Dynamic Load Balancing of Parallel Computational Iterative Routines on Platforms with Memory Heterogeneity

David Clarke, Alexey Lastovetsky, Vladimir Rychkov

School of Computer Science and Informatics, University College Dublin,
Belfield, Dublin 4, Ireland
David.Clarke.1@ucdconnect.ie, {Alexey.Lastovetsky, vladimir.rychkov}@ucd.ie

Abstract. Traditional load balancing algorithms for data-intensive iterative routines can successfully load balance relatively small problems. We demonstrate that they may fail for large problem sizes on computational clusters with memory heterogeneity. Traditional algorithms use too simplistic models of processors' performance which cannot reflect many aspects of heterogeneity. This paper presents a new dynamic load balancing algorithm based on the advanced functional performance model. The model consists of speed functions of problem size, which are built adaptively from a history of load measurements. Experimental results demonstrate that our algorithm can successfully balance data-intensive iterative routines on parallel platforms with memory heterogeneity.

Keywords: iterative algorithms; dedicated heterogeneous platforms; dynamic load balancing; data partitioning; functional performance models of heterogeneous processors.

1 Introduction

In this paper we study load balancing of data-intensive parallel iterative routines on heterogeneous platforms. These routines are characterised by a high data-to-computation ratio in a single iteration. The computation load of a single iteration can be broken into any number of equal independent computational units [1]. Each iteration is dependent on the previous one. The generalised scheme of these routines can be summarised as follows: (i) data is partitioned over the processors, (ii) at each iteration some independent calculations are carried out in parallel, and (iii) some data synchronisation takes place. Typically computational workload is directly proportional to the size of data. Examples of scientific computational routines include Jacobi method, mesh-based solvers, signal processing and image processing.

Our target architecture is a dedicated cluster with heterogeneous processors and heterogeneous distributed memory. High performance of iterative routines on this platform can be achieved when all processors complete their work within the same time. This is achieved by partitioning the computational workload and, hence, data

unevenly across all processors. Workload should be distributed with respect to the processor speed, memory hierarchy and communication network [2]. Load balancing of parallel applications on heterogeneous platforms has been widely studied for different types of applications and in various aspects of heterogeneity. Many load balancing algorithms are not appropriate to either the applications or platforms considered in this paper. Applicable algorithms use models of processors' performance which are too simplistic. These traditional algorithms are suitable for problem sizes, which are small relative to the platform, but can fail for larger problems.

This paper presents a new dynamic load balancing algorithm for data-intensive iterative routines on computational clusters with memory heterogeneity. In contrast to the traditional algorithms, our algorithm is adaptive and takes into account heterogeneity of processors and memory. Load balancing decisions are based on functional performance models which are constantly improved with each iteration [3]. Use of the functional performance models remove restrictions on the problem size which can be computed. This allows a computational scientist to utilise the maximum available resources on a given cluster. We demonstrate that our algorithm succeeds in balancing the load even in situations when traditional algorithms fail.

This paper is structured as follows. In Section 2, related work is discussed. In Section 3, we describe the target class of iterative routines and the traditional load balancing algorithm. Then we analyse the shortcomings of the traditional algorithm and present experimental results. In Section 4, we describe our algorithm and demonstrate that it can successfully balance data-intensive iterative routines with large problem sizes.

2 Related Work

In this section, we classify load balancing algorithms and discuss their applicability to data-intensive iterative routines and dedicated computational clusters with memory heterogeneity.

Load balancing algorithms can be either static or dynamic. **Static** algorithms [4, 5, 6] use *a priori* information about the parallel application and platform. This information can be gathered either at compile-time or run-time. These strategies are restricted to applications with pre-determined workload and cannot be applied to such iterative routines as adaptive mesh refinement [7], for which the amount of computation data grows unpredictably. **Dynamic** algorithms [8, 9, 10, 11, 12] do not require *a priori* information and can be used with a wider class of parallel applications. In addition, dynamic algorithms can be deployed on non-dedicated platforms. The algorithm we present in this paper is dynamic.

Another classification is based on how load balancing decisions are made: in a centralised or non-centralised manner. In **non-centralised** algorithms [11, 12], load is migrated locally between neighbouring processors, while in **centralised** ones [4, 5, 6, 8, 9, 10], load is distributed based on global load information. Non-centralized algorithms are slower to converge. At the same time, centralized algorithms typically have higher overhead. Our algorithm belongs to the class of centralised algorithms.

