FORMAL MODEL OF DECOMPOSITION AND MAPPING IN ACCELERATED CLUSTER ARCHITECTURE

Miloš OČKAY, Ľubomír DEDERA

Abstract: Accelerated Cluster is currently the core architecture that drives computing performance in HPC applications. Graphic accelerators push the boundaries of the established parallel cluster architecture in a significant way, and in the near future, it will enable the achievement of exaflops milestone in HPC systems. Presented paper outlines basic elements of accelerated cluster architecture. It also explains decomposition and mapping in multistage architecture, using the data and task parallelism. Presented formal model describes the decomposition on all stages, allowing more efficient mapping and achieving an accelerated solution in a complex problem.

Keywords: Computer cluster; GPU; Accelerator; Parallel; Decomposition; Mapping.

1 INTRODUCTION

A computer cluster is a well-established architecture in High Performance Computing (HPC). The cluster consists of a number of computational nodes connected by high-speed networks, allowing hundreds of processors to be involved in the calculation. The computer cluster is not a new architecture and appeared in the 1960s. Its primary purpose was to overcome computational and memory constraints. Incorporating accelerators into a cluster architecture makes it an ideal computational tool for complex computational problems [1]. Nowadays, the most commonly used accelerators are based on GPUs and allow computing power to rise up to several PetaFLOPS [2]. The downside of this approach is more complex programming, often unique to each task. The accelerated cluster architecture is a key computational architecture, which will allow an entry into the exascale HPC era in the near future [3].

2 ACCELERATED CLUSTER ARCHITECTURE

An accelerated cluster is a specific extension of the cluster architecture. A graphics accelerator can be included in the cluster node to enrich the node's computing capabilities with a massively parallel processor. If an accelerator is included in a cluster node, the memory system, communication model, decomposition and mapping of the problem to the architecture are modified. To simplify the architecture design and decomposition process, we have divided the accelerated cluster into three stages (Fig. 1):

- cluster node stage,
- basic cluster stage,
- accelerator stage.

The cluster node stage is represented by a single CPU based computing system. The basic cluster stage consists of interconnected cluster nodes forming the compute cluster [4]. The accelerator stage consists of graphical accelerators located within the cluster nodes. The cluster may be built on a heterogeneous basis and may contain accelerated and non-accelerated nodes. A cluster node that does not include a graphics accelerator is referred to as a non-accelerated node. Accelerated nodes may contain one or more accelerators.

Compute and memory resources can be clearly identified at individual stages. It is also possible to identify the cost of data transfers.

Due to the arrangement of the computational resources, it is possible to identify the parallel and serial portions of the problem at individual stages. At the basic cluster stage, a cluster nodes form the scalable cluster. Several cluster nodes can be included in the calculation and the algorithm can be implemented in parallel. CPU is a main computational resource at the cluster node stage. Although a single CPU may contain multiple cores or a cluster node may contain several CPUs, this single element is considered serial computational resource. The cluster node can use an accelerator stage for parallel calculations. At this stage, GPU is the basic computational element and it allows to implement massive parallelism. The algorithm should respect this distribution and use the parallel computational resources to implement parallel tasks of the problem and map the serial tasks to serial computational resources. Identifying parallel and serial tasks within the problem and map them correctly to individual stages of the accelerated cluster is an important, but not the only factor for successful acceleration. The accelerated cluster has two parallel computational stages. The first consists of a group of cluster nodes at the basic cluster stage. Graphical accelerators represent the second. While the basic stage creates parallelism by clustering the nodes, the graphics adapter has a separate massively parallel processor composed of multiple stream processors. Clustering of graphics adapters is possible but limited in terms of scalability. Grouping graphics adapters at an inter-node level is very inefficient in terms of communication cost. This implies that basic stage parallelism differs from accelerator stage parallelism. While it appears to be the most efficient task parallelism at the basic cluster stage, data parallelism is more efficient at the accelerator stage.

