J Supercomput
DOI 10.1007/s11227-014-1207-9

FuPerMod: a software tool for the optimization of data-parallel applications on heterogeneous platforms

David Clarke · Ziming Zhong · Vladimir Rychkov · Alexey Lastovetsky

Abstract Optimization of data-parallel applications for modern HPC platforms requires partitioning the computations between the heterogeneous computing devices in proportion to their speed. Heterogeneous data partitioning algorithms are based on computation performance models of the executing platforms. Their implementation is not trivial as it requires: accurate and efficient benchmarking of computing devices, which may share resources and/or execute different codes; appropriate interpolation methods to predict performance; and advanced mathematical methods to solve the data partitioning problem. In this paper, we present FuPerMod, a software tool that addresses these implementation issues and automates the development of data partitioning code in data-parallel applications for heterogeneous HPC platforms.

Keywords Heterogeneous computing · Data partitioning · Computation performance models

1 Introduction

Many scientific applications implement data-parallel algorithms, originally designed for homogeneous HPC platforms. The applications range from linear algebra routines to computer simulations, such as computational fluid dynamics. Efficient execution of

This research is supported by Science Foundation Ireland (Grant 08/IN.1/I2054).

D. Clarke · Z. Zhong · V. Rychkov (✉) · A. Lastovetsky
School of Computer Science and Informatics, University College Dublin, Dublin, Ireland
e-mail: vladimir.rychkov@ucd.ie

A. Lastovetsky
e-mail: alexey.lastovetsky@ucd.ie
URL: http://hcl.ucd.ie/project/fupermod

Published online: 20 May 2014

such data-parallel applications on a modern heterogeneous HPC platform should balance the load of its computing devices, which means that the computational workload has to be distributed in proportion to the speed of the devices. In this work, the target platform is seen as a hierarchical heterogeneous distributed-memory system. Therefore, we assume that the load balancing is achieved through data partitioning, a method widely used for distributed-memory systems. State-of-the-art algorithms designed to perform data partitioning are based on performance models of the executing platform. This work addresses implementation issues of model-based data partitioning algorithms in data-parallel applications for dedicated heterogeneous platforms.

Implementation of heterogeneous data partitioning algorithms based on computation performance models requires the construction of these models, which includes: accurate and efficient performance measurement, implementation of interpolation methods for realistic performance prediction, and formalization and solution of the data partitioning problems. Depending on the application, the requirements on the efficiency of the model-construction procedure may vary. If the application is repeatedly executed on the same stable platform, it can use the performance model constructed once in all runs. In this case, the use of exhaustive benchmarks to build a very detailed and accurate computation performance model is justified. If the application has to construct and use the model of the executing platform at runtime (adaptive application), only short measurements can be performed, whose inaccuracy has to be compensated by advanced interpolation methods. In addition, modern multicore and hardware-accelerated platforms with tightly coupled computing devices require special performance measurement techniques and computation performance models to take into account the resource contention. To the best of our knowledge, the software tool presented in this paper is the first attempt to address these challenges.

Our software tool, called FuPerMod, helps develop model-construction code for any data-parallel application. This code is guided by the accuracy and cost-effectiveness requirements of the model. It supports two types of models: constant, representing the performance of each computing device by a constant; and functional, using functions for that representation. For data partitioning itself, FuPerMod provides careful implementation of a wide range of data partitioning algorithms based on both constant and functional computation performance models. The software tool supports a wide range of heterogeneous platforms consisting of uniprocessors, multicores, and hardware accelerators. It is extensible: new measurement techniques for new types of hardware as well as new computation performance models and data partitioning algorithms can be easily added.

The paper is organized as follows. In Sect. 2, we overview existing data partitioning software. In Sect. 3, we discuss the main challenges in model-based optimization of data-parallel applications for heterogeneous platforms. In Sect. 4, we present our proposed software tool and demonstrate its use.

2 Existing data partitioning software

Most existing data partitioning software implements partitioning algorithms for graphs, which are then applied to sparse matrices and meshes, the mathematical

objects widely used in scientific applications. Algorithms implemented in ParMetis [7], SCOTCH [4], JOSTLE [12] reduce the number of edges between the target subdomains, aiming to minimize the total communication cost of the parallel application. They take into account the platform heterogeneity, which is specified by a weighted graph providing information about the speed of processors and the bandwidth of links. Algorithms implemented in Zoltan [3], PaGrid [1] minimize the execution time of the application using a cost function, which also depends on the weighted graph of the platform. To distribute computations between the processors, all these graph partitioning libraries use simplistic computation performance models, where the speed of a processor is given by a constant (weight). Despite the fact that accurate data partitioning is dependent on the weights, these libraries provide no method to find the values that would balance the load for a given application on a given heterogeneous platform. Application programmers are responsible for building the computation performance models and distributing the load.