Centralised algorithms can be subdivided into two groups: task queue and predicting the future [2]. **Task queue** algorithms [9, 10] distribute tasks. They target parallel routines consisting of independent tasks and schedule them on shared-memory platforms. **Predicting-the-future** algorithms [4, 5, 6, 8] can distribute both tasks and data by predicting future performance based on past information. They are suitable for data-intensive iterative routines and any parallel computational platform.

A traditional approach taken for load balancing of data-intensive iterative routines belongs to static/dynamic centralised predicting-the-future algorithms. In these traditional algorithms, computation load is evaluated either in the first few iterations [6] or at each iteration [8] and globally redistributed among the processors. Current load measurements are used for prediction of future performance. Neither memory structure nor memory constraints are taken into account. As it will be demonstrated in Section 3, when applied to large scientific problems and parallel platforms with memory heterogeneity, this strategy may never balance the load, because it uses simplistic models of processors' performance.

It has been shown in [13] that it is more accurate to represent performance as a function of problem size, which reflects contributions from both processor and memory. In this paper, we propose a new dynamic load balancing algorithm based on partial functional performance models of processors [3]. Unlike traditional algorithms, our algorithm imposes no restriction on problem sizes.

We would also like to mention some advanced load balancing strategies which are not directly applicable to data-intensive iterative routines on heterogeneous clusters. It has been shown that the task queue model implemented in [10] can outperform the model [9] because decisions are based on adaptive speed measurements rather then single speed measurements. The algorithm presented in this paper also applies an adaptive performance model, but in such a way that it is applicable to scientific computational iterative routines.

In this paper, we focus on dynamic load balancing with respect to processor performance and memory hierarchy, and to this end we do not take into account communication heterogeneity. Future work could be the development of a hybrid approach, similar to [5], in which our algorithm is combined with one of the many existing communication models.

3 Traditional Load Balancing Algorithm of Iterative Routines

Iterative routines have the following structure: $x^{k+1} = f(x^k),\ k = 0,1,...$ with x^0 given, where each x^k is an n-dimensional vector, and f is some function from $\mathbb{R}^n$ into itself [12]. The iterative routine can be parallelized on a cluster of p processors by letting x^k and f be partitioned into p block-components. In an iteration, each processor calculates its assigned elements of x^{k+1}. Therefore, each iteration is dependent on the previous one.

The objective of load balancing algorithms for iterative routines is to distribute computations across a cluster of heterogeneous processors in such a way that all processors will finish their computation within the same time and thereby minimising

the overall computation time: $t_i \approx t_j,\ 1 \le i, j \le p$. The computation is spread across a cluster of p processors $P_1,\ldots,P_p$ such that $p \ll n$. Processor P_i contains d_i elements of x^k and f, such that $n = \sum_{i=1}^{p} d_i$.

Traditional load balancing algorithms work by measuring the computation time of one iteration, calculating the new distribution and redistributing the workload, if necessary, for the next iteration. The algorithm is as follows:

Initially. The computation workload is distributed evenly between all processors, $d_i^0 = n/p$. All processors execute n/p computational units in parallel.

At each iteration.

1) The computation execution times $t_1(d_1^k),\ldots,t_p(d_p^k)$ for this iteration is measured on each processor and gathered to the root processor.
2) If $\max_{1 \le i,j \le p} \left| \frac{t_i(d_i^k) - t_j(d_j^k)}{t_i(d_i^k)} \right| \le \varepsilon$ then the current distribution is considered balanced and redistribution is not needed.
3) Otherwise, the root processor calculates the new distribution of computations $d_1^{k+1},\ldots,d_p^{k+1}$ as $d_i^{k+1} = n \times s_i^k / \sum_{j=1}^{p} s_j^k$ where s_i^k is the speed of the *i'th* processor given by $s_i^k = d_i^k / t_i(d_i^k)$.
4) The new distribution $d_1^{k+1},\ldots,d_p^{k+1}$ is broadcast to all processors and where necessary data is redistributed accordingly.

3.1 Analysis of Traditional Load Balancing

The traditional load balancing algorithm is based on the assumption that the absolute speed of a processor depends on problem size but the speed is represented by a constant at each iteration. This is true for small problem sizes as depicted in Fig. 1(a). The problem is initially divided evenly between two processors for the first iteration and then redistributed to the optimal distribution in the second iteration.