As we will see in the process of decomposition, the use of data parallelism at the basic cluster level is not excluded, sometimes it is even necessary.

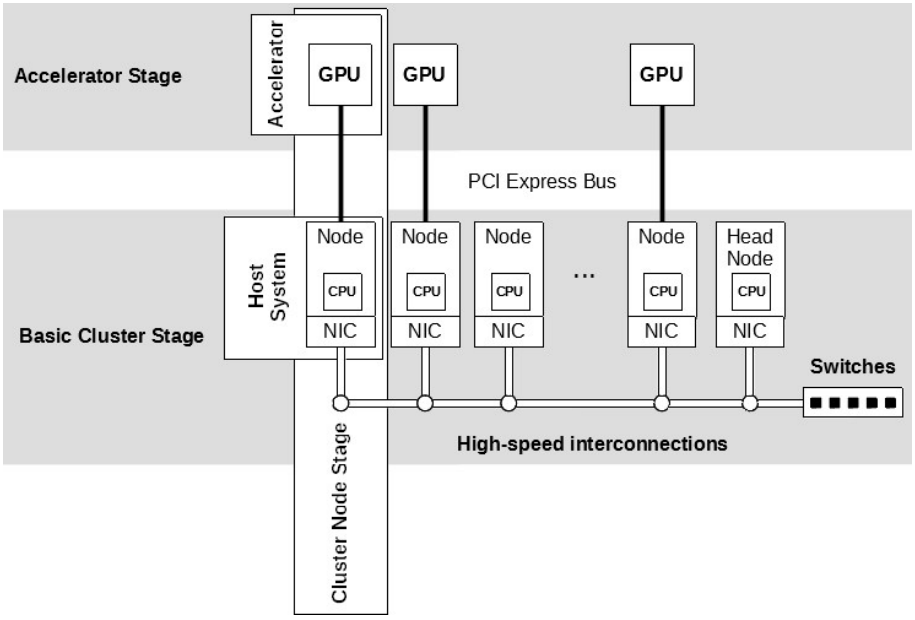

Fig. 1 Accelerated cluster architecture
Source: authors.

3 DECOMPOSITION AT THE BASIC STAGE USING TASK PARALLELISM

The P problem can be expressed as a set of its partial tasks $T_1, \ldots T_n$, ie. $P=\{T_1, T_2, \ldots, T_n\}, n \in N$ (Fig. 2).

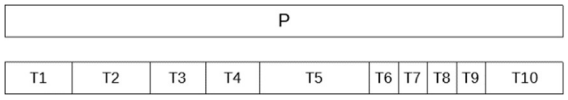

Fig. 2 Problem decomposition
Source: authors.

Dependencies of the particular partial tasks T_i can be represented by means of an oriented graph of dependencies of partial tasks (GDPT) (Fig. 3). GDPT is the set of vertices which represent partial tasks (together with a specific partial task *Result* and oriented edges which express dependencies between individual tasks. We use the *Result* as a specific task to indicate the completion of the entire calculation.

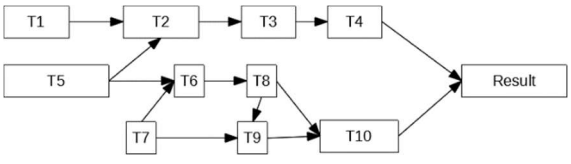

Fig. 3 An example of Graph of Dependencies of Partial Tasks
Source: authors.

Definition 1.1.: Let the $P=\{T_1, T_2, \ldots, T_n\}, n \in N$ be a problem. Then the graph of dependencies of partial tasks GDPT is called oriented graph (V, E), whose set of vertices $V=P \cup \{Result\}$ and set of edges $E \subset Px(P \cup \{Result\})$. E contains oriented edge (T_i,T_j) just when the output of the partial task T_i is connected to the input of the partial task T_j.

Independent tasks are not dependent on the output of other partial tasks. Therefore, they can be used as primary elements of task sequences. These tasks do not create downtime and allow creating as many task sequences as the number of independent partial tasks included in the problem.

