Using Multidimensional Solvers for Optimal Data Partitioning on Dedicated Heterogeneous HPC Platforms

Vladimir Rychkov, David Clarke, and Alexey Lastovetsky

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

Abstract. High performance of data-parallel applications on heterogeneous platforms can be achieved by partitioning the data in proportion to the speeds of processors. It has been proven that the speed functions built from a history of time measurements better reflect different aspects of heterogeneity of processors. However, existing data partitioning algorithms based on functional performance models impose some restrictions on the shape of speed functions, which are not always satisfied if we try to approximate the real-life measurements accurately enough. This paper presents a new data partitioning algorithm that applies multidimensional solvers to numerical solution of the system of non-linear equations formalizing the problem of optimal data partitioning. This algorithm relaxes the restrictions on the shape of speed functions and uses the Akima splines for more accurate and realistic approximation of the real-life speed functions. The better accuracy of the approximation in its turn results in a more optimal distribution of the computational load between the heterogeneous processors.

Keywords: dedicated heterogeneous HPC platforms; data partitioning; functional performance models of processors; Akima spline interpolation; multidimensional root-finding.

1 Introduction

In this paper, we study partitioning of computational load in data-intensive parallel scientific routines, such as linear algebra, mesh-based solvers, image processing. In these routines, typically, computational workload is directly proportional to the size of data. High performance of these routines on dedicated heterogeneous HPC platforms 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, with respect to the processor speed and memory hierarchy.

Conventional algorithms for distribution of computations between heterogeneous processors are based on a performance model which represents the speed of a processor by a constant positive number, and computations are distributed between the processors in proportion to this speed of the processor. The constant characterizing the performance of the processor is typically its relative speed demonstrated during the execution of a serial benchmark code solving locally the core computational task of some given size.

The fundamental assumption of the conventional algorithms based on the constant performance models (CPMs) is that the absolute speed of the processors does not depend on the size of the computational task. This assumption proved to be accurate enough if:

- The processors, between which we distribute computations, are all general-purpose ones of the traditional architecture,
- The same code is used for local computations on all processors, and
- The partitioning of the problem results in a set of computational tasks that are small enough to fit into the main memory of the assigned processors and large enough not to fit into the cache memory.

These conditions are typically satisfied when medium-sized scientific problems are solved on a heterogeneous network of workstations. Actually, heterogeneous networks of workstations were the target platform for the conventional heterogeneous parallel algorithms. However, the assumption that the absolute speed of the processor is independent of the size of the computational task becomes much less accurate in the following situations:

- The partitioning of the problem results in some tasks either not fitting into the main memory of the assigned processor and hence causing paging or fully fitting into faster levels of its memory hierarchy (Fig. 1).
- Some processing units involved in computations are not traditional general-purpose processors (say, accelerators such as GPUs or specialized cores). In this case, the relative speed of a traditional processor and a non-traditional one may differ for two different sizes of the same computational task even if both sizes fully fit into the main memory.
- Different processors use different codes to solve the same computational problem locally.

The above situations become more and more common in modern and especially perspective high-performance heterogeneous platforms. As a result, applicability of the traditional CPM-based distribution algorithms becomes more restricted. Indeed, if we consider two really heterogeneous processing units P_i and P_j, then the more different they are, the smaller will be the range R_{ij} of sizes of the computational task where their relative speed can be accurately approximated by a constant. In the case of several different heterogeneous processing units, the range of sizes where CPM-based algorithms can be applied will be given by the intersection of these pair-wise ranges, $\bigcap_{i,j=1}^{p} R_{ij}$. Therefore, if a high-performance computing platform includes a

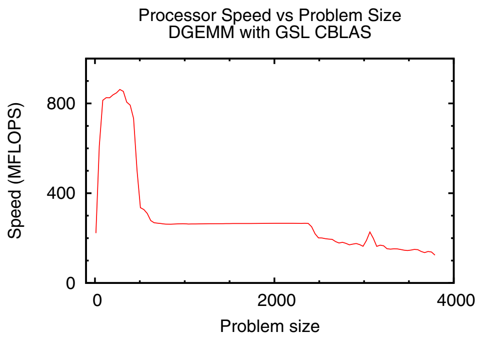

Fig. 1. Processor speed observed for matrix multiplication routine in different ranges of problem size

relatively large number of significantly heterogeneous processing units, the area of applicability of CPM-based algorithms may become quite small or even empty. For such platforms, new algorithms are needed that would be 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 analyzed in [1] has proven to be more realistic than the constant performance models 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. This model is application centric because, generally speaking, different applications will characterize the speed of the processor by different functions.

