2011 IEEE International Conference on Cluster Computing

Data Partitioning on Heterogeneous Multicore Platforms

Ziming Zhong, Vladimir Rychkov, Alexey Lastovetsky
School of Computer Science and Informatics
University College Dublin
Dublin, Ireland
ziming.zhong@ucdconnect.ie, {vladimir.rychkov, alexey.lastovetsky}@ucd.ie

***Abstract*—In this paper, we present two techniques for inter- and intra-node data partitioning aimed at load balancing MPI applications on heterogeneous multicore platforms. For load balancing between the multicore nodes of a heterogeneous multicore cluster, we propose how to define a functional performance model of an individual multicore node as a single computing unit, and use these models for data partitioning between the nodes. For load balancing within a heterogeneous multicore node, we propose a data partitioning technique between cores. Since parallel processes interfere with each other through shared memory, the speed of individual cores cannot be measured independently, and independent performance models cannot be defined for cores. Therefore, for a given problem size, we dynamically evaluate the performance of cores, while they are executing only the computational kernel of parallel application, and partition data proportionally to the observed speed.**

Keywords-heterogeneous multicore cluster; load balancing; data partitioning; functional performance models

I. INTRODUCTION

High performance of parallel applications on dedicated heterogeneous platforms can be achieved by partitioning the computational load and, hence, data unevenly across all processors. For this purpose, performance models of the heterogeneous processors are used. It has been proven that the application-specific functional performance models built from a history of time measurements better capture different aspects of heterogeneity of processors [1]. They are designed for uniprocessor machines and represent the processor speed by a function of problem size. The models and partitioning techniques are however designed for uniprocessor machines. In this paper, we apply this approach to a computing node that consists of multiple cores and executes multiple processes/threads of the application, considering the node a single processing unit. We propose a multicore functional performance model that integrates contributions of arithmetic calculations and memory traffic between cores, cache and DRAM. We use this model with the previously proposed data partitioning algorithm [1] to balance parallel computational routines on a heterogeneous cluster of multicore nodes.

A number of models have been proposed to evaluate the performance of a multicore [2]-[5]. They however have never been used for optimal distribution of computations between nodes. Their typical use is for code optimization. Most of them are hardware-centric and represent the speed of the computing node by peak arithmetic and memory performance. Parameters of the models are obtained from experiments with elemental operations. They provide the insight into how to modify the code for more efficient utilization of the multicore node. In this paper, we address a different problem, which cannot be solved with help of the existing multicore performance models. Namely, we try to improve the existing parallel application by finding its optimal configuration for execution on a heterogeneous multicore node.

Scientific parallel programs for homogeneous and heterogeneous clusters are mainly written using MPI or its extensions like HeteroMPI [6]. However MPI could be also used as a programming tool for scientific computing on a single multicore node. This would allow for reusing existing MPI code. MPI implementations often outperform their multithreaded and OpenMP counterparts for many scientific kernels [7]. The fact that MPI processes can be bound to particular cores allows us to find the optimal distribution of the workload between MPI processes and their optimal mapping to the executing cores. This optimization technique can be particularly beneficial for heterogeneous cores.

Optimal distribution of computations between heterogeneous processors is typically based on their individual performance models, which can be either pre-built or being built during the execution of the application for each processor. In the case of a multicore node, the performance model of an individual core cannot be built independently of other cores because they compete for shared resources and hence affect the performance of each other. Therefore, we cannot straightforwardly apply the traditional approach to cores, because this approach assumes the independence of the processing units. In this paper, for load balancing of MPI parallel computational routines on a heterogeneous multicore node, we still propose to use evaluation of the individual performance of cores but in a group, when all cores are executing some allocated workloads in parallel. This evaluation is performed at runtime and obtained individual performance models are then used for optimal distribution of computations.

