercomputing) parallel benchmark is run on homogeneous and heterogeneous clusters under both dedicated and non-dedicated environments.

Keywords Process allocation · Message passing interface · Load distribution · Parallel computing · Cluster

✉ J. L. Quiroz-Fabián
jlqf@xanum.uam.mx

G. Román-Alonso
grac@xanum.uam.mx

M. A. Castro-García
mcas@xanum.uam.mx

M. Aguilar-Cornejo
mac@xanum.uam.mx

[1] Universidad Autónoma Metropolitana, Mexico City, Mexico

1 Introduction

In recent years, the use of computer clusters and the Message Passing Interface (MPI) tool have been proven to solve large processing simulation problems such as: molecular dynamics [1–3], particle diffusion [4, 5], sinoatrial node cells synchronization [6], porous networks behavior [7, 8], machine learning [9, 10], among many others. This is mainly due to the decreasing price of computing infrastructure and the increasing processing capacity.

It is common for clusters to initially maintain a homogeneous infrastructure; however, in order to keep clusters updated, new infrastructures haven been frequently incorporated to make them heterogeneous. Due to this fact, when a parallel application is executed on a cluster it is convenient to use the most suitable nodes, in order to reduce response time. This leads to solve a task allocation problem [11, 12] where the most appropriate processors should be selected to execute specific application processes.

The process allocation to cluster processors can be as simple as blindly assigning a process to a single core randomly selected; however the response time can be affected while selecting nodes with scarce available resources. Another general algorithm for process allocation is the one used when running an MPI application where processes are assigned to processing units following a cyclic order looking for the load balance. More than one MPI process should be allocated when number of them is larger than the available processing units. However, this solution may have a problem if the slowest processor has to run processes that require a high processing time. On the other hand, many hardware characteristics and load information criteria could be considered for processor selection, such as: number of logical cores (multithreading technology), different levels and sizes of cache memories $L1$, $L2$, and $L3$, Random Access Memory (RAM) organization—Uniform Memory Architecture (UMA) or Non Uniform Memory Architecture (NUMA)—, processing speed, communication latency, etc. However, considering all these elements and resources can result in a complex allocation algorithm, requiring specific information provided by users about the behavior of their applications (for example, the communication graph or a processing cost estimation) or the cluster characteristics.

In this paper we propose *PAARes (Process Allocation based on the Available Resources)*, an algorithm for process allocation on cluster nodes that takes into account the following resource information of each node: processor type (physical or logical cores—Multithreading—), frequency (processor speed), cache memory (all levels, for example $L1$, $L2$, and $L3$), and load (number of running processes). Based on the aforementioned information, the nodes are ordered considering their processing availability. The nodes having higher processing availability are prioritized by placing them first and the overloaded are at last of a list. The PAARes algorithm works in three phases, firstly, it automatically and transparently gathers information about the resources of each node, avoiding the need of a user to provide any information. Secondly, a set of keys is generated for comparing the characteristics of different nodes. Thirdly, the construction of an ordered node list is carried out and finally, the generated list is used for the allocation of the processes in the nodes.

To compare the performance of our approach, PAARes is integrated in the distribution of processes performed by *OpenMPI* using the applications of *NPB (NAS Parallel Benchmarks)* [13]. Using PAARes, before each execution, the nodes with the lowest load or the highest processing capacity are selected, in order to reduce the execution time when an MPI application is executed.

The rest of the document is structured as follows: Sect. 2 presents the state of the art of process allocation algorithms. In Sect. 3, the proposed PAARes algorithm is described. Section 4 shows the experimental setup considered for comparing our proposal performance. In Sect. 5 the obtained results are presented, and finally, Sect. 6 includes the conclusions and future work.

2 Related works

In order to reduce the execution time of cluster applications, the processes distribution problem has been studied in several works. Some blind allocation policies do not consider any information to choose the processing unit where a process will be executed.

In [14, 15] the selection of processors is considered based on the requirements of the user application, such as a specific number of nodes, the number of processors per node, and a minimum amount of memory. If the required resources are available, one process is assigned per each processing unit to run the application, and otherwise, the application is not executed. The main disadvantage of this allocation policy is that heavy processes are not necessarily assigned to the nodes with higher processing capacity, but to those that comply with the requirements. Another example of this type of allocation is found in MPI implementations.

Most MPI distributions use by default a cyclic *Round Robin* policy, where an assignment can be carried out by *Processing Units (PU)* or by *Nodes* [16–18] selection. Using an assignment generated by *PU* selection, a process per processor or core on a single node is usually assigned, if there are more processes to allocate they are distributed to the next node, and so on until all processes are allocated. In an assignment completed by *Nodes* selection, a process per each cluster node is allocated, if the number of processes is greater than the number of nodes the procedure is repeated several times until all processes are allocated. Both policies are blind since they use a list of node IP addresses or node names provided by the user to establish the processes distribution order which is sequentially read by MPI to select a node.