Consider the situation in which the problem can still fit within the total main memory of the cluster but the problem size is such that the memory requirement of n/p is close to the available memory of one of the processors. In this case paging can occur. If paging does occur, the traditional load balancing algorithm is no longer adequate. This is illustrated for two processors in Fig. 1(b, c). Let the real performance of processors P_1 and P_2 be represented by the speed functions $s_1(x)$ and $s_2(x)$ respectively. Processor P_1 is a faster processor but with less main memory than P_2. The speed function drops rapidly at the point where main memory is full and paging is required. First, n independent units of computations are evenly distributed, $d_1^0 = d_2^0 = n/2$, between the two processors and the speeds of the processors, s_1^0, s_2^0 , are measured. Then at the second iteration the computational units are divided

according to $\dfrac{d_1^1}{d_2^1} = \dfrac{s_1^0}{s_2^0}$, where $d_1^1 + d_2^1 = n$. Therefore in the second iteration, P_1 will execute less computational units than P_2. However P_1 will perform much faster and P_2 will perform much slower than the model predicts, Fig. 1(b). Moreover the speed of P_2 at the second iteration is slower then P_1 at the first iteration.

Based on the speeds of the processors demonstrated at the second iteration, their constant performance models are changed accordingly, Fig. 1(c), and the computational units are redistributed again for the third iteration as: $\dfrac{d_1^2}{d_2^2} = \dfrac{s_1^1}{s_2^1}$, where $d_1^2 + d_2^2 = n$. Now the situation is reversed, P_2 performs much faster than P_1. This situation will continue in subsequent iterations with the majority of the computational units oscillating between processors.

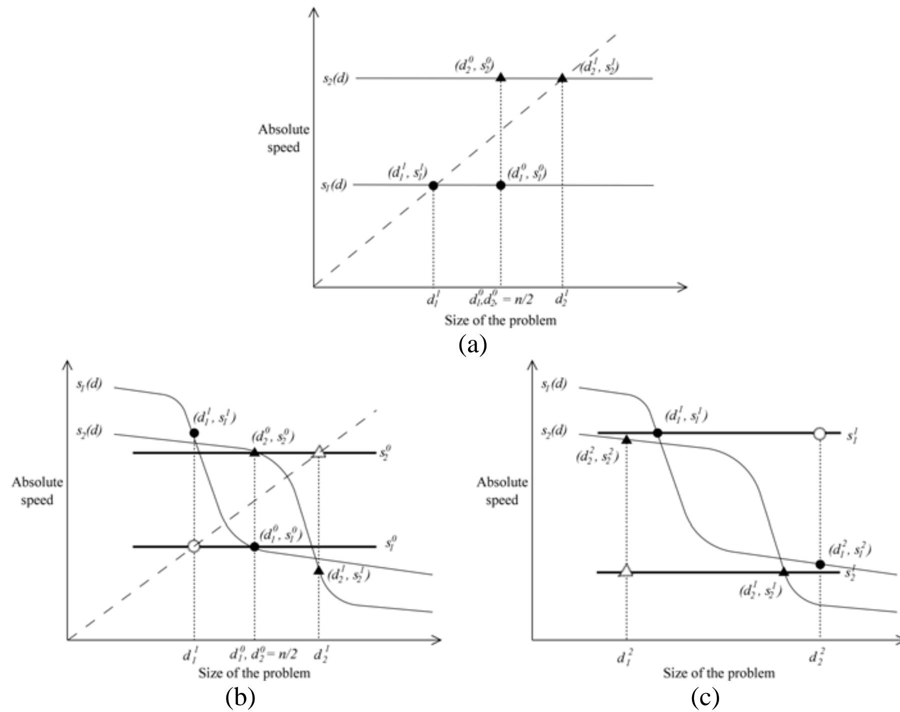


Fig. 1. Predicted results from dynamic load balancing on two processors using constant performance models. In (a) the problem size is small relative to available main memory and balance is achieved. In (b, c) the problem size is large and may require paging, the balancing algorithm causes further unbalance. (b) shows first and second iterations, (c) shows second and third iterations. Outlined points represent performance predicted by constant performance model.