In order to implement the proposed load balancing techniques, we propose a new method of benchmarking of scientific applications on multicores. Existing methods measure the performance of a platform, using different combinations of memory operations [8] or floating point

978-0-7695-4516-5/11 $26.00 © 2011 IEEE
DOI 10.1109/CLUSTER.2011.64

kernels [9]. We propose to measure a combined workload of a real parallel computational application, including both arithmetic and memory operations. We bind the processes to the cores and synchronize their execution in order to ensure consistent performance. To ensure the reliability of the measurement, we repeat experiments multiple times until the results are proved statistically reliable.

This paper is structured as follows. In Section 2, we overview existing performance models that are either specifically designed for or can be adapted to multicores. In Section 3, we propose the multicore functional performance model, describe how to built the model, and present the experimental results of its use for data partitioning on heterogeneous multicore machines of Grid5000. In Section 4, we present the intra-node load balancing technique based on dynamic performance evaluation of the kernel of an MPI application, and demonstrate that this technique can improve the performance of a heterogeneous multicore.

II. Related Work

In this section, we overview existing performance models and benchmarks designed for multicores. We also outline one existing approach to data partitioning with help of performance models that can be applied to multicores.

The main idea of existing performance models of a multicore node is to estimate the performance of parallel applications in order to reengineer them and adapt to emerging multicore-based supercomputers. Many models are not very accurate, providing only performance bounds, as, for example, an extension of the Amdahl model for multicores [5], which does not take into account memory overhead. Other models, in contrast, mostly represent memory traffic, for example [2]-[3]. The traffic models are empirically derived from the traces of full-system simulations with different applications. Using statistical analysis, they realistically capture the variety of traffic patterns arising from a wide range of applications.

A number of studies introduce a concept of balance, defined as a ratio of the number of memory operations to the number of floating-point operations [10]. This ratio is a more realistic performance model showing how fast the data is supplied and processed in the application. If the loop balance is higher than the machine one, the application needs data at a higher rate than the machine can provide and idle computational cycles exist. Otherwise, data cannot be processed as fast as it is supplied to the processor, and idle memory cycles will exist. In the first case, the application is memory bound and should be optimized if possible. In the second case, the application runs at the peak floating-point rate of the machine and does not require further optimization. Similarly, the Roofline model [4] ties together floating-point and memory performance in a two dimensional graph. All bounds provided by the Roofline model represent different optimization techniques for multicore applications. In contrast to the traditional approaches, we aim to improve the performance of existing parallel applications by finding their optimal configuration using data partitioning algorithms.

Since most scientific applications are memory-intensive, sharing memory and insufficient memory bandwidth are the main factors affecting their performance. Due to the memory-boundness of such scientific applications, the synthetic benchmarks that measure the sustainable memory bandwidth became very popular for multicores [8]. Another group of the benchmarks is floating-point kernels, representative numerical methods important for computational science and engineering [9]. These benchmarks evaluate the peak performance of the platform, using different combinations of the following characteristics: parallelization, data locality, communication-to-computation ratio, off-chip traffic. The benchmark method proposed in this work belongs to the second group and measures a combined workload, including both arithmetic and memory operations. We bind the processes to the cores and synchronize their execution in order to ensure consistent performance. To ensure the reliability of the measurement, we repeat experiments multiple times until the results are proved statistically reliable.

Conventional algorithms for distribution of computations between heterogeneous processors partition data between the processors in proportion to their speeds demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size. In the case of cluster of uniprocessors, if we do not limit the range of problem sizes, their absolute speeds of the processors will depend on the size of the computational task. For these platforms, a new class of algorithms [1] has been proposed recently, able to optimally distribute computations between processing units for the full range of problem sizes.

The functional performance model (FPM) of heterogeneous processors proposed and analysed in [1] has proven to be more realistic because it integrates many important features of heterogeneous processors such as the architectural and platform heterogeneity, the heterogeneity of memory structure, the effects of paging and so on. The algorithms employing it therefore distribute the computations across the heterogeneous processing units more accurately than the algorithms employing the constant performance models. Under this model, the speed of each processor is represented by a continuous function of the size of the problem. In this paper, we adapt the functional performance model to a multicore node, considering the latter a single processing unit. We use this adapted model in combination with the proposed earlier FPM-based data partitioning algorithms in order to balance the load between the nodes of a heterogeneous multicore cluster.