The cost of experimentally building the full functional performance model of a processor, i.e., the model for the full range of problem sizes, is very high. This is due to several reasons. To start with, the accuracy of the model depends on the number of experimental points used to build it. The larger the number, the more accurate the model is. However, there is a cost associated with obtaining an experimental data point, which requires execution of a computational kernel for a specified problem size. This cost is especially high for problem sizes in the region of paging.

The high model-construction cost limits the applicability of parallel algorithms based on full FPMs to situations where the construction of the full FPMs of heterogeneous processors and their use in the application can be separated. For example, if we develop an application for dedicated stable heterogeneous platforms, with the intention of executing the application on the same platform multiple times, we can build the full FPMs for each processor of the platform once and then use these models multiple times during the repeated execution of the application. In this case, the time of construction of the FPMs can become very small compared to the accumulated performance gains during the multiple executions of the optimized application. However, this approach does not apply to applications for which each run is considered unique. This is the case for applications that are intended to be executed in dynamic environments or any other environments where the available processors and their performance characteristics can change.

In contrast to the CPM-based data partitioning algorithms, the FPM-based ones take into account the history of load measurements stored in the speed functions. The FPM-based algorithms define the optimal data distribution through the speed functions. Traditionally, the speed functions are built as piecewise linear functions fitting historic records of the processors' workload. The piecewise linear approximation of the speed functions is used by the geometrical data partitioning algorithm proposed in [1]. It imposes some restrictions on the shape of speed functions but guarantees the existence and uniqueness of the optimal data partitioning.

In this paper, we present a new data partitioning algorithm. This new algorithm formulates the original data partitioning problem as a problem of finding a solution to a multi-dimensional system of nonlinear equations. It employs a numerical multi-dimensional non-linear solver to find the optimal partitioning. It is not that restrictive to the shape of speed functions as the geometrical algorithm and therefore can use more accurate approximations of the real-life speed functions. However, the proposed algorithm does not guarantee a unique solution. In this paper, we propose to interpolate the speed functions by the Akima splines. Passing through all experimental points, the Akime splines form a smooth function of a shape that closely reflects the real performance of the processor.

To demonstrate the advantages of the new FPM-based data partitioning algorithm, we present the experimental results of dynamic load balancing of data-intensive iterative routines on highly heterogeneous computational clusters. In our experiments, the speed functions of the processors are dynamically constructed during the iterations of the routine. Use of the functional performance models allows a computational scientist to utilise the maximum available resources on a given cluster. We demonstrate that our algorithm succeed in balancing the load even in situations when the traditional algorithms fail. We show that the Akima splines provide very accurate approximation of the speed functions.

This paper is structured as follows. In Section 2, we describe traditional piecewise linear approximation of speed functions and the geometrical data partitioning algorithm. In Section 3, we present the new FPM-based data partitioning algorithm based on multidimensional solvers. The algorithm uses the Akima spline interpolation of the speed functions. In Section 4, we demonstrate that this algorithm can successfully balance data-intensive iterative routines for the whole range of problem sizes.

2 Traditional Piecewise Linear Approximation of Speed Functions and Geometrical Data Partitioning Algorithm

Functions much more accurately represent the speed of processors than constants [2]. Being application-centric and hardware-specific, functional performance models reflect different aspects of heterogeneity. In this section, we describe a traditional approach to approximate the speed of the processors and the geometrical data partitioning algorithm based on this approach.

Let speeds of p processors be represented by positive continuous functions of problem size $s_1(x),...,s_p(x)$: $s_i(x) = \frac{x}{t_i(x)}$, where $t_i(x)$ is the execution time for processing of x elements on the processor i. Speed functions are defined at $[0,n]$, where n is a problem size to partition. The optimal data partition is achieved when all processors execute their work at the same time: $t_1(x_1) \approx ... \approx t_p(x_p)$. This can be expressed as:

$$\frac{x_1}{s_1(x_1)} = ... = \frac{x_p}{s_p(x_p)}, \text{ where } x_1 + x_2 + ... + x_p = n . \tag{1}$$

The integer-value solution of these equations, $d_1,...,d_p$, can be represented geometrically by intersection of the speed functions with a line passing through the origin of the coordinate system (Fig. 2).