Other works [19, 20] take into account theoretical models, assuming that there is prior knowledge of the processes behavior such as the time in which a process is incorporated into the cluster and the execution time of each process. This information is used to define an execution configuration reducing the response time of the cluster processes on average. Although theoretical models look for obtaining optimal assignments, it is often difficult (or impossible) to have the information they refer to (such as the execution time of a process); besides, the optimal allocation of resources is known to be a problem with computational complexity (NP-Complete problem [21]).

Approaches such as [22–25] consider the way in which the processes of an application communicate, allocating in the same processor the processes that communicate more frequently, thus decreasing network communications and reducing latency. In [22, 25] a parallel application is first executed at least once to trace the processes interaction and build a communication graph. The processes are finally assigned based on the level of communication they maintain, following the graph information. In [23] the assignment of processes to cluster nodes is done considering the number of network cards that each node has. The processes are assigned by network card looking for the balance of communications and intending to avoid bottlenecks. In [24] a logical tree is generated to represent the cluster hardware (nodes, cores per node, etc.) where the cores correspond to the leaves of the tree. Then, a communications map is defined where the interaction (communication) between the application processes is represented. Processes that carry out more communications form groups that are assigned to nodes based on the generated tree. Assigning them to the same node strengthens local communication and reduces cost. Although this solution helps reduce latency, it is not practical as it requires the user to provide information that is not always available or easy to obtain about how processes interact.

We can also find some assignment policies that analyze more specific information about the available hardware to decide processes assignment and improve the performance of applications. Some MPI distributions (for example [26–29]) consider the hardware of a cluster, allowing a range of possible assignments of processes based on the characteristics and quantity of cluster nodes, processors, cores, cache memory levels $L1$, $L2$ or $L3$ (without considering their sizes, only if they are present), or the memory architecture of nodes (UMA or NUMA). While these works consider different aspects of hardware, they have two main disadvantages. The first disadvantage is that these assignments are very dependent on the specification of the required hardware; for example, if a process defines the restriction of using $L3$ cache to run on one of the most recent computing hardware, the nodes that do not have the required cache level will not be considered for collaboration even though they may have the best available processing resources. Another drawback is that developers must indicate (and know) the hardware characteristics required for execution. The second disadvantage is that if there is a multiprogramming environment where a node executes processes from different applications, the assignment of processes made by these MPI distributions does not consider the overloading that may exist in some nodes.

In this paper we propose a process allocation strategy called *PAARes* (Process Allocation based on the Available Resources) which is based on the construction of a node list where nodes of a dedicated or non-dedicated cluster are sorted considering their available processing capacity. It is achieved by collecting the hardware resources and the load state information of each node to determine its processing capacity. The assignment of processes to nodes is then guided by the ordered node list. In the next section this algorithm is described in detail.

3 Process allocation proposal

The purpose of this algorithm is to decrease parallel application execution time by determining a process allocation based on the generation of a properly ordered list of cluster nodes. Considering each node available processing and cache capacities; the nodes with higher processing capacity are at the beginning of the list while the slower nodes are located at the end of the list. Algorithm 1 shows the general PAARes algorithm that should be performed when an MPI parallel application is going to be launched, the list parameter is a list containing the names of the existent cluster nodes.

Algorithm 1 General behavior of the PAARes proposal.

```
1: procedure PAARes($list$)
2:     for each $n_i \in list$ do                                  ▷ First and Second phase
3:         $info \leftarrow$ gatherNodeInformation($n_i$ )
4:         calculateKeys($n_i$,info, Keys )
5:     end for
6:     Sort_nodes_by_available_resources(list, Keys)               ▷ Third phase
7:     $hostfile \leftarrow$ HostFile_generation(list )
8:     MPI_application_process_allocation(hostfile)
9: end procedure
```

The PAARes strategy is carried out in three main phases or stages which are the information gathering, information analysis, and the process allocation. The first two phases are executed by the lines 2–5 of Algorithm 1. In these phases, a view of the characteristics and load state of the nodes is obtained and analyzed to quantify their processing availability by means of the initialization of four key values per node. The third phase is performed by lines 4–6, which include the ordering of the nodes according to their calculated key values and finally the allocation of the application processes. Each one of these stages are described below.

3.1 Information gathering

A fundamental phase of the PAARes algorithm is the information gathering to obtain a measure of the resources usage and workload state of each node on a cluster. PAARes works on a Linux-cluster and gathers the following information from each node.

1. The number of processing cores
2. The maximum core frequency in MHz
3. Memory cache characteristics
4. The number of running processes.

Fig. 1 Processor information obtained from a *cpuinfo* command

```
VIII

processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 15
model name : Xeon(R) CPU 5160 @ 3.00GHz
stepping : 6
microcode : 0xd2
cpu MHz : 2999.924
cache size : 4096 KB
physical id : 0
siblings : 2
core id : 0
cpu cores : 2
apicid : 0
```

Algorithm 2 details the information gathered to estimate the available resource of a node. This includes the number of physical cores per node (line 2), the number of logical cores (line 3), the maximum frequency of cores (line 4), the amount of cache memory at the different levels (line 5), and the number of running node processes (line 6) which uses more than 5% CPU (most system processes use less than 5%).

