Scientific Programming for Heterogeneous Systems – Bridging the Gap between Algorithms and Applications

Alexey Lastovetsky
School of Computer Science and Informatics, University College Dublin
Alexey.Lastovetsky@ucd.ie

Abstract

High performance computing in heterogeneous environments is a dynamically developing area. A number of highly efficient heterogeneous parallel algorithms have been designed over last decade. At the same time, scientific software based on the algorithms is very much under par. The paper analyses main issues encountered by scientific programmers during implementation of heterogeneous parallel algorithms in a portable form. It explains how programming systems can address the issues in order to maximally facilitate implementation of parallel algorithms for heterogeneous platforms and outlines two existing programming systems for high performance heterogeneous computing, mpC and HeteroMPI.

1. Introduction

There are several reasons why heterogeneous platforms are constantly increasing their share in high performance computing. The point is that hierarchical infrastructure of supercomputer centres cannot meet increasing demand in high performance computing from research and development community. One reason is that access to such facilities is too complicated and formal. The other is that the batch mode predominantly used in the supercomputer centres cannot satisfy many interactive applications. Therefore, many researchers and engineers use local clusters for high performance computing. The local clusters are mostly and naturally heterogeneous. Moreover, collaborative inter-institutional efforts normally lead to heterogeneous high performance computing platforms in the form of two or more interconnected clusters. Faster communications make even geographically distributed clusters suitable for high performance computing. Finally, global networks and high performance Grid computing are inherently heterogeneous.

Parallel programming for heterogeneous platforms is a more difficult and challenging task than that for traditional homogeneous ones. The heterogeneity of processors means that they can execute the same code at different speeds. The heterogeneity of communication network means that different communication links may have different bandwidths and latency. As a result, traditional parallel algorithms that distribute computations and communications evenly over available processors and communication links will not be optimal for heterogeneous platforms. New, heterogeneous, algorithms should be designed to achieve top performance on heterogeneous networks of computers. Such algorithms should distribute computations and communications unevenly, taking into account the heterogeneity of the processor and the material nature and heterogeneity of the communication network.

The design of heterogeneous parallel algorithms has been quite an active research area over last decade. A number of highly efficient heterogeneous algorithms have been proposed and analysed. At the same time, there is practically no available scientific software based on these algorithms. The most important reason is that accurate implementation of the algorithms in a portable form is extremely tedious and poses a number of additional challenges that should be addressed by scientific programmers. The first one is the accuracy of the hardware model. A lot of complex extra code is needed to provide accurate values of parameters of the performance model of heterogeneous hardware. The second challenge is the portability and the ability to automatically tune to any executing platform, probably, dynamically changing their performance characteristics. Another good deal of complex code has to be written to enable the application with this feature.

The paper analyses in detail these challenges and outlines how they can be addressed by programming systems for heterogeneous parallel computing in order to maximally facilitate implementation of parallel algorithms for heterogeneous platforms. The paper also briefly explains how the proposed principles are implemented in existing programming systems for high performance heterogeneous computing, mpC and HeteroMPI.

2. Heterogeneous algorithms

An immediate implication from the heterogeneity of processors is that the processors run at different speeds. A good parallel algorithm for homogeneous distributed memory multiprocessor systems tries to evenly distribute

computations over available processors. This very distribution ensures the maximal speedup on the system consisting of identical processors. If the processors run at different speeds, faster processors will quickly perform their part of computations and begin waiting for slower ones at points of synchronization and data transfer. Therefore, the total time of computations will be determined by the time elapsed on the slowest processor. In other words, when executing parallel algorithms, which evenly distribute computations among available processors, the cluster of heterogeneous processors will demonstrate the same performance as a set of interconnected identical processors equivalent the slowest processor of the heterogeneous cluster.

Therefore, a good parallel algorithm for heterogeneous processors must distribute computations unevenly taking into account the difference in processor speed. The faster the processor is, the more computations it must perform. Ideally, the volume of computation performed by a processor should be proportional to its speed.