Definition 1.2.: The task T is called an independent task if there is no oriented edge (V,T), $V \in P$ in the set of oriented edges.

Using GDPT, we can divide problem P into disjunctive subsets of tasks $P_1, \ldots, P_k$ so that

$$P = \bigcup_{i=1}^{k} P_i \,, \qquad (1)$$

$$P_i \cap P_j = \emptyset \text{ for } i \neq j,$$

if two partial tasks belong to the same subset of tasks P_i, then there must be (oriented) path between the vertices in the GDPT.

Since the GDPT may contain loops between tasks and the tasks of the individual P_i subsets must be arranged based on their mutual dependencies, for each subset of P_i described above, we define the sequence of tasks PT_i:

$$PT_i: N_{k_i} \rightarrow P_i, \quad N_{k_i} = \{1, \ldots, k_i\}, \qquad (2)$$

which will reflect the order of execution of individual tasks within a subset P_i. k_i represents the number of elements in a given subset of tasks, and since there may generally be a cyclic dependency between the tasks, the above representation may not be bijective.

The elements of the individual sequence of tasks are arranged on the basis of the GDPT, whose arrangement determines in which order the individual partial tasks within the individual sequence of tasks must be performed.

Definition 1.3.: The calculation plan CP will be understood as a set of task sequences.

$$CP = \{PT_i, i = 1, \dots, k\}, P_i \in P \qquad (3)$$

where each sequence of tasks contains only partial tasks pertaining to one particular problem *P*.

Definition 1.4.: Initial calculation plan is a calculation plan in which each sequence of tasks contains exactly one independent task.

Independent tasks allow us to create several task sequences that can be processed in parallel. The initial calculation plan will contain as many sequences as the number of independent tasks in the problem.

Definition 1.5.: An optimized calculation plan is a calculation plan on which the optimization steps O1 and O2 were performed.

Definition 1.6.: Let G = (V, E) be GDPT and

$$t_V: V \rightarrow < 0, \infty) \qquad (4)$$

$$t_E: E \rightarrow < 0, \infty) \qquad (5)$$

in which $t_V(Ti)$ represents the execution time of the task *Ti*, and $t_E(Ti,Tj)$ indicates the communication complexity between the output of the task *Ti* and the input of the task *Tj*. Then the oriented rated graph, which is obtained from the GDPT by evaluating its vertices using the t_V mapping and the edges using the t_E mapping, is called the rated graph of dependencies of partial tasks RGDPT (Fig. 4).

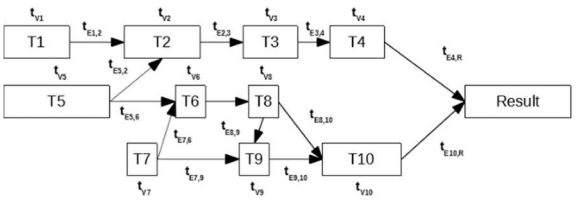

Fig. 4 An example of Rated Graph of Dependencies of Partial Tasks
Source: authors.

RGDPT unlike GDPT is an oriented graph whose vertices and edges are rated. Time values can be obtained by realizing the problem using the initial calculation plan, or in another way, e.g. testing and/or approximation.

Definition 1.7.: The time period of the calculation $<t_1, t_2>$ is called the serial period of the problem, if there is exactly one task active at each time $t \in < t_1, t_2>$.

The time period of calculation $<t_1, t_2>$ is called the parallel period of the problem if at each time $t \in <t_1,t_2>$ the number of the active tasks *k* is equal to the number of sequences of tasks.

The period of the calculation $<t_1, t_2>$ is called a partially parallel period of the problem if at each time $t \in < t_1, t_2>$, *1 < number of active tasks < k*.

Definition 1.8.: The time interval $<t_1, t_2>$ for the PT_i task sequence is called idle if the time period $<t_1, t_2>$ is a serial or partially parallel period and no task is active in the interval $<t_1, t_2>$ in the given task sequence PT_i (Fig. 5).