L_i is an array that stores the amount of memory in each particular level of cache; currently, most nodes have three cache levels: $L1$, $L2$, and $L3$. For simplicity, all cache memories are considered to have the same frequency or speed, regardless of their level.

Algorithm 2 Gathering information from a cluster node.

```
1: function GATHERNODEINFORMATION($n_i$)
2:     $info.nb\_Pcores \leftarrow$ GET_NB_PHYSICALCORES($n_i$)
3:     $info.nb\_Lcores \leftarrow$ GET_NB_LOGICALCORES($n_i$)
4:     $info.frequency \leftarrow$ GETCOREFREQUENCY($n_i$)
5:     $info.L_i \leftarrow$ GETCACHEMEMORY_SIZES($n_i$)
6:     $info.p\_load \leftarrow$ GETPROCESSES($n_i$)
7:     return $info$
8: end function
```

These data are extracted from operating system commands that describe the hardware. For example, with */proc/cpuinfo*, *lscpu*, or *ps* system commands it is possible to get the amount of cores and their frequencies (Fig. 1 shows an example of the output of *cpuinfo* command). It is worth mentioning that this is not the only way to obtain information of the used resources, commands are continuously being developed to allow monitoring and obtaining system resources information, for example *hwloc* and *numactl* [30, 31].

3.2 Information analysis

After the information gathering of each node, PAARes generates a set of four values or keys which refer to the processing capability available in each node. The larger the key values, the higher the available processing throughput on a node. The four keys generated by PAARes are listed below.

1. key_1: A numeric value that defines the processing capacity of each node. It is calculated by Eq. (1):

$$key_1 = \frac{\left(info.nb\_Pcores + \frac{info.nb\_Lcores}{2}\right) \times info.frequency}{info.p\_load + 1} \tag{1}$$

   This key value considers the physical and logical cores, their frequency and the number of processes currently running on the node. As it is observed, key_1 is calculated in two parts: firstly, the total number of cores is multiplied by the maximum core frequency to obtain a total node processing capacity; although the number of logical cores could be equal to this of the physical cores, they do not obtain the same processing gain as the physical ones; in [32, 33] the reported gains using the logical cores only reached between 30% and 50%. For the above mentioned, in key_1 the number of logical cores is divided by 2 to obtain a gain of 50%. In the second part, a penalty is added by dividing the processing capacity between the number of already running processes on the node, $load_i + 1$. When a large number of processes is being executed on a specific node, key_1 will obtain lower values.
2. key_2: It is defined by the maximum amount of $L3$ cache memory associated with a core. If the kernel does not have this cache type, the value of this key is 0.
3. key_3: This key value is initialized with the maximum amount of $L2$ cache memory associated with a core. If the kernel does not have an $L2$ cache, the value of this key is 0.
4. key_4: The maximun amount of $L1$ cache memory associated with a core initializes this key.

Table 1 shows an example of a cluster with 6 nodes. For each node, the information about physical cores, logical cores, core frequency, cache memory $L3$, $L2$, and $L1$, and the number of processes using more than 5% CPU (Load) is given. In this example, it can be seen that not all nodes have logical cores and $L3$ cache memory (it is a heterogeneous cluster). Moreover, the only node that has a Load value greater than 0 is node 3. Table 2 shows the calculated key values using the algorithm of the last paragraph.

Table 1 Example of a set of 6 cluster nodes and their resources

Nodes	Physical cores	Logic cores	Frequency	$L3$	$L2$	$L1$	Load
Node 1	4	0	3	0	2048	32	0
Node 2	4	0	3	0	2048	64	0
Node 3	10	10	4	1408	1024	32	8
Node 4	4	4	3.4	2048	256	32	0
Node 5	4	0	2.4	0	1024	32	0
Node 6	6	6	3.5	1408	1024	32	0

Table 2 Keys values of the nodes described in Table 1

Nodes	key_1	key_2	key_3	key_4
Node 1	12	0	2048	32
Node 2	12	0	2048	64
Node 3	6.6	1408	1024	32
Node 4	20.4	2048	256	32
Node 5	9.6	0	1024	32
Node 6	31.5	1408	1024	32

3.3 PAARes process allocation

The key values information is taken into account to build a sorted node list. Based on the position of nodes in the list, their processing availability is defined. The first nodes are those considered to have more available resources (therefore the highest processing throughput), and the nodes located at the end of the list are identified as the more overloaded. The PAARes strategy to build the sorted list is given in Algorithm 3.