The problem of distribution of computations in proportion to the speed of processors is typically reduced to the problem of partitioning of some mathematical objects such as sets, matrices, graphs, etc. Optimal data partitioning over interconnected heterogeneous processors has attracted constantly growing attention of researchers in past 10 years. A number of interesting mathematical problems have been formulated and investigated (see [1] for their overview). In a generic form, a typical partitioning problem is formulated as follows:

- Given a set of processors, the speed of each of which is characterized by a positive constant,
- Partition a mathematical object into sub-objects of the same type (a set into sub-sets, a matrix into sub-matrices, a graph into sub-graphs, etc) so that
  - There is one-to-one mapping between the partitions and the processors,
  - The size of each partition (the number of elements in the sub-set or sub-matrix, the number of nodes in the sub-graph) is proportional to the speed of the processor owing the partition (that is, it is assumed that the volume of computation is proportional to the size of the processed mathematical object),
  - Some additional restrictions on the relationship between the partitions are satisfied (for example, the sub-matrices of the matrix may be required to form a two-dimensional `p×q` arrangement, where `p` and `q` may be either given constants or the parameters of the problem, the optimal value of which should be also found),
  - The partitioning minimizes some functional, which is used to measure each partitioning (for example, minimizes the sum of the perimeters of the rectangles representing the sub-matrices; intuitively, this functional measures the volume of communications for some parallel algorithms).

The investigated problems mainly deal with partitioning matrices because matrices are probably the most widely used mathematical objects in scientific computing. A general problem of optimal partitioning a square into rectangles with no restrictions on the shape and arrangement of the rectangles was studied in [2] and proved to be NP-complete. It has been also proved that the optimal column-based partitioning that minimizes the sum of the perimeters of the rectangles could be achieved in polynomial time. A version of this optimal column-based partitioning obtained under the additional restriction that the rectangles must be arranged into a given two-dimensional p×q grid was originally suggested and analyzed in [3]. The latter partitioning can be easily achieved in linear time.

The performance model of heterogeneous hardware in all these (and many other) algorithms is very simple – each processor is characterized by a positive constant representing its relative speed. A more realistic performance model that takes account of memory heterogeneity and paging effects has been recently proposed and investigated [4]. Under this model, the speed of each processor is represented by a continuous function of the size of the problem. Basic data partitioning algorithms with this functional performance model have been designed and analyzed. In particular, an algorithm of the complexity $O(p \times \log_2 n)$ solving the problem of partitioning of an n-element set over p heterogeneous processors was proposed.

Heterogeneous parallel algorithms using more advanced communication models are also being designed and investigated (compare with the above generic problem where the total volume of communication is minimized).

3. Implementation of heterogeneous algorithms: challenges and solutions

There is an obvious disproportion in the number of heterogeneous parallel algorithms, which have been designed, and scientific software based on these algorithms. The point is that implementation of a heterogeneous parallel algorithm itself is also a difficult and non-trivial task. The program implementing the algorithm has to be portable and able to automatically tune itself in order to achieve top performance in any executing heterogeneous environment. Let us take a closer look at challenges that should be addressed during the implementation of a typical heterogeneous parallel algorithm in a portable form.

A heterogeneous parallel algorithm is normally designed in a generic, parameterized form. Parameters of the algorithm can be divided into three groups. The first group includes *problem parameters*, that is, parameters of the problem to be solved (for example, the size of the matrix to be factorized). Those parameters can only be provided by the user.

The second group consists of *algorithmic parameters*, that is, parameters representing different variations and configurations of the algorithm. Examples are the size of a matrix block in local computations for linear algebra algorithms, the total number of processes executing the algorithm, their arrangement. The parameters do not change the result of computations but can have an impact on the performance. The user can be required to provide (optimal) values of these parameters, or this task can be delegated to the software implementing the algorithm.

The third group is *platform parameters*, that is, parameters of the performance model of the executing heterogeneous platform such as the speed of the processes, the bandwidth and latency of communication links between the processes. The parameters have a major impact on the performance of the program implementing the algorithm.

Consider any algorithm distributing computations in proportion to the speed of processors and based, say, on a simple constant performance model of heterogeneous processors. The algorithm should be provided with a set of positive constants representing the relative speed of the processors. The efficiency of the corresponding application will strongly depend on the accuracy of estimation of the relative speed. If this estimation is not accurate enough, the load of processors will be unbalanced, resulting in poorer execution performance.

Traditional approach to this problem is to run some test code once to measure the relative speed of the processors of the network and then use this estimation when distributing computation across the processors.

Unfortunately, the problem of accurate estimation of the relative speed of processors is not as easy as it may look. Of course, if you consider two processors, which only differ in clock rate, it is not a problem to accurately estimate their relative speed. You can use a single test code to measure their relative speed, and the relative speed will be the same for any application. This approach may also work if the processors used in computations have very similar architectural characteristics. But if you consider processors of really different architectures, the situation changes drastically. Everything in the processors may be different: set of instructions, number of instruction execution units, number of registers, structure of memory hierarchy, size of each memory level, and so on, and so on. Therefore, the processors may demonstrate different relative speeds for different applications. Moreover, processors of the same architecture but different models or configurations may also demonstrate different relative speeds on different applications. Even different applications of the same narrow class may be executed by two different processors at significantly different relative speeds.