III. Inter-Node Data Partitioning

In this section, we present the load balancing technique based on data partitioning between the nodes of a heterogeneous multicore cluster.

In the proposed multicore functional performance model (MFPM), a node consisting of cores is considered as a single processing unit. The node executes processes of a data parallel computational routine. The speed is represented by a positive continuous function, $s(x) = \frac{x}{t(x)}$, where $x = \sum_{i=0}^{c-1} x_i$ is the total problem size processed on the node (the problem

sizes assigned to individual cores, x_i, $0 \le i < c$, are supposed to be given); $t(x) = \max_{i=0}^{c-1} t_i(x_i)$ is the execution time of the parallel routine and $t_i(x_i)$ is the execution time of the process on i-th core. This speed function includes contribution from both cores and memory. It takes into account the potential heterogeneity of cores and multi-level storage system within the multicore computer.

According to the Roofline model [4], peak floating-point performance, bandwidth and operational intensity are the three main factors that influence the computing performance. Since we do not optimize the code to improve the floating-point performance or the utilization of the bandwidth, the change of operational intensity with the increase of the problem size determines the computing performance. The performance decreases with the increase of problem size until it becomes so large that the memory traffic dominates computation. Then the performance becomes almost stable. Strictly speaking, the proposed speed function depends not only on the problem size, x, but also on its distribution between the cores, x_i, $0 \le i < c$. Uneven data distribution will result in idle computational cycles on some cores where the processes complete their job earlier. However, on a heterogeneous multicore, the maximum speed can be achieved with some uneven data distribution. In Section 4 of this paper, we present the approach to optimal intra-node data partitioning for a heterogeneous multicore.

To build multicore functional performance models of an MPI application on a heterogeneous multicore cluster, we design and implement the following benchmarking procedure that allows us to measure the execution time accurately on each multicore. We assume that all nodes have a homogeneous architecture but different features (different number of cores/sockets, different sizes of caches/memory, etc). This allows us to partition data evenly within each node. Since automatic rearranging of the processes provided by operating system may result in performance degradation, we bind the processes to cores. The procedure can be summarized as follows:

1. The communicator of all processes is spilt into intra-node MPI communicators so that there is one such communicator per a multicore node. The data of the size x is partitioned evenly between the cores within each node.

In the loop:

2. A barrier synchronizes the processes within each node. This way we minimize the idle computational cycles on the cores, aiming at the highest floating-point rate for the application on the node. We also ensure that the resources will be shared between the maximum number of processes, generating the highest memory traffic.
3. The execution time of the routine is measured on each core.
4. Statistical analysis of all time measurements observed so far for the given problem size is performed.
5. If all intra-node processes get statistically reliable results, the total speed of the node is calculated, otherwise, more repetitions are required (goto 2).

We use the multicore functional performance models built this way for FPM-based data partitioning on a heterogeneous multicore cluster. In the experiments, we use MFPMs and the heterogeneous two-dimensional matrix-matrix multiplication routine that employs the two-dimensional column-based matrix partitioning algorithm presented in [11]. Our modification uses the MFPMs instead of the constant performance models (CPMs) to calculate the relative speeds of the heterogeneous multicore nodes as the input parameters for the 2D column-based matrix partitioning algorithm.

Fig. 1(a) shows one step of the algorithm of parallel matrix multiplication over a 2D heterogeneous processor grid of size 3×3. Each element of the matrix is a square block of size $b \times b$. As shown in Fig. 1(b), taking processer P_{12} as an example, the computational kernel for 2D matrix multiplication performs a matrix update of a matrix C_b of size $m_b \times n_b$ using A_b of size $m_b \times 1$ and B_b of size $1 \times n_b$. The size of the problem is represented by two parameters, m_b and n_b. We use a combined computation unit, which is made up of one addition and one multiplication, to express the volume of computation. Therefore, the total number of computation units (namely, multiplications of two $b \times b$ matrices) performed during the execution of the benchmark code will be approximately equal to $m_b \times n_b$, and the absolute speed of the processor exposed by the application can be calculated as $m_b \times n_b$ divided by the execution time of the matrix update.