Algorithm 3 PAARes node ordering algorithm.

```
1: procedure SORT_NODES_BY_AVAILABLE_ RESOURCES(list, Keys)
2:     Sort nodes by $key_1$ (list, Keys);
3:     Sort the nodes with same $key_1$ value by $key_2$ (list, Keys);
4:     Sort the nodes with same $key_2$ value by $key_3$ (list, Keys);
5:     Sort the nodes with same $key_3$ value by $key_4$ (list, Keys);
6: end procedure
```

The ordering of nodes (lines 2–5) in the list considers the four *key* attributes. All nodes are first sorted by key_1 in descending order (line 2), leaving the node with the highest key_1 value at the beginning of the list.

Whether two nodes get the same key_1 value, the key_2, key_3 and key_4 are used to decide which of them should be placed before the other (lines 3–5); in case

Table 3 Example of values of the four keys for each node of Table 2; nodes ordered by the value of key_1

Nodes	key_1	key_2	key_3	key_4
Node 6	31.5	1408	1024	32
Node 4	20.4	2048	256	32
Node 1	12	0	2048	32
Node 2	12	0	2048	64
Node 5	9.6	0	1024	32
Node 3	6.6	1408	1024	32

Table 4 Final arrangement obtained by ordering nodes 1 and 2 by key_4

Nodes	key_1	key_2	key_3	key_4
Node 6	31.5	1408	1024	32
Node 4	20.4	2048	256	32
Node 2	12	0	2048	64
Node 1	12	0	2048	32
Node 5	9.6	0	1024	32
Node 3	6.6	1408	1024	32

all keys have same values, both nodes are placed one after the other, indistinctly, illustrated in Tables 2, 3, and 4.

Table 3 shows the nodes of Table 2 after being sorted by key_1. From the table, it can be observed that node 6 gets the first position (with key_1 value = 31.5) and node 3 the last one (with key_1 value = 6.6) in the list. Here, nodes 1 and 2 get the same key_1 value; in this case the algorithm considers the other key values to find out which of them should be placed before the other. Since both key_2 and key_3 values are the same for node 1 and node 2, the key_4 value is used to decide the final ordering. Node 2 (with $key_4 = 64$) is then placed before node 1 (with $key_4 = 32$), obtaining the results given in Table 4.

After the ordering steps, a file called *hostfile* is generated (line 7 of Algorithm 1) storing the sorted list (only considering the name of the nodes or their IP addresses). Before the execution of a parallel program, the *hostfile* file is read (line 8 of Algorithm 1) to first use the nodes with the highest processing availability found at the beginning of the file, running the application processes following an assignment by PU policy.

4 Experimental setup

The *NAS Parallel Benchmarks (NPB)* [13] contain a set of MPI programs intended to evaluate parallel computers mainly in terms of processing and memory performance. In this work the NPB kernel is used to compare the performance of PAARes *vs* the OpenMPI's default distribution policy which consists of a process mapping by

Processing Units (by slot[1][18, 34]), using a user-provided node list. The application characteristics and the used experimental infrastructure are described below.

4.1 Applications

The characteristics of the five NPB kernel applications are the following:

1. IS (Integer sorted) application: The main cluster challenge executing this program is to perform random accesses in memory.
2. EP (Embarrassingly parallel) application: The objective of this application is to execute independent tasks, that is, processing tasks with very little or no communication among them.
3. CG (Conjugate gradient) method: In this application mathematical calculations and irregular and distant communications are performed.
4. A simplified MultiGrid (MG) kernel: This program performs structured communications[2] which are short and distant, as well as an intensive use of memory.
5. Partial solution using the Fast Fourier Transform (FFT): The objective of this application is to evaluate the communication between all the processes.

Each of these applications contains three different classes (different problem sizes) represented by letters. Between each class, the problem size is increased by 4 orders of magnitude regarding the immediate previous class. For our proposal evaluation classes A (the smallest size problem), B, and C (the largest size problem) are used.

4.2 Infrastructure

The experimental infrastructure consists of two clusters, a homogeneous cluster (cluster 1) and a heterogeneous cluster (cluster 2). The hardware specifications and the operating system of each cluster are shown in Tables 5 and 6. All nodes are connected through a Gigabit Ethernet switch.

The software specifications are: gcc 4.8.5 compiler version, OpenMPI version 1.10.2, and java 1.8.0_25 SDK.

4.3 Test scenarios

The proposed PAARes strategy is compared in two scenarios: dedicated and non-dedicated functioning. In the dedicated scenario NPB application is executed at a time, without the existence of external processing tasks affecting the system performance. In case of a dedicated homogeneous cluster, the selection of nodes to allocate processes is indistinct since they all have the same characteristics and all their processing capacity is available, so PAARes behaves the same as OpenMPI

[1] In OpenMPI a slot is an allocation unit for a process.