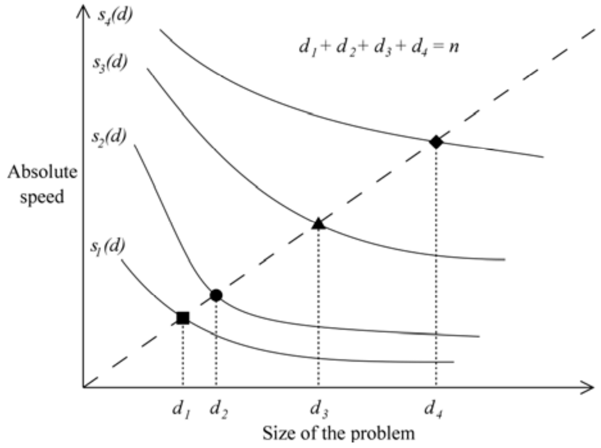

Fig. 2. Optimal distribution of computational units showing the geometric proportionality of the number of chunks to the speed of the processor

2.1 Data Partitioning Algorithm Based on Geometrical Solution

The geometrical data partitioning algorithm is based on the geometrical solution of the problem (1), assuming that any straight line passing through the origin of the coordinate system intersects the speed functions only once. The algorithm can be summarized as follows. Any line passing through the origin and intersecting all speed functions represents an optimum distribution for a particular problem size. Therefore the space of solutions of the problem (1) consists of all such lines. The two outer bounds of the solution space are selected as the starting point of algorithm. The upper line represents the optimal data distribution $d_1^u,...,d_p^u$ for some problem size $n_u < n$,

$n_u = d_1^u + ... + d_p^u$, while the lower line gives the solution $d_1^l,...,d_p^l$ for $n_l > n$, $n_l = d_1^l + ... + d_p^l$. The region between two lines is iteratively bisected. At the iteration k, the problem size corresponding to the new line intersecting the speed functions at the points $d_1^k,...,d_p^k$ is calculated as $n_k = d_1^k + ... + d_p^k$. Depending on whether n_k is less than or greater than n, this line becomes a new upper or lower bound. Making n_k close to n, this algorithm finds the optimal partition of the given problem $d_1,...,d_p$: $d_1 + ... + d_p = n$. The assumptions about the shape of the speed functions provide the existence and uniqueness of the solution. Fig. 3 illustrates the work of the bisection algorithm.

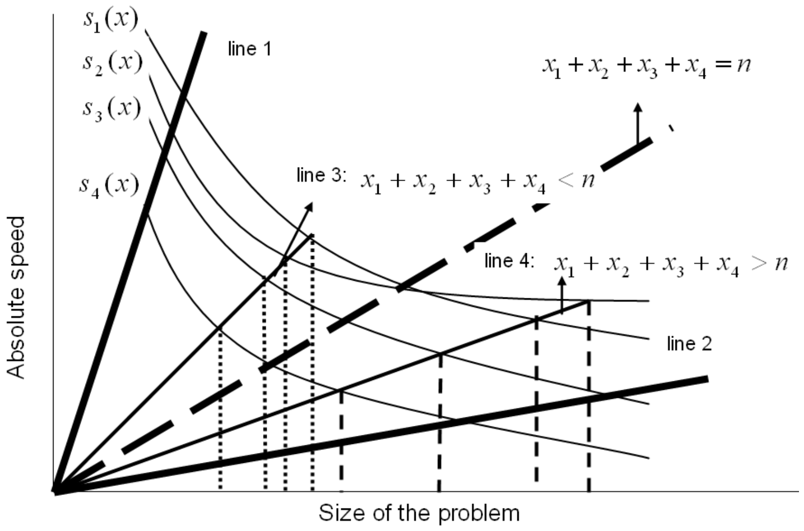
$s_1(x)$
$s_2(x)$
$s_3(x)$
$s_4(x)$
line 1
$x_1 + x_2 + x_3 + x_4 = n$
line 3: $x_1 + x_2 + x_3 + x_4 < n$
line 4: $x_1 + x_2 + x_3 + x_4 > n$
line 2
Absolute speed
Size of the problem

Fig. 3. Geometrical data partitioning algorithm. Line 1 (the upper line) and line 2 (the lower line) represent the two initial outer bounds of the solution space. Line 3 represents the first bisection. Line 4 represents the second one. The dashed line represents the optimal solution.