Traditionally, the constants characterizing the performance of the processors are found as their relative speeds demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size. This approach is not always accurate and may result in non-optimal partitioning on modern highly heterogeneous platforms, as demonstrated in [6]. Existing data partitioning software, which is based on this approach, does not take into account memory hierarchy, hierarchy of computing devices, software heterogeneity, optimizations, and out-of-core techniques used in software. For modern heterogeneous platforms, more realistic computation performance models have been proposed [8] along with more elaborate general-purpose model-based data partitioning algorithms to find the optimal load distribution ratios, which can be used as weights in graph partitioning. However, integration of these algorithms into data-parallel applications is not trivial. In the following section, we discuss the main challenges of implementation of model-based heterogeneous data-parallel applications and identify the features of a software tool that would address these challenges.

3 Optimization of parallel applications for heterogeneous platforms

In this section, we analyze the main challenges the application programmers face while optimizing data-parallel applications for modern heterogeneous HPC platforms. Given a data-parallel scientific application, originally designed for distributed-memory systems and implemented with help of MPI, how to execute it efficiently on a heterogeneous platform? We assume that the total volume of communication is minimized at the application level (for example, by multilevel graph partitioning in mesh applications [7], or by arrangement of matrix blocks in matrix applications [2]). We also assume that in the computationally intensive part, the application calls a library of routines, for which the hardware-optimized implementations are available (for example, multi-threaded and GPU solvers). Then, to execute this application efficiently on the heterogeneous platform, we need to distribute the application data unevenly between its heterogeneous computing devices, based on the a priori information about their performance.

A general-purpose data partitioning algorithm based on computation performance models proceeds as follows. As input, it requires a performance model, which can be constructed either in advance or at run-time. The model must accurately approximate the speed of the application on each of the computing devices. Therefore, its construction requires reliable empirical information about the real performance, which can be obtained from the carefully designed benchmarks that can assess the performance of the whole application on the devices. Thus, the application programmer has to develop methods and code solving the following non-trivial tasks: (i) accurate and efficient performance measurement, (ii) construction of computation performance models, and (iii) implementation of model-based data partitioning algorithms.

Accurate and cost-effective methods of performance measurement are paramount for data partitioning to work in real-life heterogeneous environments. The use of wrong estimates can fully destroy the resulting performance of the application. Performance can be found by benchmarking *a computation kernel*, a serial code performing much less computations but still representative for the entire application [8]. For example, computationally intensive applications often perform the same core computation multiple times in a loop. The benchmark made of one such core computation can be representative of the performance of the whole application and can be used as a kernel. Timing the computation kernel on heterogeneous devices may be non-trivial, especially on platforms with tightly coupled devices. For example, on multicore platforms, parallel processes interfere with each other through shared memory so that the speed of individual cores cannot be measured independently. In this case, the performance of cores should be measured in a group, when all cores are executing the benchmarks in parallel [13]. Interactions between CPUs and GPUs include data transfers between the host memory and the GPU memory over PCI Express, launching of GPU kernels, and some other operations. Performance measurement techniques for heterogeneous GPU-accelerated systems were studied in [10]. It was concluded that the synchronous approach, when the host CPU core observes the beginning and the end of an operation, is valid for the measurement of routines implemented in synchronous libraries, such as CUBLAS. This technique covers all interactions between devices and does not require any special measurement mechanisms. The performance of out-of-core routines can also be measured from the host CPU. Incorporation of these and other state-of-the-art **performance measurement techniques** into the data-parallel application currently represents a significant development effort, but could be and should be supported by an appropriate software tool.

The results of the performance measurements are used to construct computation performance models, which will be then used in heterogeneous data partitioning algorithms. The choice of the model and the algorithm depends on the application and the platform. If the application data always fits into the memory of the devices, which do not switch between different codes during the execution, then their speed does not vary much with problem size. In this case, highly efficient data partitioning algorithms based on the **constant performance model** (CPM) can be used. For the general case, when the absolute speed depends on the problem size, we proposed the **functional performance model** (FPM) [8]. In this model, the speed is represented by a continuous function of problem size, which is built empirically and integrates performance

characteristics of both the architecture and the application. The speed is defined as the number of computation units processed per second. The computation unit can be defined differently for different applications. The important requirement is that it does not vary during the execution of the application. This model can be estimated in the same way for any data-parallel application. It approximates the execution time and speed using piecewise linear or Akima spline interpolation [11]. Originally, the functional performance model was designed for uniprocessor machines: it provided optimal data partitioning [5] and efficient dynamic load balancing [6] on heterogeneous networks of uniprocessors. Later, this approach was extended to multicore and hybrid CPU/GPU [13] platforms. To address the resource contention in these platforms, we proposed to partition the set of devices into relatively independent groups, such as multiple cores on the same socket or a GPU and its host core, construct the performance model of each such a group independently, and use these models in data partitioning algorithms. Implementation of this approach would be very challenging without an appropriate supportive software tool.