[2] Communications where the sender, receiver and message are well defined

Table 5 Characteristics of the Cluster 1 (homogeneous)

Node	CPU Model	Physical cores	Logical cores	Frequency (GHz)	*L*1 (KB)	*L*2 (KB)	*L*3 (KB)	RAM (GB)	Hard Disk	OS
1–6	QuadCore AMD Opteron 2356	8	0	2.4	64	256	12,288	8	HDD 250 GB	Centos 7

Table 6 Characteristics of the Cluster 2 (heterogeneous)

Node	CPU Model	Physical cores	Logical cores	Frequency (GHz)	*L*1 (KB)	*L*2	*L*3	RAM (GB)	Hard disk (GB)	OS
1–5	QuadCore Q6600	4	0	2.4	64	1024	–	4	500	Centos 7
6–8	Dual Xeon 1333	4	0	3	64	2048	–	4	160	Centos 7
9	Core i5-7400	4	0	3	64	256	1536	16	1000	Centos 7
10–12	Intel(R) Core(TM) i7-4770	4	4	3.4	64	256	2048	16	1000	Centos 7
13	Intel(R) Core(TM) i7-7800X	6	6	3.5	64	1024	1408	128	1000	Centos 7
14	Intel(R) Core(TM) i9-7900X	10	10	4	64	1024	1408	32	250	Centos 7

obtaining same performance; for this reason this document only presents the results where a difference is observed. With a dedicated scenario using a heterogeneous cluster PAARes considers the different processing capacities of nodes to build the *hostfile* and place the NPB processes.

The non-dedicated scenario considers a homogeneous/heterogeneous cluster where external interfering processes are being executed in one or more nodes to generate additional load on them. In our experiments the external load was generated by the execution of *stress*[3] processes.

5 Results

In this section, the evaluation of the PAARes strategy versus the default OpenMPI processes distribution executing the NPB MPI applications is given, considering the test scenarios previously presented. The default OpenMPI process allocation algorithm (assignment by slot) considers, for the heterogeneous cluster, a hostfile containing a list of nodes whose order of appearance is given in Table 6, with the most recent nodes at the bottom. For the case of the homogeneous cluster, the hostfile list contains the names of nodes 1 to 6 from Table 5, all of them having the same characteristics.

Each reported NPB application execution time is the average of the three classes (A, B, and C) for the same number of processes. For each test, five comparatives (one per each NPB application) are presented. It is worth mentioning that PAARes does not need to know the intercommunication graph between processes nor the number of calculations of each application process, whether class A, B or C application. To simplify the results presentation, the default OpenMPI processes distribution is named MPI. In each result graph, the number of used processes is varied, in cluster 1 from 2 to 32 and in cluster 2 from 2 to 64 processes.

5.1 Non-dedicated homogeneous cluster

In a homogeneous cluster with some nodes overloaded through the execution of more than one application, PAARes first tries to allocate processes based on the least loaded nodes and then on the most loaded ones. Figure 2 shows five graphs (one per each NPB application) plotting the execution times obtained by PAARes and MPI. Here, only one node has extra load executing one stress process. As described in Sect. 4.1, each application in the NPB has different characteristics; we can see that for all application types PAARes obtained shorter times than the default MPI distribution, i.e., 1.9% (IS with 32 processes) and 73.5% (CG with 2 processes) less execution time.

[3] Process that generates load on the CPU performing floating point operations.

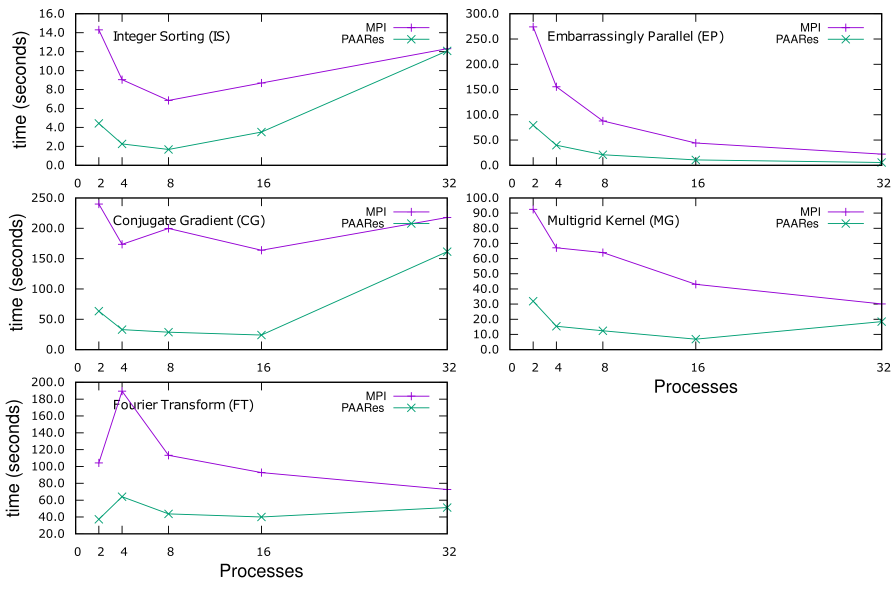

Fig. 2 Average execution for the NPB applications on the non-dedicated homogeneous cluster (cluster 1)