The MFPM-based modification of the heterogeneous 2D column-based matrix partitioning can be summarized as follows.

(a)

(b)

Figure 1. (a) One step of the algorithm of parallel matrix multiplication employing a 2D heterogeneous processor grid of size 3×3. Matrices A, B, and C are partitioned so that the area of the rectangle is proportional to the speed of the processor owning it. First, each $b \times b$ block of the pivot column $a_{.k}$ of matrix A (shown with curly arrows) is broadcast horizontally, and each $b \times b$ block of the pivot row $b_{k.}$ of matrix B (shown with curly arrows) is broadcast vertically. Then, each $b \times b$ block c_{ij} of matrix C is updated, $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$. (b) The computational kernel (shown here for processor P_{12} for example) performs a matrix update of a dense matrix C_b of size $m_b \times n_b$ using A_b of size $m_b \times 1$ and B_b of size $1 \times n_b$. The matrix elements represent $b \times b$ matrix blocks.

1. The computational kernel updating a square matrix of a given area (m_b=n_b in Fig. 1(b)) is used to pre-build MFPMs of the nodes, assuming each node to be a homogeneous multicore.
2. The area of the partition (measured in the number of matrix blocks) to be allocated to each node is calculated by the FPM-based data partitioning algorithm [1], which uses the MFPMs. The resulting areas will be proportional to the speeds of the nodes.
3. The algorithm [11] then uses these areas to calculate the optimal column-based matrix partitioning which minimizes the total volume of communication between the nodes (see Fig. 2(a)).
4. The matrix blocks allocated to each node are evenly partitioned between cores in horizontal slices (see Fig. 2(b)).

We perform our experiments on a heterogeneous platform, Grid5000 (http://www.grid5000.fr). Four different types of multicore nodes are involved in the experiments, with different numbers of cores and different sizes of RAM. We use 4 nodes of each type to construct a 16-node heterogeneous cluster to run the experimental applications. We compare the execution time of the above matrix multiplication application with three different data partitioning schemes. The first scheme uses the pre-built MFPMs of the nodes to find the optimal 2D column-based data distribution of matrix. The second one uses CPMs instead of MFPMs to find the optimal 2D column-based data distribution and the third one partition the matrix evenly (homogeneous data partition) to all nodes without considering their heterogeneity.

Fig. 3 shows the multicore functional performance models of the four types of nodes when they execute the computational kernel of the 2D matrix multiplication. The x-axis represents the total number of matrix blocks, and the y-axis represents the speed of the nodes. These four types of nodes represent a common heterogeneity of high performance computing platform in the real world.

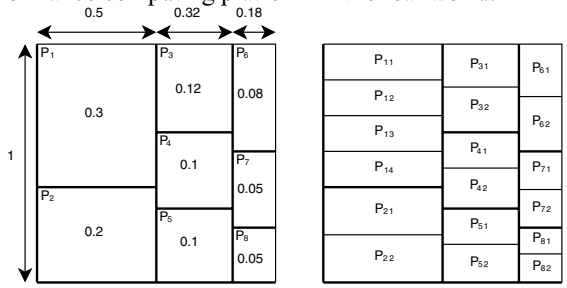

Figure 2. Example of the 2D column-based matrix partitioning over 8 multicore nodes. The relative speed of nodes is given by {0.3, 0.2, 0.12, 0.1, 0.1, 0.08, 0.05, 0.05} and the number of cores within these nodes are {4, 2, 2, 2, 2, 2, 2, 2}. (a) The optimal partitioning is obtained for three columns. The first column of width 0.5 is composed of 2 elements (P1 and P2). The second column of width 0.32 is composed of three elements (P3, P4 and P5). Then the last column of width 0.18 is made up with the smallest elements (P6, P7 and P8). Thicker lines correspond to the sum of the half-perimeters, which represents the communication overhead. (b) The matrix blocks allocated to each node are evenly partitioned between cores in horizontal slices.