Another complication comes up if the network of computers allows for multi-processing. In this case, the processors executing your parallel application may be also used for other computations and communications. Therefore, the real performance of the processors can dynamically change depending on the external computations and communications.

The problem of accurate estimation of platform parameters for more advanced performance models (using, say, the functional model of processors) is obviously more difficult.

Thus, a good program implementing a heterogeneous parallel algorithm should provide accurate platform parameters of the algorithm and optimal values of (some) algorithmic parameters. This means that in addition to the core code of the program, implementing the algorithm for each valid combination of the values of its parameters, the scientific programmer has to write a significant amount of non-trivial code responsible for solution of the above tasks.

How can a programming system help the scientific programmer write all the code? First of all, it does not look realistic to expect that the programming system can significantly automate writing the core code of the program. Actually, if it was the case, this would mean the possibility of automatic generation of good heterogeneous parallel algorithms from some simple specifications. As we have seen, the design of heterogeneous parallel algorithms is a very difficult and challenging task that is wide open for research. This research area is just taking first steps and requires a lot of skill and creativity from contributors. In other words, it is unrealistic to expect that parallel programming systems for heterogeneous computing can help common users having no idea about heterogeneous parallel algorithms but willing, with minimal efforts, to obtain a good parallel program efficiently solving their problems in heterogeneous environments.

At the same time, the programming system can help qualified algorithm designers write the code responsible for providing accurate platform parameters and for optimization of algorithmic parameters. The code provided by the programming system comes in two forms. The first one is the application specific code generated by a compiler from the specification of the implemented algorithm provided by the application programmer. The second type of code is not application specific and comes in the form of run-time support system and library. It is worth to note that the size and complexity of such code is

very significant and can account for more than 90% of the total code for some algorithms.

Programming systems for heterogeneous parallel computing can help not only in implementation of original heterogeneous parallel algorithms but also in efficient implementation of traditional homogeneous parallel algorithms for heterogeneous platforms. This approach to parallel programming for heterogeneous networks can be summarized as follows:

- The whole computation is partitioned into a large number of equal chunks;
- Each chunk is performed by a separate process;
- The number of processes run by each processor is proportional to the relative speed of the processor.

Thus, while distributed evenly across parallel processes, data and computations are distributed unevenly over processors of the heterogeneous network so that each processor performs the volume of computations proportional to its speed. More details on this approach can be found in [3]. Again, the code responsible for accurate estimation of platform parameters, optimization of algorithmic parameters and optimal mapping of processes to the processors can be provided by the programming system. The main responsibility of application programmer is to provide an accurate specification of the implemented algorithm. The practical value of the approach is that it can be used to port legacy parallel software to heterogeneous platforms.

4. Parallel programming systems for high performance heterogeneous computing

In this section, we briefly outline two parallel programming systems for heterogeneous computing: mpC, the first language for heterogeneous parallel programming, and HeteroMPI, an extension of MPI for high performance heterogeneous computing.

8.1. The mpC programming language

mpC is a programming language for parallel computing on heterogeneous networks [5]. It allows the application programmer to implement their heterogeneous parallel algorithms by using high level language abstractions rather than going into details of the message passing programming model of the MPI level. In addition, it takes care of the optimal mapping of the algorithm to the computers of the executing heterogeneous networks. This mapping is performed at runtime by the mpC programming system and based on two performance models:

- The performance model of the implemented algorithm,
- The performance model of the executing heterogeneous network.

The performance model of the heterogeneous network of computers is summarized as follows:

- The performance of each processor is characterized by the execution time of the same serial code
  - The serial code is provided by the application programmer.
  - It is supposed that the code is representative for the computations performed during the execution of the application.
  - The code is performed at runtime in the points of the application specified by the application programmer. Thus, the performance model of the processors provides current estimation of their speed demonstrated on the code representative for the particular application.
- The communication model is seen as a hierarchy of homogeneous communication layers. Each is characterized by the latency and bandwidth. Unlike the performance model of processors, the communication model is static. Its parameters are obtained once on the initialization of the environment and do not change since then.