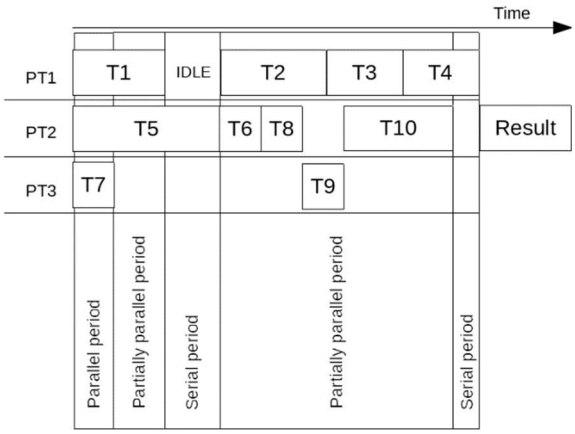

Fig. 5 An example of parallel, partially parallel and serial time periods of problem
Source: authors.

The objective of the optimization step *O1* is to maximize the use of hardware computing resources. The transfer of tasks to replace idles is possible if $t_V(T_i) \leq t_{idle}$. Such move does not increase the problem processing time, but it reduces inter-nod communication.

The objective of the optimization step *O2* is to reduce communication overhead. It is necessary to consider the possibility of moving the tasks which communicate with the tasks included in other task sequences to these sequences.

Let $T \in PT_i$ be a task included in the PT_i task sequence. Then its input communication complexity (ICC) will be defined as follows

$$ICC_{PTi}(T) = \sum_{(V,T)\in E, V\notin PT_i} t_E(V,T) \qquad (6)$$

and output communication complexity *(OCC)* will be defined with the following formula

$$OCC_{PTi}(T) = \sum_{(T,V)\in E, V\notin PT_i} t_E(T,V) \qquad (7)$$

Formulas (6) and (7) take into account the fact that communication complexity is neglected for the tasks assigned to the same sequence.

$$\max(T) = max_{V\in P}\{t_E(T,V)\} \text{ and} \qquad (8)$$

$V_{max(T)}$ is a task to which it applies
$t_E(T,V_{max(T)}) = max(T)$

Thus $V_{max}(T)$ is the task to which from T leads the edge with the maximum rating (ie. with the maximum communication complexity).

Further, $PTV_{max}(T)$ is the sequence of tasks to which $V_{max}(T)$ belongs. Then in the optimization step $O2$ we move the partial task T to the sequence of tasks $PTV_{max}(T)$ if

$$\left(OCC_{PTVmax(T)}(T) + ICC_{PTVmax(T)}(T)\right) < (OCC_{PT_i}(T) + ICC_{PT_i}(T)) \quad (9)$$

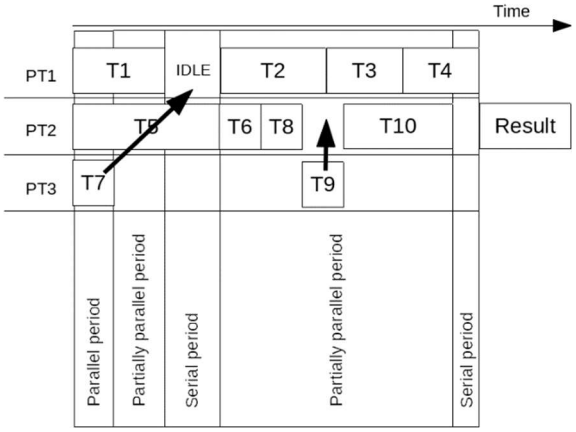

Fig. 6 An example of optimization
Source: authors.

During the $O2$ optimization step it is necessary to determine between which tasks the maximum communication complexity arises. Based on formula 9, the input and output communication complexity is compared before and after the transfer of task T to $PTV_{max}(T)$. If the formula 9 is true, the task T is moved to $PTV_{max}(T)$ and the result is the optimization of the communication costs for the task. If the formula 9 does not apply, the task remains in the original sequence of tasks (Fig. 6).