Fig. 4 shows the execution times of the parallel matrix multiplication application with those three different data partitioning schemes. We randomly select two sizes of matrix and run the computational kernel to generate the CPMs of these four nodes. The CPM-based data partition 1 uses the CPMs when the size of matrix for benchmarking is 3000 blocks for each node, and the size of 7000 blocks is used in CPM-based data partition 2. Since the MFPM-based data partitioning scheme is more accurate and better capture different aspects of heterogeneity of heterogeneous multicore nodes than the other two schemes, it achieves the best performance. Fig. 4 demonstrates that CPM-based and Homogeneous data partition schemes are 2 times to 3 times slower than the MFPM-based data partition scheme.

IV. Intra-Node Data Partitioning

In this section, we address balancing the load between the cores of a heterogeneous multicore node. Instead of redesigning parallel applications for this architecture, we are looking to find the optimal configuration to run the applications, using data partitioning algorithms.

The speeds of the cores within a node cannot be considered separately if they execute the application in parallel. They perform computational operations independently only within some range of problem sizes. When the problem does not fit the private memory of the cores, parallel processes would compete for shared resources and consequently would affect the performance of each other. The functional performance model of an individual core can be deterministically defined only in the range of small problem sizes. For larger problem sizes, the speed of the core will depend not only on the problem size but also on the

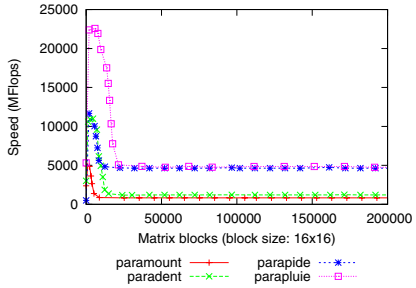

Figure 3. Multicore functional performance models of the Grid5000 nodes specified in Table 1, executing parallel matrix multiplication.

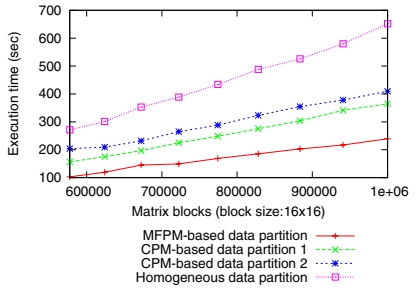

Figure 4. Execution time of parallel matrix multiplication on 16 multicore nodes with MFPM-based, CPM-based and homogeneous data partition schemes.

workload of other cores. Therefore, traditional CPM-based and FPM-based data partitioning algorithms cannot be applied to load balancing on a heterogeneous multicore node. We propose the following load balancing technique:

1. The data of a given size, x, is distributed evenly between the cores, $x_i = x/c, 0 \le i < c$, and the speed of the computational kernel of the routine, s_i, is measured on all cores. To obtain more realistic speed estimates, the processes are synchronized.
2. The data is redistributed, $x = \sum_{i=0}^{c-1} x_i^*$, in proportion to the observed speeds so that: $\frac{x_0^*}{s_0} = ... = \frac{x_{c-1}^*}{s_{c-1}}$.
3. Finally, the parallel routine is executed with this data partition, x_i^*.

Thus, we measure the speeds of the cores at runtime in a group by using the computational kernel (Fig. 1(b)), where each core receives some workload, and distribute data in proportion to the observed speeds. In the following subsection, we present the experimental heterogeneous platforms and the results of load balancing for the parallel matrix multiplication application.

Our experimental platform is Dell PowerEdge 6850. Although the Dell PowerEdge 6850 has a homogeneous architecture, its memory system is typically hierarchy, with two separate FSBs and multi-level caches, which could cause imbalance. In our experiments, we use two experimental setups with uneven process-binding scheme on this platform to simulate a heterogeneous architecture, and we run the computational kernel (Fig. 1(b)) to measure the performance for all the experiments in this section.