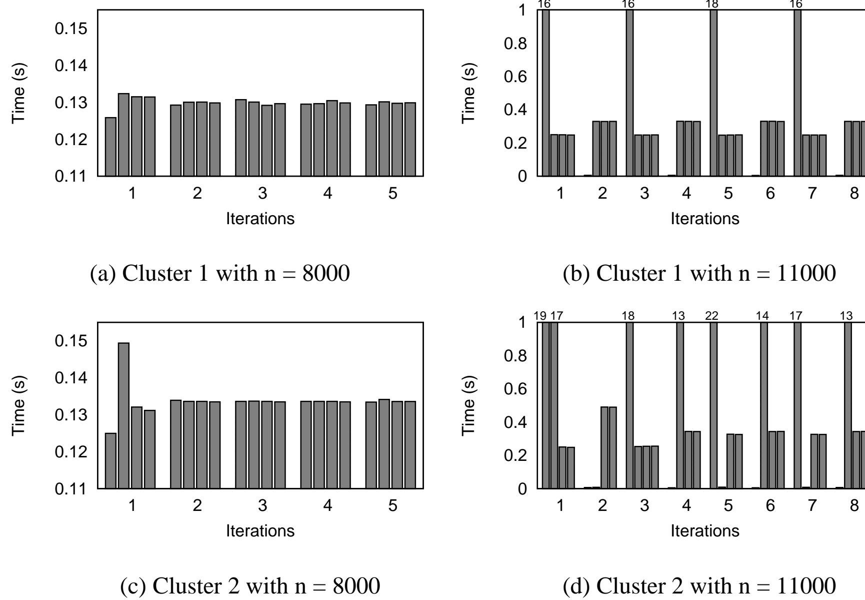
3.2 Experimental Results of the Traditional Load Balancing Algorithm

The traditional load balancing algorithm was applied to the Jacobi method, which is representative of the class of iterative routines we study. The program was tested successfully on a cluster of 16 processors. For clarity the results presented here are from two configurations of 4 processors, Table 1. The essential difference is that cluster 1 has one processor with 256MB RAM and cluster 2 has two processors with 256MB RAM.

Table 1. Specifications of test nodes. Cluster 1 consists of nodes: P_1, P_3, P_4, P_5. Cluster 2 consists of nodes: P_1, P_2, P_3, P_4.

	P_1	P_2	P_3	P_4	P_5
Processor	3.6 Xeon	3.0 Xeon	3.4 P4	3.4 Xeon	3.4 Xeon
Ram (MB)	256	256	512	1024	1024

The memory requirement of the partitioned routine is a $n \times d_i$ block of a matrix, three *n* dimensional vectors and some additional arrays of size *p*. For 4 processors with an even distribution, problem sizes of *n*=8000 and *n*=11000 will have a memory requirement which lies either side of the available memory on the 256MB RAM machines, and hence will be good values for benchmarking.



(a) Cluster 1 with n = 8000

(b) Cluster 1 with n = 11000

(c) Cluster 2 with n = 8000

(d) Cluster 2 with n = 11000

Fig. 2. Time taken for each of the 4 processors to complete their assigned computational units for each iteration 1,2,3,… . In (a) and (c) the problem fits in main memory the load converges to a balanced solution. In (b) and (d) paging occurs on some machines and the load remains unbalanced.

The traditional load balancing algorithm worked efficiently for small problem sizes, Fig. 2(a, c). For problem sizes sufficiently large to potentially cause paging on some machines the load balancing algorithm caused divergence as the theory, in section 2.1, predicted, Fig. 2 (b,d).