If the partial tasks of the problem are completely independent, these problems can be mapped to separate computational nodes.

4 DECOMPOSITION AT THE BASIC STAGE USING DATA PARALLELISM

Definition 1.9.: If the problem $P = \{T_1, T_2, ..., T_n\}$, $n \in N$ contains exactly one independent task T_1 and at the same time GDPT contains just edges $(T_1, T_2), (T_2, T_3), (T_3, T_4), ..., (T_{n-1}, T_n)$, then we call the P problem a serial problem.

It is not possible to effectively use task parallelism for a serial problem. If the processed data can be decomposed into smaller units (subsets) with the same data structure that do not directly depend on each other (the task uses just one subset of the processed data during the calculation), then it is possible to parallelize the serial problem using data parallelism. If data parallelism is used, several copies of the problem are created and they process different parts of the data on separate computational nodes (Fig. 7).

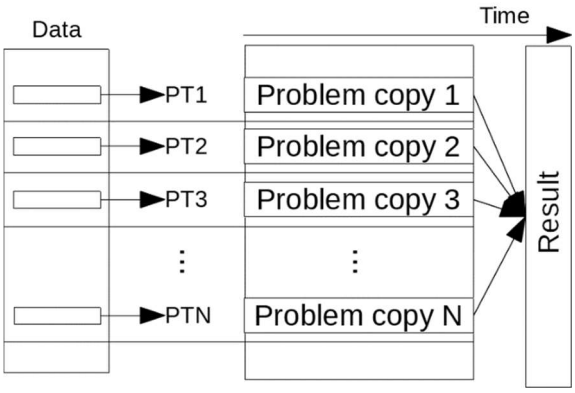

Fig. 7 Data parallelism
Source: authors.

5 DECOMPOSITION AT THE NODE STAGE

In order to use the acceleration stage, we need to express the task T as a sequence of its portions $(u_1, u_2, ..., u_k)$. Projection $ACCT_i: u \rightarrow \{S, P\}$ assigns the way the portions $(u_1, u_2, ..., u_k)$ are processed. The portion labeled S (serial) is processed by the CPU and the portions labeled P (parallel) are processed using the GPU [5] (Fig. 8).

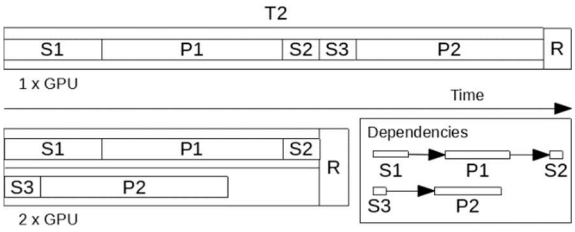

Fig. 8 Task portions S and P
Source: authors.

6 MAPPING PROBLEM TO ARCHITECTURE

The accelerated cluster AC can be expressed as a set of cluster nodes K containing GPU accelerators $AC = \{Ki, i = 1, .., N\}$.

The mapping of the problem to computational resources can be expressed at cluster node level as projection of CP to AC with the mapping of serial and parallel portions of tasks described in the previous section. Problem to architecture mappings use the following scheme:

- problem $P \rightarrow$ accelerated cluster AC,
- task sequence $PT \rightarrow$ cluster node $K_i, i \in 1...N$,
- serial portion $S \rightarrow$ CPU,
- parallel portion $P \rightarrow$ GPU.

7 CHARACTERISTICS OF THE ADAPTABLE PROBLEM GROUP

The decomposition process allows us to tailor selected problems so that they can be implemented on an accelerated cluster architecture, and the overall result is available in a shorter time in contrast with a serial system. Furthermore, decomposition makes it possible to identify problems whose processing is not effective on accelerated parallel architecture and does not speed up the overall solution of the problem.