1. *Two processes are bound to both cores of socket 0; three processes are bound to the first cores of sockets 1-3.* The first two processes share L3 cache and compete for the first FSB with the third process.
2. *Six processes are bound to both cores of sockets 0-2; one process is bound to the first core of socket 3.* The first three pairs of processes share L3 cache, while the last one treats L3 cache as its own private cache. The numbers of processes sharing FSBs are four vs. three.

Fig. 5 illustrates the speeds of the cores executing the matrix multiplication kernel with equal data distribution. To achieve the balance, we partitioned the problem size to the cores in proportion to their speeds. So these cores can finish the execution within the same time.

Fig. 6 demonstrates the speedup of the computational kernel of the 2D matrix multiplication routine employing intra-node load balancing over even data distribution on the simulated heterogeneous multicore architectures. The speedup is calculated as a ratio of the execution times measured for the multicore node. With the help of the proposed technique, we achieve up 35% speedup.

Acknowledgement. This publication has emanated from research conducted with the financial support of SFI 08/IN.1/I2054 and the UCD-CSC joint scholarship.

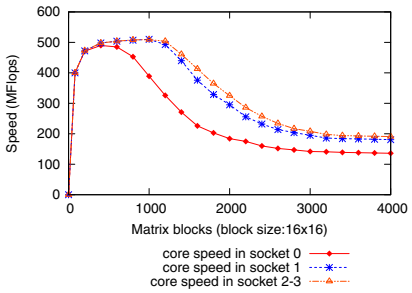

Figure 5. Speed of five cores executing the matrix multiplication kernel with equal data distribution.

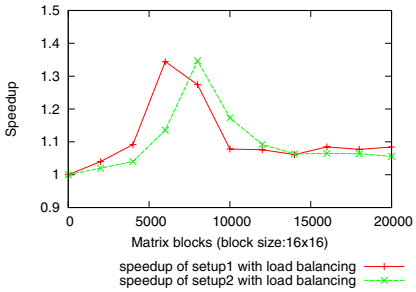

Figure 6. Speedup of the 2D matrix multiplication computational kernel employing intra-node load balancing over even data distribution on the simulated heterogeneous multicore architectures.

REFERENCES

[1] A. Lastovetsky, and R. Reddy, "Data Partitioning with a Functional Performance Model of Heterogeneous Processors," Int. J. High Perform. Comput. Appl., vol. 21, pp. 76-90, Feb. 2007.

[2] V. Soteriou, H. Wang, and L. Peh, "A Statistical Traffic Model for On-Chip Interconnection Networks," in Symp. MASCOTS, 2006, p. 104.

[3] M. Tikir, L. Carrington, E. Strohmaier, and A. Snavely, "A genetic algorithms approach to modeling the performance of memory-bound computations," in SC'07, 2007, p. 1.

[4] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," Commun. ACM, vol. 52, pp. 65-76, Apr. 2009.

[5] M. Hill, and M. Marty, "Amdahl's Law in the Multicore Era," Computer, vol. 41, pp. 33-38, July 2008.

[6] A. Lastovetsky, R. Reddy, "HeteroMPI: Towards a message-passing library for heterogeneous networks of computers," J. Parallel Distr. Com., 2006, vol. 66, pp. 197-220, Feb. 2006.

[7] D. Eadline. (2007) MPI on multicore, an OpenMP alternative? Linux Magazine. [Online]. Available: http://www.linux-mag.com/id/4608/.

[8] J. McCalpin. (1995) STREAM: Sustainable Memory Bandwidth in High-Performance Computers. [Online]. Available: http://www.cs.virginia.edu/stream/.

[9] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," Princeton Univ., Tech. Rep. TR-811-08, 2008.

[10] S. Carr, and K. Kennedy, "Improving the ratio of memory operations to floating-point operations in loops," ACM T. Progr. Lang. Sys. Vol. 16, pp. 1768-1810, Nov. 1994.

[11] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix Multiplication on Heterogeneous Platforms," IEEE Trans. Parallel Distrib. Syst., vol. 12, vol. 1033-1051, Oct 2001.