To apply this algorithm to the speed functions, some restrictions are placed on their shape. Experiments performed with many scientific kernels on various heterogeneous networks of workstations have demonstrated that, in general, processor speed could be approximated, within some acceptable degree of accuracy, by a function satisfying the following assumptions [1]:

- On the interval x $[0, X]$, the function is monotonically increasing and concave.
- On the interval $[X, \infty]$, the function is monotonically decreasing.
- Any straight line coming through the origin of the coordinate system intersects the graph of the function in no more than one point.

2.2 Piecewise Linear Approximation of Speed Functions

Traditionally, the speed function is built as a piecewise linear function fitting within a band of historic records of workload fluctuations of the processor. To satisfy the above assumptions, the shape of the piecewise linear approximation $s_i(x)$ should be verified after adding the new data point (d_i^j, s_i^j), and the value of s_i^j should be updated when required. Namely, to keep the speed function increasing and convex on the interval $[0, X]$, it is necessary that $\frac{s_i^{j-1} - s_i^{j-2}}{d_i^{j-1} - d_i^{j-2}} > \frac{s_i^j - s_i^{j-1}}{d_i^j - d_i^{j-1}} > \frac{s_i^{j+1} - s_i^j}{d_i^{j+1} - d_i^j} > 0$. This expression represents decreasing tangent of the pieces, which is required for the convex shape of the piecewise linear approximation. On the interval $[X, \infty]$, it is necessary that $s_i^{j-1} \geq s_i^j \geq s_i^{j+1}$ for monotonously decreasing speed function.

The procedure of building piecewise linear approximation is very time consuming, especially for full functional performance models, which are characterized by numerous points. Generating the speed functions is especially expensive in the presence of paging. This forbids building full functional performance models at run time. To reduce the cost of building the speed functions, the partial functional performance models were proposed [3]. They are based on a few points and estimate the real functions in detail only in the relevant regions: $\overline{s}_i(x) \approx s_i(x)$, $1 \leq i \leq p$, $\forall x \in [a,b]$. Both the partial models and the regions are determined at runtime, while the data partitioning algorithm is iteratively applied to the partially built speed functions. The result of the data partitioning, the estimate of the optimal data distribution $d_1^k, ..., d_p^k$, determines the next experimental points $(d_1^k, s_1(d_i^k)), ..., (d_p^k, s_p(d_p^k))$ to be added to the partial models $\overline{s}_1(x), ..., \overline{s}_p(x)$. The more points are added, the closer the partial functions approximate the real speed functions in the relevant regions. Fig. 4 illustrates the construction of the partial speed functions, using the piecewise linear approximation and the geometrical data partitioning algorithm.

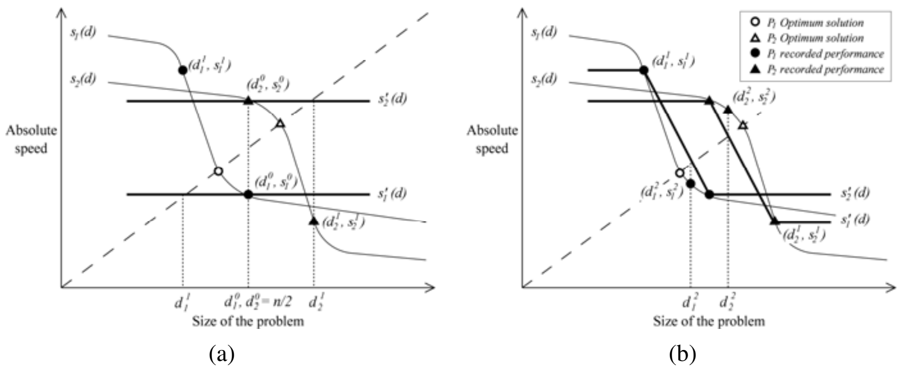

Fig. 4. Construction of the partial speed functions, using the piecewise linear approximation and the geometrical data partitioning algorithm

After adding the experimental points to the partial speed functions, their shape is adjusted to satisfy the above assumptions. The low cost of partial building the models makes it ideal for employment in self-adaptive parallel applications, particularly in dynamic load balancing.

3 New Data Partitioning Algorithm Based on Multidimensional Root-Finding

The geometrical data partitioning algorithm requires each speed function to be monotonically increasing and concave up to some point and then monotonically decreasing and in addition to be intersected only once by any line passing from the origin. In general, speed functions have this shape, but it is not always the case. For example, a non-optimised algorithm such as Netlib BLAS can have a sawtooth function due to cache misses (Fig. 5). A less accurate function must be fitted to the data to satisfy the shape restrictions (Fig. 5(a)).