With accelerated cluster architecture, the decomposition process is divided into several stages. At the basic cluster stage, the problem is divided into the tasks. The distribution should be coarse-grained and the tasks should not be elementary. If there is a high degree of dependency between the tasks, or if the tasks have serial characteristics, it is necessary to reconsider the structure of the processed data package. If the data package contains repetitive data or data that can be processed independently, the task can be effectively parallelized using the above methods. Problems that do not have these characteristics are not suitable for processing using accelerated cluster architecture.

At the cluster node stage, tasks are divided into serial and parallel portions. The CPU handles serial portions while parallel portions are processed by the GPU. The quality of the processing of parallel portions at the acceleration stage are highly dependent on the level of code optimization and hardware utilization [6].

Decomposition is looking for the appropriate arrangement of the individual parts of the problem and the data so that the problem can be mapped to accelerated cluster resources in order to achieve the best problem-solving performance.

From the perspective of decomposition, problems with low level of dependency among the tasks or/and low level of dependencies within the processed data will be the best candidates for the mapping to the to the accelerated cluster architecture [7].

References

[1] KOLLÁR, J.: *Metódy a prostriedky pre výkonné paralelné výpočty.* Košice : Elfa, 2003. 106 p. ISBN 80-89066-70-4.

[2] MEUER, H., DONGARRA, J., STROHMAIER, E.: *TOP500 Supercomputing sites.* TOP500, 2020, [cit. 2020]. Available at: <http://www.top500.org/>

[3] INTEL: *Driving Exascale Computing and HPC with Intel.* 2019, [cit. 2019] Available at: <https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html>

[4] BOOKMAN, CH.: *Linux Clustering: Building and Maintaining Linux Cluster.* Indianapolis : Sams Publishing, 2002. 300 p. ISBN 1-57870-274-7.

[5] SHOWERMAN, M., ENOS, J., PANT, J. A., KINDRATENKO, V., STEFFEN, C., PENNINGTON, R., HWU, W.: QP: A Heterogeneous Multi-Accelerator Cluster. In *10th LCI International Conference on High-Performance Clustered Computing,* 2009. Available at: <http://web-test.ncsa.illinois.edu/~kindr/papers/lci09_paper.pdf>

[6] RYOO, S., RODRIGUES, Ch., BAGHSORKHI, S., STONE, S., KIRK, D., HWU, W.: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proc. of 13th ACM SIGPLAN Symposium*, New York : ACM, 2008. ISBN 978-1-59593-795-7.

[7] OČKAY, M., DROPPA, M.: Embarrassingly parallel problem processed on accelerated multi-level parallel architecture. 2011, Informatics 2011 : proceedings of the eleventh international conference on Informatics : November 16 - 18, 2011 Rožňava, Slovakia. Košice : Technical University of Košice, 2011. ISBN 978-80-89284-94-8. - S. 29-32p.

Dipl. Eng. Miloš **OČKAY**, PhD.
Armed Forces Academy of General M. R. Štefánik
Department of Informatics
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: milos.ockay@aos.sk

Assoc. Prof. RNDr. Ľubomír **DEDERA**, PhD.
Armed Forces Academy of General M. R. Štefánik
Department of Informatics
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: lubomir.dedera@aos.sk

Miloš Očkay is an assistant professor at the Department of Informatics at the Armed Forces Academy in Liptovský Mikuláš. In 2003 he graduated (MSc.) at Military Academy in Liptovský Mikuláš as a civil student. He holds PhD. degree in the field of Informatics, received in 2012 from the Technical University of Košice. His scientific research is focuses on parallel computing, computer graphics and steganography.

Ľubomír Dedera works as an Associate Professor at the Department of Informatics, Armed Forces Academy in Liptovský Mikuláš. He graduated (RNDr.) from the Faculty of Mathematics and Physics, Comenius University in Bratislava in 1990. He received a PhD. degree in Artificial Intelligence from the Military Academy in Liptovský Mikuláš in 1997. His research interests include computer languages, computer security and artificial intelligence.