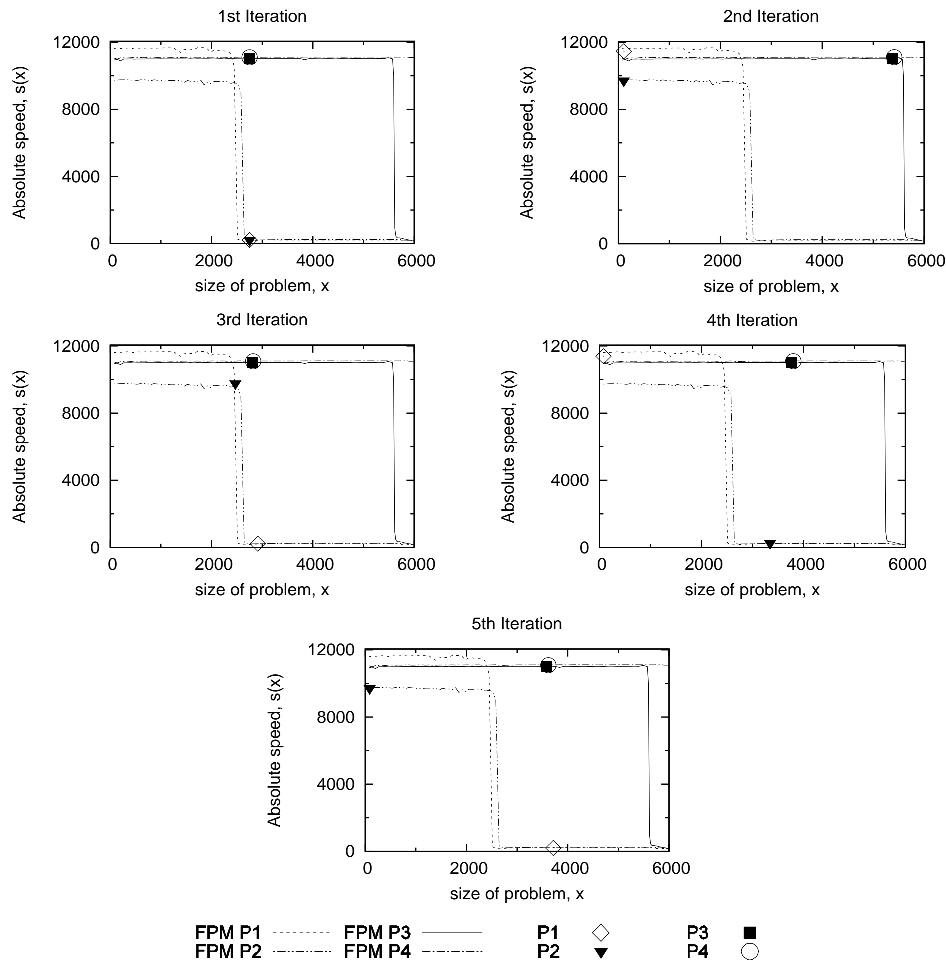


Fig. 3. Traditional load balancing algorithm for four processors on cluster 2 with *n*=11000. Showing initial distribution at n/4 and four subsequent iterations. The x axis represents the number of rows of the matrix are held in memory, and the number of elements of x′ computed by each processor. The full functional performance models are dotted in to aid visualisation.

A plot of problem size vs. absolute speed can help illustrate why the traditional load balancing algorithm is failing for large problems. Fig. 3 shows the absolute speed of each of the processors for the first five iterations. The experimentally built full functional model for each processor are dotted in to aid visualisation but this information was not available to the load balancing algorithm. Initially each processor

has *n/4* rows of the matrix. In the second iteration, P_1 and P_2 are given very few rows as they both performed slowly in the first iteration, however they now compute these few rows quickly. In the third iteration, P_1 is given sufficient rows to cause paging and hence a cycle of oscillating row allocation ensues.

Since data partitioning is employed in our iterative routine, it is necessary to do data redistribution with each rebalancing. When the balancing algorithm converges quickly to an optimum distribution the network load from data redistribution is acceptable. However as the distribution oscillates not only is the computation time affected but so too is the network load. On cluster 2 with n=11000 approximately 300MB is been passed back and forth between P_1 and P_2 with each iteration.

4 Dynamic Load Balancing Based on Accurate Evaluation of Computation Load and Memory Hierarchy

Our dynamic load balancing algorithm is based on functional performance models [13], which are application centric and hardware specific. Functional performance models reflect both processor and memory heterogeneity. In this section, we describe how the load can be balanced with help of these models.

The functional performance models of the processors are represented by their speed functions $s_1(d),\ldots,s_p(d)$, with $s_i(d) = d / t_i(d)$, where $t_i(d)$ is the execution time for processing of d elements on the processor P_i. As in traditional algorithms, load balancing is achieved when $t_i \approx t_j,\ 1 \le i, j \le p$. This can be expressed as $\frac{d_1}{s_1(d_1)} \approx \frac{d_2}{s_2(d_2)} \approx \ldots \approx \frac{d_p}{s_p(d_p)}$, where $d_1 + d_2 + \ldots + d_p = n$. These equations can be solved geometrically by intersection of the speed functions with a line passing through the origin of the coordinate system (Fig. 4). This approach can be used for static load balancing.