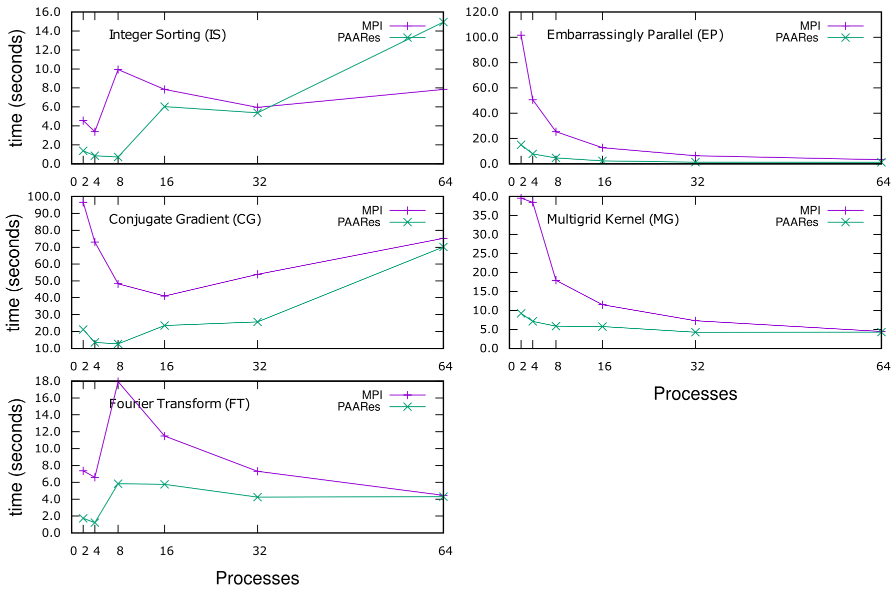

Fig. 3 Average execution for the NPB applications on the dedicated heterogeneous cluster (cluster 2)

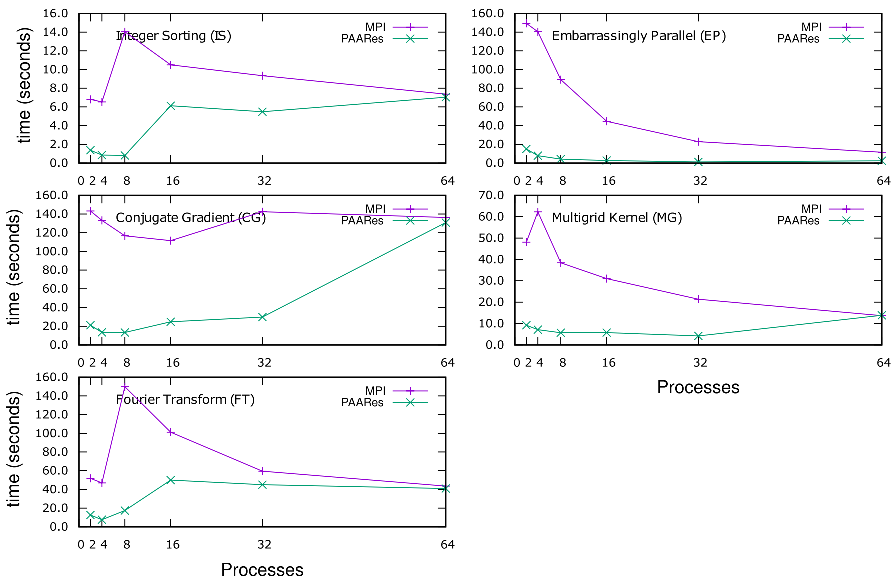

Fig. 4 Average execution times for the NAS applications on the non-dedicated heterogeneous cluster (cluster 2)

5.2 Dedicated heterogeneous cluster

In a dedicated heterogeneous cluster, PAARes first selects the nodes with more processing capability in order to obtain better execution time in most cases. Figure 3 shows a comparison between PAARes vs MPI (one per each NPB application) using the cluster 2. In this scenario, for the FT application with 2 and 4 processes, only the problem sizes A and B were executed due to memory constraints that arose while running class C. In general, the execution time of PAARes is better in most NPB applications. As seen in the figure, for 64 processes while using the entire cluster the execution time is almost similar; however, in some cases MPI got reduced times. This is due to two factors, (i) the problem size and (ii) the default nodes organization of MPI. From the beginning, PAARes uses the nodes with the most available resources, while MPI uses them only when the execution involves 64 processes because the fastest nodes are at the end. As the characteristics of the processes regarding the processing or communications to be carried out are not priorly known, it could be the case that the firstly assigned processes have a lower cost than the last ones, resulting in no improvement in PAARes times. However, the results obtained after executing the NPB applications demonstrate that this is a rare case which only happen when all the cluster cores are used.

5.3 Non-dedicated heterogeneous cluster

A non-dedicated heterogeneous cluster adds another parameter to consider in process allocation which in turn increases the degree of heterogeneity. Since the nodes have different processing capabilities, the fastest ones will not be at the top of the list if they are executing extra load and the slowest ones will not be at the bottom if they are lightly loaded. Figure 4 shows a comparison between PAARes and MPI (one graph for each NPB application) using the cluster 2, with 50 % overloaded nodes in order to obtain a higher degree of heterogeneity. For the FFT application with 2 and 4 processes, only the problem sizes A and B are averaged. Similar to the previous results, in most cases PAARes results in improved timing. It can be observed that when 64 processes are used, the results are very similar.