Here we present a new data partitioning algorithm which allows us to remove these restrictions and therefore represent the speed of the processor with more accurate continuous functions. This allows us to perform more accurate partitioning. For example, by using the more accurate fit in Fig. 5(b), we can achieve a speedup of 1.26 for some problem sizes. For different routines this speedup could potentially be much bigger.

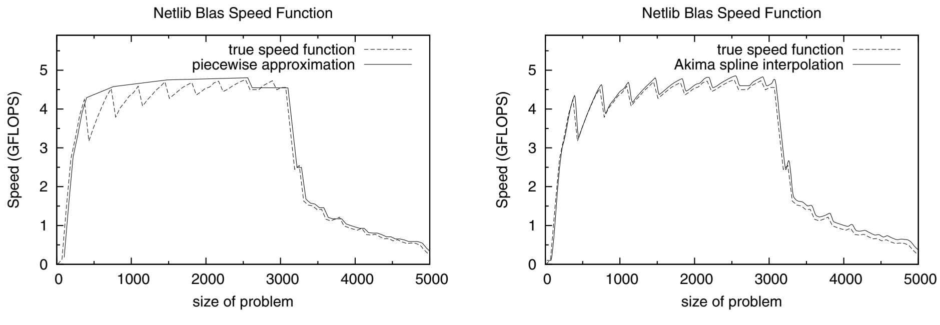

Fig. 5. Speed function for non-optimised Netlib BLAS. (a) Fitting shape restricted piecewise approximation. (b) Fitting Akima spline interpolation. Both speed functions have been offset slightly for clarity.

3.1 Data Partitioning Algorithm Based on Nonlinear Multidimensional Root Finding

If the processor speeds are approximated by continuous differentiable functions of arbitrary shape, the problem of optimal data partitioning (1) can be formulated as ***multidimensional root finding*** for the system of nonlinear equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, where

$$\mathbf{F}(\mathbf{x}) = \begin{cases} n - \sum_{i=1}^{p} x_i \\ \dfrac{x_i}{s_i(x_i)} - \dfrac{x_1}{s_1(x_1)}, & 2 \le i \le p \end{cases}. \tag{2}$$

$\mathbf{x} = (x_1, ..., x_p)$ is a vector of real numbers corresponding a data partition $\mathbf{d} = (d_1, ..., d_p)$. The first equation specifies the distribution of n computational units between p processors. The rest specify the balance of computational load. The problem (2) can be solved by a number of iterative algorithms based on the Newton–Raphson method:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \mathbf{J}(\mathbf{x_k})\mathbf{F}(\mathbf{x_k}). \tag{3}$$

The equal data distribution

$$\mathbf{x}^0 = (n/p, ..., n/p) \tag{4}$$

can be reasonably taken as the initial guess for the location of the root. $\mathbf{J}(\mathbf{x})$ is a Jacobian matrix, which can be calculated as follows:

$$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} -1 & -1 & ... & -1 \\ -\dfrac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & \dfrac{s_2(x_2) - x_2 s_2'(x_2)}{s_2^2(x_2)} & 0 & 0 \\ ... & 0 & ... & 0 \\ -\dfrac{s_1(x_1) - x_1 s_1'(x_1)}{s_1^2(x_1)} & 0 & 0 & \dfrac{s_p(x_p) - x_p s_p'(x_p)}{s_p^2(x_p)} \end{pmatrix}. \tag{5}$$

We use the HYBRJ algorithm, a modified version of Powell's Hybrid method, implemented in the MINPACK library [4]. It retains the fast convergence of the Newton method and reduces the residual when the Newton method is unreliable. Our experiments demonstrated that for the given vector-function (2) and initial guess (4), the HYBRJ algorithm is able to find the root $\mathbf{x}^* = (x_1^*, ..., x_p^*)$, which will be the optimal data partition after rounding and distribution of remainders: $\mathbf{d} = round(\mathbf{x}^*)$.

3.2 New Approximation of Partial Speed Functions Based on the Akima Splines

Let us consider a set of k data points (x_i, s_i), $0 < x_i < n$, $1 \le i \le k$. Here and after in this section, the data points (x_i, s_i) correspond to a single processor, for which the speed function $s(x)$ is approximated. To approximate the speed function in the interval $[0, n]$, we also introduce two extra points: $(0, s_1)$ and (n, s_k). The linear interpolation does not satisfy the condition of differentiability at the breakpoints

(x_i, s_i). The spline interpolations of higher orders have derivatives but may yield significant oscillations in the interpolated function. However, there is a special non-linear spline, the ***Akima spline*** [5], that is stable to outliers (Fig. 6). It requires no less than 5 points. In the inner area $[x_3, x_{k-2}]$, the interpolation error has the order $O(h^2)$. This interpolation method does not require solving large systems of equations and therefore it is computationally efficient.