The performance model of the implemented algorithm is provided by the application programmer and is a part of the mpC application. The model is specified in a generic form and includes:

- The number of processes executing the algorithm (which is normally a parameter of the model).
- The total volume of computation to be performed by each process during the execution of the algorithm.
  - The volume is specified in the form of formula including the parameters of the model.
  - The volume of computation is measured in computation units provided by the application programmer (the very code which has been used to characterize the performance of processors of the executing heterogeneous network).
- The total volume of data transferred between each pair of the processes during the execution of the algorithm.
- How exactly the processes interact during the execution of the algorithm, that is, how the processes perform the computations and communications (which computations are performed in parallel, which are serialized, which computations and communication overlap, etc.).
  - To specify the interaction of the processes, traditional serial and parallel are used such as **for**, **while**, **parallel for**, etc.

- Expressions in the statements specify the amount of computations or communications rather than the communications and computations themselves. Parameters of the algorithm and locally declared variables are widely used in that description.

The mpC compiler will translate this model specification into the code calculating the total execution time of the algorithm for every mapping of the processes of the application to the computers of the heterogeneous network. In the mpC program, the programmer can specify all parameters of the algorithm. In this case, the mpC programming system will try to find the mapping of the fully specified algorithm minimizing its estimated execution. At the same time, the programmer can leave some parameters of the algorithm unspecified (for example, the total number of processes executing the algorithm can be unspecified). In that case, the mpC programming system tries to find both the optimal value of unspecified parameters and the optimal mapping of the fully-specified algorithm.

Examples of mpC applications can be found in [5-7].

8.2. Heterogeneous MPI

Heterogeneous MPI (HeteroMPI) is an extension of MPI for programming high-performance computations on heterogeneous networks of computers (HNOCs) [8]. The standard MPI specification provides communicator and group constructors, which allow the application programmers to create a group of processes that execute together some parallel computations to solve a logical unit of a parallel algorithm. The participating processes in the group are explicitly chosen from an ordered set of processes. This approach to the group creation is quite acceptable if the MPI application runs on homogeneous distributed-memory computer systems, one process per processor. In this case, the explicitly created group will execute the parallel algorithm typically with the same execution time as any other group with the same number of processes, because the processors have the same computing power, and the latency and the bandwidth of communication links between different pairs of processors are the same. However on HNOCs, a group of processes optimally selected by taking into account the speeds of the processors, and the latencies and the bandwidths of the communication links between them, will execute the parallel algorithm faster than any other group of processes. Selection of processes in such a group is usually a very difficult task. It requires the programmers to write a lot of complex code to detect the actual speeds of the processors and the latencies of the communication links between them, and then to use this information to select the optimal set of processes running on different computers of heterogeneous network.

The main idea of HeteroMPI is to automate the process of selection of such a group of processes that executes the heterogeneous algorithm faster than any other group.

The first step in this process of automation is the specification of the performance model of the implemented heterogeneous parallel algorithm. The performance model allows an application programmer to specify his/her high-level knowledge of the application that can assist in finding the most efficient implementation on HNOCs. HeteroMPI provides a small and dedicated model definition language for specifying this performance model. The model and the model definition language are borrowed from the mpC programming language. A compiler compiles the specification of the performance model generating a set of functions that make up an algorithm-specific part of the HeteroMPI runtime system.

Having provided such a description of the performance model, the application programmer can use a new operation that tries to create a group that would execute the heterogeneous algorithm faster than any other group of processes:

```
HMPI_Group_create(HMPI_Group* gid,
     const HMPI_Model* perf_model,
     const void* model_parameters)
```

The parameter **`perf_model`** is a handle that encapsulates all the features of the performance model in the form of a set of functions generated by the compiler from the description of the performance model; **`model_parameters`** are the parameters of the performance model. This function returns a HeteroMPI handle to the group of MPI processes in **`gid.`**

During the creation of this group of processes, the HeteroMPI runtime system solves the problem of selection of the optimal set of processes running on different computers of the heterogeneous network. The solution to the problem is based on the following:

- The performance model of the parallel algorithm in the form of the set of functions generated by the compiler from the description of the performance model.
- The performance model of the executing HNOC, which reflects the state of this network just before the execution of the parallel algorithm. This model considers the executing heterogeneous network as a multilevel hierarchy of interconnected sets of heterogeneous multiprocessors. This model takes into account the material nature of communication links and their heterogeneity.