It is worth mentioning that the results in the case of MPI will depend on how the programmer ordered the nodes on the hostfile, while PAARes guarantees that nodes with less workload or more available resources will always be used first due to its 4 keys-based sorting strategy.

6 Conclusions and future work

The allocation of processes is a challenge in applications that require high processing capacity. In this paper, we proposed a process allocation strategy called *PAARes* which works in dedicated and non-dedicated environments. The algorithm proposes the creation of a machine file containing a sorted node list which takes into account the available processing capacity of the nodes, considering the physical and logical cores, their frequency, the different levels of cache memory existing at each node, and the number of running processes. In the algorithm, it is not necessary for the user to provide these information manually, PAARes automatically obtains information. With the collected information, PAARes use four keys to quantify it. These keys are the basic criteria to perform the ordering of processing nodes.

To evaluate the proposal we employed the *NAS Parallel Benchmark* considering its 5 kernel applications executed on homogeneous and heterogeneous clusters. We compared the performance of PAARes with respect to the default distribution of OpenMPI executing the benchmark in dedicated and non-dedicated scenarios. The results show that PAARes give better performance compared to MPI that does not consider the physical and logical node characteristics for process allocation. This allows us to claim that by having a process assignment that takes into account more detailed information about the characteristics and load state of cluster nodes, in most cases it is possible to reduce the execution time of parallel applications without having a previous knowledge of their communication or processing cost.

The future work is oriented to consider other information about the architecture of processors; for example, NUMA architecture, connected I/O devices. In case the information about the communication graph of the parallel applications is available, PAARes could be extended to consider it as an additional key with the aim of improving performance.

Author contributions Dr. José Luis and Dr. Graciela wrote the main manuscript text, Dr. Miguel and Dr. Manuel did the experiments, Dr. José Luis and Dr. Manuel prepared figures, and Dr. Graciela and Dr. Miguel prepared tables. All authors conceived of the presented idea, discussed the results and contributed to the final manuscript.

Funding Not applicable.

Availability of data and materials Not applicable.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable.

References

1. Acun B, Hardy DJ, Kale LV, Li K, Phillips JC, Stone JE (2018) Scalable molecular dynamics with NAMD on the summit system. IBM J Res Dev 62(6):4–149. https://doi.org/10.1147/JRD.2018.2888986
2. Guo Z, Lu D, Yan Y, Hu S, Liu R, Tan G, Sun N, Jiang W, Liu L, Chen Y, Zhang L, Chen M, Wang H, Jia W (2022) Extending the limit of molecular dynamics with ab initio accuracy to 10 billion atoms. In: Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. PPoPP'22. Association for Computing Machinery, New York, pp 205–218. https://doi.org/10.1145/3503221.3508425
3. Morillo J, Vassaux M, Coveney PV, Garcia-Gasulla M (2022) Hybrid parallelization of molecular dynamics simulations to reduce load imbalance. J Supercomput 78(7):9184–9215. https://doi.org/10.1007/s11227-021-04214-4
4. Pérez-Espinosa A, Aguilar-Cornejo M, Dagdug L (2020) First-passage, transition path, and looping times in conical varying-width channels: comparison of analytical and numerical results. AIP Adv 10(5):055201. https://doi.org/10.1063/5.0004026
5. Qiu H, Xu C, Li D, Wang H, Li J, Wang Z (2022) Parallelizing and balancing coupled DSMC/PIC for large-scale particle simulations. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 390–401. https://doi.org/10.1109/IPDPS53621.2022.00045
6. Mata AN, Castellanos Abrego NP, Alonso GR, Castro García MA, Garza GL, God ínez Fernández JR (2018) Parallel simulation of sinoatrial node cells synchronization. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp 126–133. https://doi.org/10.1109/PDP2018.2018.00025
7. Cordero-Sánchez S, Rojas-González F, Román-Alonso G, Castro-García MA, Aguilar-Cornejo M, Matadamas-Hernández J (2016) Pore networks subjected to variable connectivity and geometrical restrictions: a simulation employing a multicore system. J Comput Sci 16:177–189. https://doi.org/10.1016/j.jocs.2016.06.003
8. Ando S, Kaneda M, Suga K (2022) Permeability prediction of fibrous porous media by the lattice Boltzmann method with a fluid-structure boundary reconstruction scheme. J Ind Text 51(4_suppl):6902–6923
9. Pearson C, Javeed A, Devine K (2022) Machine learning for CUDA+MPI design rules. In: 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp 880–889. https://doi.org/10.1109/IPDPSW55747.2022.00144
10. Alemany S, Nucciarone J, Pissinou N (2021) Jespipe: a plugin-based, open MPI framework for adversarial machine learning analysis. In: 2021 IEEE International Conference on Big Data (Big Data), pp 3663–3670. https://doi.org/10.1109/BigData52589.2021.9671385
11. Al-Rahayfeh A, Atiewi S, Abuhussein A, Almiani M (2019) Novel approach to task scheduling and load balancing using the dominant sequence clustering and mean shift clustering algorithms. Future Internet. https://doi.org/10.3390/fi11050109
12. Tyagi R, Gupta SK (2018) A survey on scheduling algorithms for parallel and distributed systems. In: Mishra A, Basu A, Tyagi V (eds) Silicon Photonics & High Performance Computing. Springer, Singapore, pp 51–64
13. Nasa: NASA Advanced Supercomputing Division. https://www.nas.nasa.gov/publications/npb.html#url. Accessed April 2022