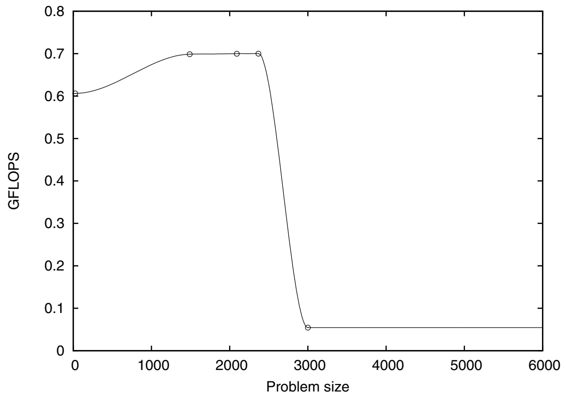

Fig. 6. Akima spline interpolation of a dynamically built functional performance model

When the model consists of less than 5 data points, the Akima splines interpolation can be bzuilt for an extended model that duplicates the values of the left- and rightmost points, s_1, s_k, as follows:

1. $k = 1$: $x_1 = n/p$, $s_1 = s(n/p)$, the extended model specifies the constant speed as $(0, s_1), \left(\frac{x_1}{2}, s_1\right), (x_1, s_1), \left(\frac{n - x_1}{2}, s_1\right), (n, s_1)$.
2. $k = 2$: the extended model is $(0, s_1), (x_1, s_1), (x_2, s_2), \left(\frac{n - x_2}{2}, s_2\right), (n, s_2)$.
3. $k = 3$: the extended model is $(0, s_1), (x_1, s_1), (x_2, s_2), (x_3, s_3), (n, s_3)$.

Proposition 1. The speed functions s_i are defined within the range $0 < x \le n$ and are bounded, continuous, positive, non-zero and have bounded, continuous first derivatives.

Proof. The data points (x_i, s_i) are calculated with $s_i(x) = \frac{x}{t_i(x)}$. As it is a practical requirement that each iteration finishes in a finite time and the Akima splines closely fit the data points with continuous smooth functions we can conclude that s_i is continuous, bounded, positive, non-zero within the range $0 < x \le n$. A feature of Akima splines is that they have continuous first derivatives [5].

3.3 Convergence and Complexity Analysis

Proposition 2. Within the range $0 < x \le n$, the system of nonlinear equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ contains no stationary points and the functions $f_i(\mathbf{x})$ have bounded, continuous first derivates, where $f_i(\mathbf{x})$ is the *i*'th equation of $\mathbf{F}(\mathbf{x})$.

Proof. $\mathbf{F}'(\mathbf{x})$ is non-zero for all $0 < x \le n$, hence $\mathbf{F}(\mathbf{x})$ contains no stationary points. $f_0(\mathbf{x})$ has a constant first derivative. For $f_i(\mathbf{x})$, $1 \le i < n$, if s_i and s_i' are continuous, bounded and if s_i is non zero then $f_i'(\mathbf{x})$ is bounded, continuous. This requirement is satisfied by proposition 1.

Proposition 3. The new data partitioning algorithm presented in this section converges in a finite number of iterations.

Proof. It is proven in [4] that if the range of x is finite, and $\mathbf{F}(\mathbf{x})$ contains no stationary points and if $f_i'(\mathbf{x})$ is bounded continuous then the HYBRJ solver will converge to $|\mathbf{F}(\mathbf{x})| < \varepsilon$, where ε is a small positive number, in a finite number of iterations. These requirements are satisfied by proposition 2.

Proposition 4. The complexity of one iteration of the solver is $O(p^2)$.

Proof. It is show in [6] that the HYBRJ solver has complexity $O(p^2)$. All other steps of the algorithm are of order $O(p)$.

The number of solver iterations depends on the shape of the functions. In practice we found that often 2 iterations are sufficient when the speed functions are very smooth and up to 30 iterations when partitioning in regions of rapidly changing speed functions.

4 Dynamic Load Balancing of Parallel Computational Iterative Routines

In this section, we demonstrate how the new data partitioning algorithm can be used for dynamic load balancing of parallel computational iterative routines on highly heterogeneous platforms.