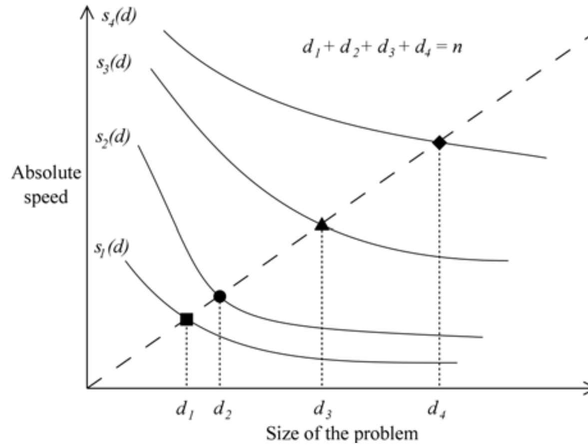


Fig. 4. Optimal distribution of computational units showing the geometric proportionality of the number of chunks to the speed of the processor.

Functional performance models are built experimentally. Their accuracy depends on the number of experimental points. Unfortunately, generating these speed functions is computationally expensive, especially in the presence of paging. To create just 20 points of a function in Fig. 3 took approximately 1473seconds, 4 times longer then the actual calculation with a homogeneous distribution for 20 iterations. This forbids building full functional performance models at run time. However, in this paper, we apply partial functional performance models to dynamic load balancing of iterative routines. The partially built performance models are piecewise linear approximations of the real speed functions, $s'_i(d) \approx s_i(d)$, which estimate the real functions in detail only in the relevant regions [3]. The low cost of partially building the models makes it ideal for employment in self-adaptive parallel applications. The partial models can be built during the execution of the computational iterative routine.

We modified the traditional dynamic load balancing algorithm, presented in Section 2, using partial speed functions instead of single speed values. The partial functions $s'_i(d)$ are built by adding an experimental point (d_i^k, s_i^k) after each iteration of the routine. The more points are added, the closer the partial function approximates the real speed function in the relevant region. At each iteration, we apply the balance criteria to find a new distribution $d_1^{k+1},...,d_p^{k+1}$ by solving the system of equations: $\frac{d_1^{k+1}}{s'_1(d_1^{k+1})} \approx \frac{d_2^{k+1}}{s'_2(d_2^{k+1})} \approx \ldots \approx \frac{d_p^{k+1}}{s'_p(d_p^{k+1})}$, $d_1^{k+1} + d_2^{k+1} + ... + d_p^{k+1} = n$. In few iterations, our algorithm will adaptively converge to the optimal data distribution, since $s'_i(d) \to s_i(d)$. Let us outline how the partial functions $s'_i(d)$ are constructed.

The first iteration. The speed of each processor is calculated as $s_i^0 = \frac{n/p}{t_i(n/p)}$. The first approximation of the partial speed function, $s'_i(d)$, is created as a constant $s'_i(d) = s_i^0$, Fig. 5(a).

Subsequent iterations. The speed of each processor is calculated as $s_i^k = d_i^k / t_i(d_i^k)$. The piecewise linear approximations $s'_i(d)$ are improved by adding the points (d_i^k, s_i^k), Fig. 5(b). Namely, let $\{(d_i^{(j)}, s_i^{(j)})\}_{j=1}^m$, $d_i^{(1)} < \ldots < d_i^{(m)}$, be the experimentally obtained points of $s'_i(d)$ used to build its current piecewise linear approximation, then

- If $d_i^k < d_i^{(1)}$, then the line segment $(0, s_i^{(1)}) \to (d_i^{(1)}, s_i^{(1)})$ of the $s'_i(d)$ approximation will be replaced by two connected line segments $(0, s_i^k)) \to (d_i^k, s_i^k)$ and $(d_i^k, s_i^k) \to (d_i^{(1)}, s_i^{(1)})$;
- If $d_i^k > d_i^{(m)}$, then the line $(d_i^{(m)}, s_i^{(m)}) \to (\infty, s_i^{(m)})$ of this approximation will be replaced by the line segment $(d_i^{(m)}, s_i^{(m)}) \to (d_i^k, s_i^k)$ and the line $(d_i^k, s_i^k) \to (\infty, s_i^k)$;
- If $d_i^{(j)} < d_i^k < d_i^{(j+1)}$, the line segment $(d_i^{(j)}, s_i^{(j)}) \to (d_i^{(j+1)}, s_i^{(j+1)})$ of $s'_i(d)$ will be replaced by two connected line segments $(d_i^{(j)}, s_i^{(j)}) \to (d_i^k, s_i^k)$ and $(d_i^k, s_i^k) \to (d_i^{(j+1)}, s_i^{(j+1)})$.