Elaborate computation performance models provide more accurate prediction but complicate data partitioning algorithms. In contrast to the traditional algorithms, which only need to distribute computations in proportion to given constants, the algorithms based on functional models have to solve much more complex partitioning problems to yield the balance. Given the speeds are interpolated from empirical data, like in [8], the solution can be found using different geometrical [8] or numerical [11] algorithms, depending on the shape of the speed functions. **Complexity of these modern data partitioning algorithms** makes their quality implementation a challenging task, and a software tool solving this task would significantly facilitate the development of heterogeneous data-parallel applications.

4 New software tool for model-based data partitioning

In this paper, we present the new software tool that addresses the above challenges. We illustrate how to adapt data-parallel MPI applications to hybrid heterogeneous platforms using this tool.

Our software tool, FuPerMod, provides the programming interface for: (i) accurate and cost-effective performance measurement; (ii) construction of computation performance models that implement different methods of interpolation of time and speed; (iii) invocation of model-based data partitioning algorithms for static and dynamic load balancing. This functionality can be incorporated into a data-parallel applications as follows. First, the application programmer has to provide the computation kernel of their application and define its computation unit using the API provided. This kernel will be used for computation performance measurements, which can be carried out either within the application or separately, to obtain the a priori performance information. Then, the programmer chooses the appropriate computation performance model and data partitioning algorithm, and incorporates them into the application. Upon execution of the data-parallel application on the heterogeneous platform, the models of processors/devices/groups of devices will be constructed and the data partitioning algorithm will yield the optimal distribution of workload for a given problem

size. Finally, the programmer should distribute the application data according to this optimal distribution.

Measuring computation performance with FuPerMod. Performance measurement is automated by FuPerMod as follows. In addition to the computation kernel, the application programmer has to provide the functions to allocate and deallocate the data. In these functions, the application programmer defines the computation unit and reproduces the memory requirements of the application. To enable conversion of speed from units/sec to FLOPS, the programmer has the option to specify the complexity of the computation unit. For illustration, we use parallel multiplication of dense matrices [5].

Performance measurement of computation kernels on heterogeneous devices that share resources and use different programming models is challenging. In [13], we propose the measurement technique for a multicore GPU-accelerated node, which is also implemented in FuPerMod. It provides reproducible results within a given accuracy and can be summarized as follows. As automatic rearrangement of the processes by the operating system may result in performance variations, we bind the processes to cores to ensure its stability. We also synchronize the processes that share resources (on a node or a socket), to reproduce the worst-case scenario in terms of resource contention during the execution of the application. To ensure the reliability of the measurement, the experiments are repeated multiple times until the results are statistically representative. In GPU-accelerated nodes, we measure the combined performance of each GPU and its host core, including the overhead incurred by data transfer between them. Due to limited GPU memory, the execution time of GPU kernels can be measured only within some range of problem sizes, unless out-of-core implementations, which address this limitation, are available.

Supported performance models and data partitioning algorithms. Currently, FuPerMod implements the following performance models:

- CPM (requires only one experimental point);
- FPM based on the piecewise linear interpolation of the speed;
- FPM based on the Akima spline interpolation of the speed.

The piecewise linear FPM is constructed so that it will satisfy some rather restrictive assumptions on the shape of the speed function [8], excluding outliers from the experimental data as shown in Fig. 1a. The FPM based on the Akima spline interpolation relaxes these restrictions [11], and therefore, represents the speed of the processor with more accurate functions (Fig. 1b). FuPerMod can be extended by adding other computation performance models.

The computation performance models are used as input for model-based data partitioning algorithms. FuPerMod provides the following algorithms:

- straightforward CPM-based algorithm;
- geometrical algorithm based on the piecewise linear FPMs;
- numerical algorithm based on the Akima-spline FPMs.