14. Feng H, Misra V, Rubenstein D (2007) PBS: a unified priority-based scheduler. SIGMETRICS Perform Eval Rev 35(1):203–214. https://doi.org/10.1145/1269899.1254906
15. Zhao T, Gu J, Zhang X (2021) Two-level scheduling technology for heterogeneous clusters using analytical hierarchy processes. In: 2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS), pp 121–127. https://doi.org/10.1109/ICCCS52626.2021.9449223
16. Intel: Running an MPI Program. https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top/running-applications/running-an-mpi-program.html. Accessed April 2022
17. mpich: Using the Hydra Process Manager. https://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager. Accessed April 2022
18. openmpi: Running MPI jobs. https://www.open-mpi.org/faq/?category=running. Accessed April 2022
19. Li K (2008) Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments. J Syst Architect 54(1):111–123. https://doi.org/10.1016/j.sysarc.2007.04.003
20. Skenteridou K, Karatza HD (2015) Job scheduling in a grid cluster. In: 2015 International Conference on Computer, Information and Telecommunication Systems (CITS), pp 1–5. https://doi.org/10.1109/CITS.2015.7297738
21. Ullman J (1975) Np-complete scheduling problems. J Comput Syst Sci 10:384–393
22. Cao H, Jin H, Wu X, Wu S, Shi X (2010) DAGMap: efficient and dependable scheduling of DAG workflow job in grid. J Supercomput 51(2):201–223. https://doi.org/10.1007/s11227-009-0284-7
23. Ganapathi RB, Gopalakrishnan A, McGuire RW (2017) MPI process and network device affinitization for optimal HPC application performance. In: 2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI), pp 80–86. https://doi.org/10.1109/HOTI.2017.12
24. Jeannot E, Mercier G (2010) Near-optimal placement of MPI processes on hierarchical NUMA architectures. In: Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part II. Euro-Par'10. Springer, Berlin, pp 199–210. http://dl.acm.org/citation.cfm?id=1885276.1885299
25. Jeannot E, Mercier G, Tessier F (2014) Process placement in multicore clusters: algorithmic issues and practical techniques. IEEE Trans Parallel Distrib Syst 25(4):993–1002. https://doi.org/10.1109/TPDS.2013.104
26. Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain RH, Daniel DJ, Graham RL, Woodall TS (2004) Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, pp 97–104
27. Goglin B (2014) Managing the topology of heterogeneous cluster nodes with hardware locality (HWLOC). In: 2014 International Conference on High Performance Computing Simulation (HPCS), pp 74–81. https://doi.org/10.1109/HPCSim.2014.6903671
28. Gropp W (2002) MPICH2: a new start for MPI implementations. In: Kranzlmüller D, Volkert J, Kacsuk P, Dongarra J (eds) Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer, Berlin, pp 7–7
29. Hursey J, Squyres JM (2013) Advancing application process affinity experimentation: Open MPI's lama-based affinity interface. In: Proceedings of the 20th European MPI Users' Group Meeting. EuroMPI'13. ACM, New York, pp 163–168. https://doi.org/10.1145/2488551.2488603
30. Goglin B (2017) On the overhead of topology discovery for locality-aware scheduling in HPC. In: PDP2017—25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2017). IEEE Computer Society, St Petersburg, p 9. https://doi.org/10.1109/PDP.2017.35
31. Goglin B (2018) Memory footprint of locality information on many-core platforms. In: IEEE (ed.) 6th Workshop on Runtime and Operating Systems for the Many-core Era (ROME 2018), Held in Conjunction with IPDPS, Vancouver, BC, Canada, p 10. https://hal.inria.fr/hal-01644087
32. Leng T, Ali R, Hsieh J, Mashayekhi V, Rooholamini R (2002) An empirical study of hyper-threading in high performance computing clusters
33. Marr DT, Binns F, Hill DL, Hinton G, Koufaty DA, Miller AJ, Upton M (2002) Hyper-threading technology architecture and microarchitecture. Intel Technol J 6(1)
34. openmpi: mpirun. https://www.open-mpi.org/doc/v4.1/man1/mpirun.1.php. Accessed April 2022