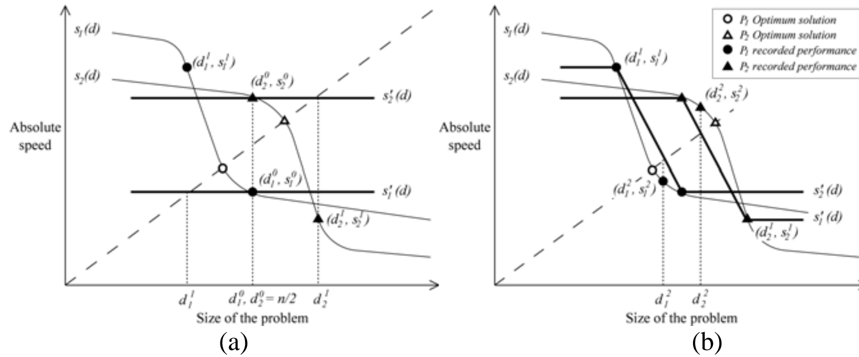


Fig. 5. Dynamic load balancing using partial estimation of the functional performance model.

4.1 Experimental Results

For small problem sizes (n = 8000, p = 4), our algorithm performed in much the same way as the traditional algorithm. For larger problem sizes (n = 11000), our algorithm was able to successfully balance the computational load within a few iterations (Fig. 6). As in the traditional algorithm, paging also occurred but our algorithm experimentally fit the problem to the available RAM. Paging at the 8th iteration on P_1 demonstrates how the algorithm experimentally finds the memory limit of P_1. The 9th iteration represents a near optimum distribution for the computation on this hardware.

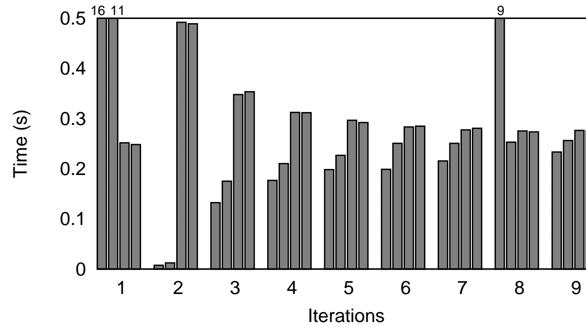


Fig. 6. Time taken for each of the 4 processors to complete their task for each iteration. These results are from the same experiment as fig. 5 with problem size n=11000.

A plot of speed vs. problem size, Fig. 7, shows how the computational distribution approaches an optimum distribution within 9 iterations. We can see why P_1 performs slowly at the 8th iteration. At the 9th iteration in Fig. 7, we can see that the maximum performance of processors P_1 and P_2 has been achieved.