The accuracy of the model of the executing network of computers depends upon the accuracy of the estimation of the actual speeds of processors. HeteroMPI provides an operation to dynamically update the estimation of processor speeds at runtime. It is especially important if computers, executing the target program, are used for

other computations as well. In that case, the actual speeds of processors can dynamically change dependent on the external computations. The use of this operation, whose interface is shown below, allows the application programmers to write parallel programs, sensitive to such dynamic variation of the workload of the underlying computer system,

```
HMPI_Recon (HMPI_Benchmark_function func,
            const void* input_p,
            int num_of_parameters,
            void* output_p)
```

where all the processors execute the benchmark function ***func*** in parallel, and the time elapsed by each of the processors is used to refresh the estimation of its speed.

Another principal operation provided by HeteroMPI allows application programmers to predict the total time of execution of the algorithm on the underlying hardware without its real execution:

```
HMPI_Timeof(
     const HMPI_Model* perf_model,
     const void* model_parameters)
```

This function invokes the HeteroMPI runtime system, which selects the optimal set of processes based on the performance model of the parallel algorithm **perf_model**, and the performance model of the executing network of computers, which reflects the state of this network just before the execution of the parallel algorithm. The estimated execution time of the algorithm by this optimal set of processes is then returned.

One of the most important parameters, which influence the performance of the parallel application on HNOCs, is the number of processes used to execute the parallel application. Another principal operation provided by HeteroMPI frees application programmers from having to find the optimal number of processes that can execute the parallel application. They can specify only the rest of the parameters thus leaving the detection of the optimal number of processes to the HeteroMPI runtime system. Its interface is shown below,

```
HMPI_Group_auto_create (HMPI_Group* gid,
            const HMPI_Model* perf_model,
            const void* model_parameters)
```

This function returns an HeteroMPI handle to the group of MPI processes in **gid**. The parameter **perf_model** is a handle that encapsulates all the features of the performance model. These features are in the form of a set of functions generated by the compiler from the description of the performance model. The parameter **model_parameters** is an input parameter. Application programmer fills the parameter **model_parameters** with values of the input parameters to the performance model and ignores the return parameters specifying the number of processes to be involved in executing the algorithm and their relative performances.

9. Conclusion

The paper analysed in detail two main challenges encountered by scientific programmer during the implementation of heterogeneous parallel algorithms, accurate estimation of parameters of the performance model of the executing platform and automatic tuning of the implemented algorithm to each particular (possibly, dynamically changing) heterogeneous platform. It outlined how the challenges can be addressed by programming systems for heterogeneous parallel computing in order to maximally facilitate implementation of parallel algorithms for heterogeneous platforms. The paper also briefly explained how the proposed principles were implemented in existing programming systems for high performance heterogeneous computing, mpC and HeteroMPI, and how.the systems help algorithm designers implement their algorithms, minimizing their efforts in writing complex and error-prone but routine code and allowing them to focus on creative aspects of the application.

10. References

[1] J. Dongarra and A. Lastovetsky, "A Survey of Heterogeneous High Performance and Grid Computing", In *Engineering the Grid: Status and Perspective*, Eds B.DiMartino, J.Dongarra, A.Hoisie, L.Yang, and H.Zima, American Scientific Publishers, February 2006.

[2] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Mtrix Multiplication on Heterogeneous Platforms", *IEEE Transactions on Parallel and Distributed Systems* 12(10), October 2001, pp. 1033-1051.

[3] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers", *Lecture Notes in Computer Science* 1593, Springer, 1999, pp. 191-200

[4] A. Lastovetsky and R. Reddy, **"**Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers", *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS 2004), 2004.

[5] A. Lastovetsky, *Parallel Computing on Heterogeneous Networks*, John Wiley & Sons, June 2003, 423 pages.

[6] A. Kalinov, S. Klimov, et al, "Mathematical Modeling of a Supernova Explosion on a Parallel Computer", *Computational Mathematics and Mathematical Physics*, 44(5), Springer, 2004, pp. 907-914.

[7] G. Chen, P. Thulasirama, and R. Thulasiram, "Distrubuted Quasi-Monte Carlo Algorithm for Option Pricing on HNOWs Using mpC", *Proceedings of the 39th Annual Simulation Symposium*, IEEE Computer Society Press, 2006, pp. 90-97.

[8] A. Lastovetsky and R. Reddy, **"**HeteroMPI: Towards a Message-Passing Library for Heterogeneous Networks of Computers", *Journal of Parallel and Distributed Computing* 66(2), Elsevier, 2006, pp. 197-220.