Iterative routines have the following structure: $x^{k+1} = f(x^k)$, $k = 0,1,\ldots$ with x^0 given, where each x^k is an n-dimensional vector, and f is some function from $\mathbf{R}^n$ into itself [7]. 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 performance of computational iterative routines can be represented by the speed of a single iteration as all iterations perform the same amount of computation.

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 $P_1,...,P_p$ such that $p << n$. Processor P_i contains d_i elements of x^k and f , such that $n = \sum_{i=1}^{p} d_i$.

The 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 [8] or at each iteration [9] and globally redistributed among the processors. Current load measurements are used for prediction of future performance. When applied to large scientific problems and highly heterogeneous parallel platforms, this strategy may never balance the load, because it uses simplistic models of processors' performance.

Instead of single speed values, we propose more accurate models, namely, partially built functional performance models. At each iteration, we redistribute data with help of the new data partitioning algorithm based on multidimensional root-finding. Our dynamic load balancing can be summarized as follows. At the iteration k of the routine:

1. The data is distributed in accordance with the partition obtained at the previous iteration $\mathbf{d}^k = (d_1^k,...,d_p^k)$. For $k = 0$, the data is distributed evenly: $\mathbf{d}^0 = (n/p,...,n/p)$.
2. The computation is executed and its performance is evaluated at all processors $s_1^k,...,s_p^k$.
3. The new observation points $\left(d_1^k, s_1^k\right),...,\left(d_p^k, s_p^k\right)$ are added to the partial performance models of processors and ***approximations of the speed functions*** $\overline{s}_1(x),...,\overline{s}_p(x)$ are recalculated.
4. ***Data partitioning algorithm*** is applied to the current approximations of the speed functions and returns the refined partition $\mathbf{d}^{k+1}$ for the next iteration.

Since $\overline{s}_i(x) \to s_i(x)$ as $k \to \infty$, $1 \le i \le p$, this procedure adaptively converges to the optimal data distribution $\mathbf{d}^k \to \mathbf{d}^*$.

This dynamic 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

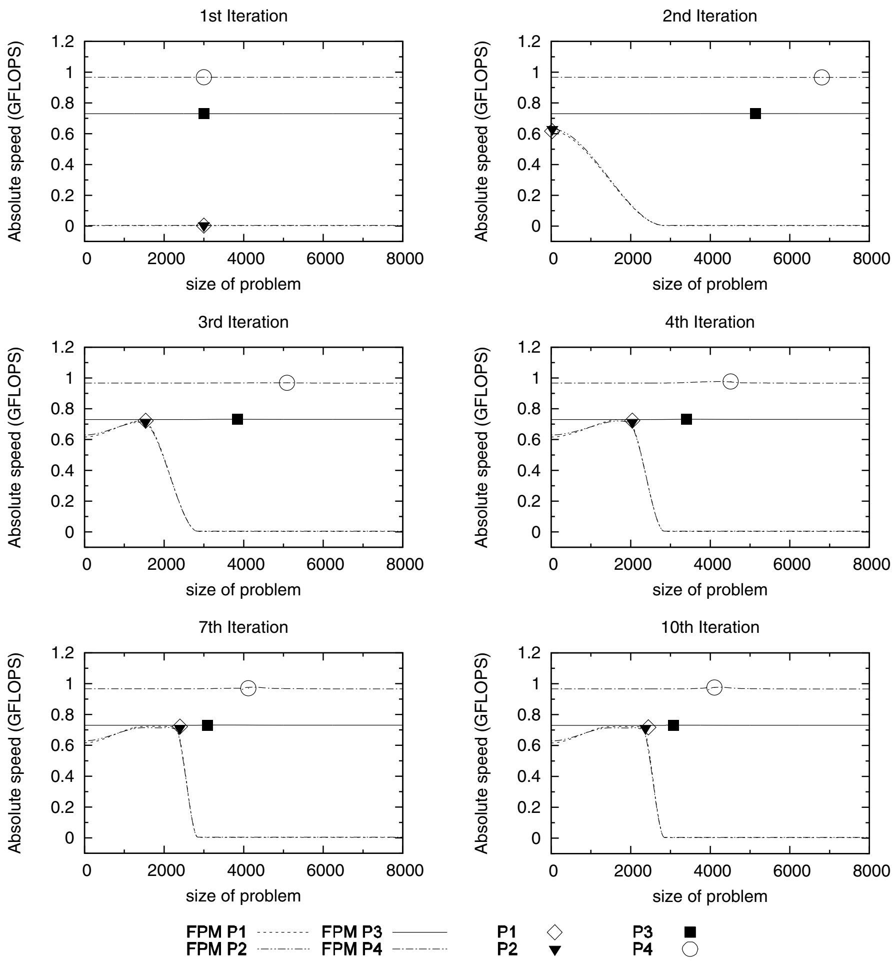

Fig. 7. Dynamic load balancing using multidimensional root-finding partitioning algorithm and the Akima spline interpolation for n=12000

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.