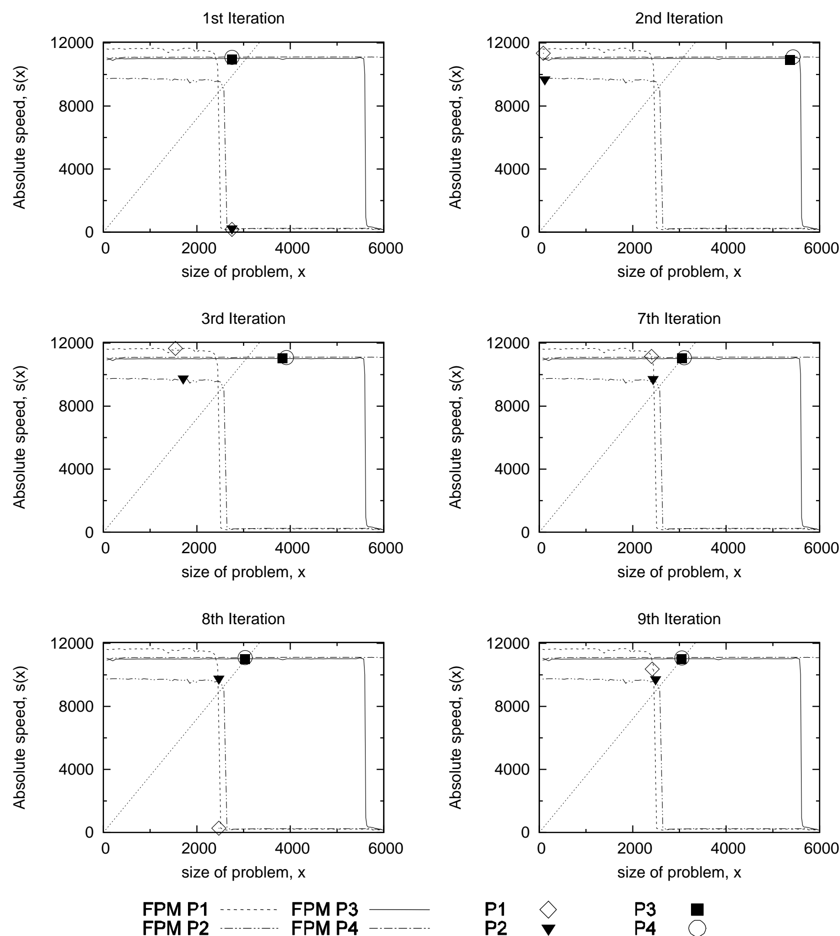


Fig. 7. Experimental results from load balancing using partial estimation of the functional performance model with n=11000. Showing the 1st, 2nd, 3rd, 7th, 8th and 9th iterations. The line intersecting the origin represents the optimum solution and points converge towards this line.

5 Conclusion

In this paper, we have shown that traditional dynamic load balancing algorithms can fail for large problem sizes on parallel platforms with memory heterogeneity. They do not take into account memory hierarchy and use simplified models of processors' performance. We have shown that our dynamic load balancing algorithm, in which performance is represented by a function of problem size, can be used successfully with any problem size and on a wide class of heterogeneous platforms.

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/IN.1/I2054.

References

1. Bharadwaj, V., Ghose, D., Robertazzi, T.G.: Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. Cluster Comput. 6, 7--17 (2003)
2. Cierniak, M., Zaki, M.J., Li, W.: Compile-Time Scheduling Algorithms for Heterogeneous Network of Workstations. Computer J. 40, 356--372 (1997)
3. Lastovetsky, A., Reddy, R.: Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of their Functional Performance Models. In: HeteroPar'2009. LNCS, vol. 6043, pp. 91--101. Springer (2010)
4. Ichikawa, S., Yamashita, S.: Static Load Balancing of Parallel PDE Solver for Distributed Computing Environment. In: PDCS-2000, pp. 399--405. ISCA (2000)
5. Legrand, A., Renard, H., Robert, Y., Vivien, F.: Mapping and load-balancing iterative computations. IEEE T. Parall. Distr. 15, 546--558 (2004)
6. Martínez, J.A., Garzón, E.M., Plaza, A., García, I.: Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. J. Supercomput. (to appear)
7. Li, X.-Y., Teng, S.-H.: Dynamic Load Balancing for Parallel Adaptive Mesh Refinement. In: IRREGULAR'98, pp. 144--155. Springer (1998)
8. Galindo, I., Almeida, F., Badía-Contelles, J. M.: Dynamic Load Balancing on Dedicated Heterogeneous Systems. In: EuroPVM/MPI 2008, pp. 64--74. Springer (2008)
9. Hummel, S.F., Schmidt, J., Uma, R. N., Wein, J.: Load-sharing in heterogeneous systems via weighted factoring. In: SPAA'96, pp. 318--328. ACM (1996)
10. Cariño, R.L., Banicescu, I.: Dynamic load balancing with adaptive factoring methods in scientific applications. J. Supercomput. 44, 41--63 (2008)
11. Cybenko, G.: Dynamic load balancing for distributed memory multi-processors. J. Parallel Distr. Com. 7, 279--301 (1989)
12. Bahi, J.M., Contassot-Vivier, S., Couturier, R.: Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms. IEEE T. Parall. Distr. 16, 289--299 (2005)
13. Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. Int. J. High Perform. Comput. Appl. 21, 76--90 (2007)