The CPM-based algorithm divides the data in proportion to the constant speeds. This is the fastest but least accurate data partitioning algorithm. It is appropriate for the cases when the speed does not vary significantly with the problems size. The geometrical algorithm implements iterative bisection of the speed functions with lines

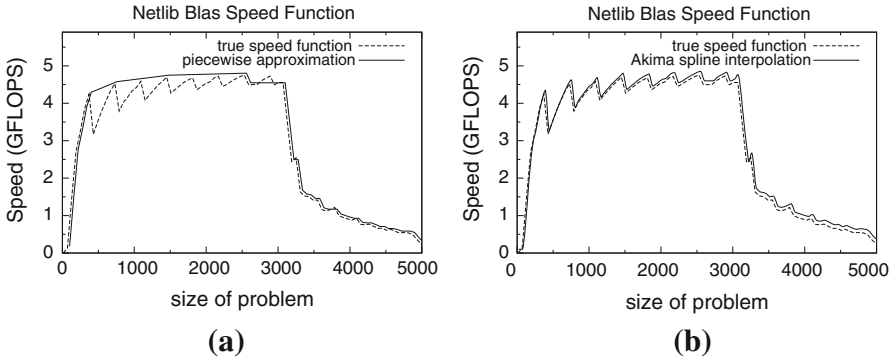

Fig. 1 Speed functions of the matrix multiplication kernel based on the Netlib BLAS GEMM: **a** piecewise linear interpolation, **b** Akima spline interpolation

passing through the origin of the coordinate system [8]. Quick convergence of this algorithm to the global optimum is ensured by putting restrictions on the shape of the speed functions, which is implemented in the piecewise linear FPMs. The numerical algorithm applies multidimensional solvers to numerical solution of the system of non-linear equations that formalize the problem of optimal data partitioning [11]. It can be applied to smooth speed functions of any shape. As input, the algorithm takes the Akima-spline FPMs, since this approximation provides continuous derivative.

The construction cost of a full computation performance model, i.e., a functional model for the full range of problem sizes, may be very high, which limits the applicability of the above partitioning algorithms to situations where the construction of the models and their use in the application can be separated. For example, if we develop an application that will be executed on the same platform multiple times, we can build the full model once and then use this model multiple times during the repeated execution of the application. In this case, the time of construction of the model can become very small compared to the accumulated performance gains during the multiple executions of the optimized application. Building the full functional performance model is not suitable for applications, each execution of which is seen as unique. In this case, computations should be optimally distributed between processors without *a priori* information about their performance characteristics.

Dynamic data partitioning and load balancing with FuPerMod. FuPerMod provides efficient data partitioning algorithms that do not require performance models as input. These algorithms do not construct complete performance models, but partially estimate them with the accuracy sufficient for near-optimal distribution of computations. Due to a much smaller number of experimental points, they have a low execution cost that makes them suitable for the use in self-adaptable applications. FuPerMod provides two such algorithms: for dynamic data partitioning [9] and dynamic load balancing [6].

The dynamic algorithms are searching for a near-optimal partitioning iteratively. At each iteration, they invoke the FPM-based data partitioning algorithm, using as input not the accurate full computation performance models but their partial estimates

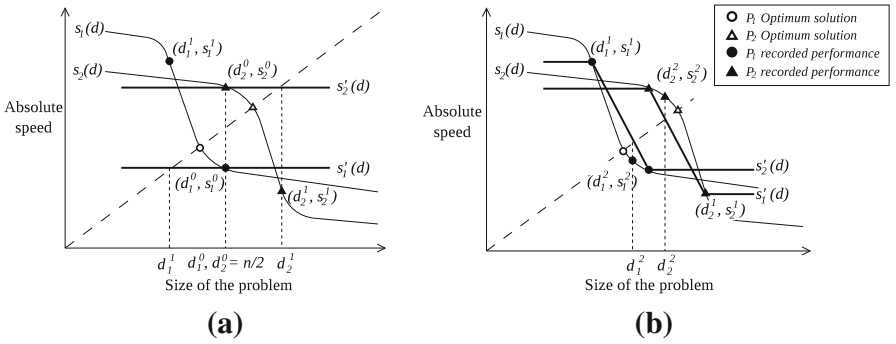

Fig. 2 Construction of the partial FPMs based on piecewise linear interpolation, using the geometrical data partitioning algorithm, for two processors P_1 and P_2 solving the problem of size n. The distribution found at the second iteration is already quite close to the optimum

constructed from the experimental points obtained at the previous iterations. The new partitioning found at this step is then used in the next experiment, and the execution time on each processor is measured. If these times are sufficiently close, then the partitioning is returned as the final solution. If not, the new experimental points are used to refine the partial estimates of the speed functions, and the search proceeds to the next iteration. Figure 2 shows a few steps of dynamic data partitioning, with the piecewise linear FPMs and the geometrical data partitioning used at each iteration.