Fig. 7 illustrates the work of this algorithm for the Jacobi method for 4 processors with $n = 12000$. The algorithm converges to the optimal data distribution with each iteration. By the 7th iteration optimum partitioning has been achieved.

Fig. 8 shows the speedup of the CPM and FPM algorithms over a homogeneous distribution. The FPM algorithm used in the experiments is the one based on nonlinear multidimensional root finding. For small problem sizes the speedup is not realised because of the cost involved with data redistribution, however as the size increases both load balancing algorithms improve up to the point were the traditional algorithm based on a constant performance model fails, from which point it performs worse then the homogeneous distribution. The speed up achieved by FPM based load balancing increases as the difference between the relative speeds of the processors increases.

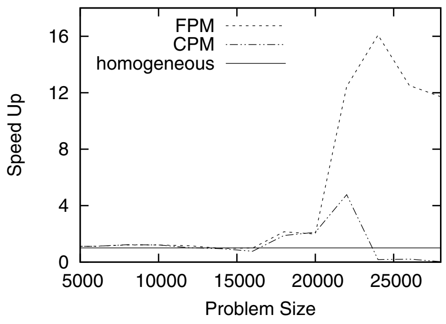

Fig. 8. Speed up of dynamic load balancing algorithms over a homogeneous distribution of n/p using a cluster of 16 heterogeneous machines

5 Conclusion

Functional performance models of processors are successfully applied to balancing the computational load of data-intensive parallel applications on dedicated highly heterogeneous HPC platforms. Based on a history of time measurements, they better reflect different aspects of heterogeneity of processors. Traditionally, the speed of a processor is approximated by a piecewise linear function. The larger the number of the experimental points, the more accurate the speed function is. However, there is a cost associated with obtaining an experimental data point, which requires execution of a computational kernel for a specified problem size. In addition, the straightforward geometrical solution of the data partitioning problem imposes some restrictions on the shape of speed functions. This requires some adjustments of the experimental points and may result to the non-optimal data partitioning because the speed functions become less accurate. In this paper, we proposed a new accurate approximation of speed functions and a new algorithm that employs the numerical solution of the data partitioning problem. We relax the restrictions on the shape of speed functions and numerically solve the system of non-linear equations that formalizes the problem of optimal data partitioning. We have shown that the dynamic load balancing algorithms based on functional models can be used successfully with any problem size and on a wide class of heterogeneous platforms.

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

References

1. Lastovetsky, A., Reddy, R.: Data Partitioning with a Functional Performance Model of Heterogeneous Processors. Int. J. High Perform. Comput. Appl. 21, 76–90 (2007)
2. Lastovetsky, A., Reddy, R., Higgins, R.: Building the Functional Performance Model of a Processor. In: SAC 2006, pp. 746–753. ACM, New York (2006)
3. Lastovetsky, A., Reddy, R.: Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of Their Functional Performance Models. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 91–101. Springer, Heidelberg (2010)
4. Powell, M.J.D.: A Hybrid Method for Nonlinear Equations. In: Gordon, Breach (eds.) Numerical Methods for Nonlinear Algebraic Equations, pp. 87–114 (1970)
5. Akima, H.: A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. J. ACM 17, 589–602 (1970)
6. Powell, M.J.D.: A Fortran Subroutine for Solving Systems on Nonlinear Algebraic Equations. In: Gordon, Breach (eds.) Numerical Methods for Nonlinear Algebraic Equations, pp. 115–161 (1970)
7. 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)
8. Martínez, J.A., Garzón, E.M., Plaza, A., García, I.: Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. J. Supercomput., 1–9 (November 2009)
9. Galindo, I., Almeida, F., Badía-Contelles, J.M.: Dynamic Load Balancing on Dedicated Heterogeneous Systems. In: Lastovetsky, A., Kechadi, T., Dongarra, J. (eds.) EuroPVM/MPI 2008. LNCS, vol. 5205, pp. 64–74. Springer, Heidelberg (2008)