In conclusion, we demonstrate how to use dynamic algorithms for optimization of a parallel application implementing the Jacobi method. This application iteratively solves a system of linear equations, $A\mathbf{x} = \mathbf{b}$. A is a square matrix of size $N \times N$, which is distributed by rows between p processors and decomposed into the diagonal and remainder, $A = D + R$. The partial piecewise linear FPMs are constructed at runtime. At each iteration, the load-balancing function invokes the geometrical data partitioning algorithm. The system of equations is redistributed accordingly to the newly obtained data distribution. The pseudocode of the optimized Jacobi method is shown in Algorithm 1. Figure 3 shows that after few iterations of the application, the load of the processors gets balanced.

Algorithm 1 FPM-based dynamic load balancing of the Jacobi method

```
Initial partition of rows: d_i ← N/p
Allocate the slice of matrix A_i and vectors b, x^k, x^{k+1}
FuPerMod: Allocate an empty piecewise linear model s_i
while |x^{k+1} − x^k| > ε do
    Redistribute the matrix rows A_i according to d_i
    FuPerMod: Start timer t_i
    Jacobi iteration: x^{k+1} ← D^{-1}(b − Rx^k)
    FuPerMod: End timer; update the model s_i ← s_i + (d_i, t_i)
    FuPerMod: d_i ← geometrical data partitioning based on s_i
end while
```

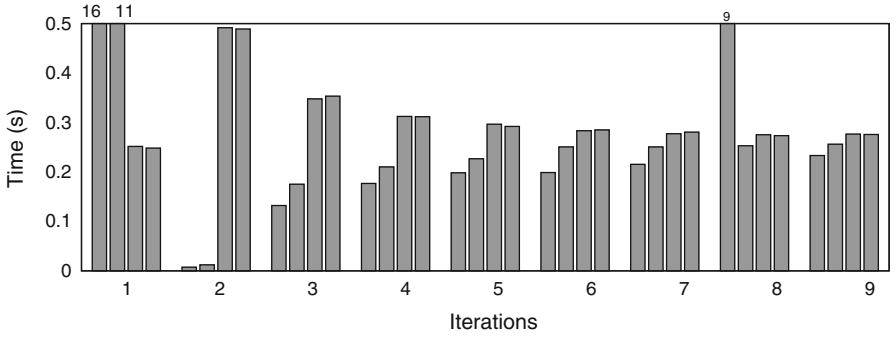

Fig. 3 Dynamic load balancing of the Jacobi method with geometrical data partitioning on four heterogeneous processors

References

1. Aubanel E, Wu X (2007) Incorporating latency in heterogeneous graph partitioning. In: IPDPS 2007, pp 1–8
2. Beaumont O, Boudet V, Rastello F, Robert Y (2001) Matrix multiplication on heterogeneous platforms. IEEE Trans Parallel Distrib Syst 12(10):1033–1051
3. Catalyurek U, Boman E, Devine K et al (2007) Hypergraph-based dynamic load balancing for adaptive scientific computations. In: IPDPS 2007, pp 1–11
4. Chevalier C, Pellegrini F (2008) PT-Scotch: a tool for efficient parallel graph ordering. Parallel Comput 34(68):318–331
5. Clarke D, Lastovetsky A, Rychkov V (2012) Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models. In: HeteroPar'2011, pp 450–459
6. Clarke D et al (2011) Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms. Parallel Process Lett 21:195–217
7. Karypis G, Schloegel K (2013) ParMETIS: parallel graph partitioning and sparse matrix ordering library. Version 4
8. Lastovetsky A, Reddy R (2007) Data partitioning with a functional performance model of heterogeneous processors. Int J High Perform C 21:76–90
9. Lastovetsky A, Reddy R (2010) Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models. In: Euro-Par 2009, LNCS, vol 6043. Springer, pp 91–101
10. Malony AD, Biersdorff S, Shende S et al (2011) Parallel performance measurement of heterogeneous parallel systems with GPUs. In: ICPP '11, pp 176–185
11. Rychkov V, Clarke D, Lastovetsky A (2011) Using multidimensional solvers for optimal data partitioning on dedicated heterogeneous HPC platforms. In: PaCT-2011, LNCS, vol 6873. Springer, pp 332–346
12. Walshaw C, Cross M (2001) Multilevel mesh partitioning for heterogeneous communication networks. Future Gener Comput Syst 17(5):601–623
13. Zhong Z, Rychkov V, Lastovetsky A (2012) Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications. In: Cluster, pp 191–199