John von Neumann Institute for Computing

Scheduling for Heterogeneous Networks of Computers with Persistent Fluctuation of Load

R. Higgins, A. Lastovetsky

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata
(Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 171-178, 2006.

© 2006 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume33

Scheduling for Heterogeneous Networks of Computers with Persistent Fluctuation of Load

R. Higgins[a*] A. Lastovetsky[a]

[a]School of Computer Science and Informatics, University College Dublin, Dublin 4, Ireland

In this paper we present a model of performance for nodes in a heterogeneous Network of Computers (NOC). Unlike a dedicated cluster a NOC is made up of machines that have varying levels of integration with the rest of a general purpose network. This integration results in different load fluctuations on nodes in the NOC. Our model aims to represent how these routine fluctuations effect performance and we demonstrate the construction of the model and its use in the design and implementation of a parallel applications.

1. Introduction

Networks of Computers (NOCs) provide high performance parallel computing capabilities without the significant investment of acquiring a dedicated cluster. They are typically built from a wide variety of machines existing in a campus, office or some other general purpose network. This heterogeneity between the computers exists at a number of levels as a NOC can be built using any computing resource available.

Performance models for heterogeneous NOCs attempt to represent the differences between the speeds of machines in the network and describe topology of the network interconnect. In this paper we are concentrating on modelling the speed of a non-dedicated machine. Our model describes how the processing speed may vary during the execution of some problem. We present a method of partitioning a data-parallel application which attempts to balance the load on heterogeneous processors in the presence of load fluctuations.

Traditional heterogeneous parallel programming systems estimate performance of individual nodes in the computing network using a single benchmark number. This number is calculated by timing the execution of some critical section of an algorithm[2] or some standard test code[1]. Workload is distributed proportionally according to the benchmarks of the machines involved in the computation. The traditional model shows some weakness on machines that have a high level of integration with the network. A NOC may consist of a number of non-dedicated resources that have some role in the wider network: they may be acting as a network file server, a users personal desktop, mailserver, etc. These machines experience fluctuations in their workload operating in their different roles. The single benchmark model of performance must be built in a short period of time. On a machine operating under a constant fluctuation in load this benchmark may be run during a period of higher or lower than average load. The conditions that an actual computation executes under are quite different to those that a benchmark would. An actual computation executes for a much longer period of time. The fluctuating load that it must contend with will average out over this period to a more steady level. We aim to improve performance of jobs run on a NOC by using a more detailed model of processor performance that represents the varying speed of a machine and by partitioning a computation in a manner that accounts for possible changes in external loads across the NOC.

*This research is funded jointly by the Irish Research Council for Science, Engineering & Technology and IBM's Center for Advanced Studies in Dublin.

We are building upon concepts introduced in [4]. Our model represents the effect of fluctuations in workload on the performance of a machine in a NOC. In place of the single benchmark we use a more detailed performance function as presented in [3]. The performance function describes the execution speed of an application on a machine as the size of the data operated on is increased. We adjust it to create a band of performance that describes a range of possible execution speeds as the problem size increases. Machines that experience large fluctuations in workload will have their performance represented by a wider band than those that experience a more consistent level of workload. The bands are used to find a partitioning of a problem that is optimal for the widest range of load fluctuations across all machines in the network.

The remainder of this paper is organised as follows. In section 2 we describe the procedure to build the band model using load history and a performance function. A algorithm to solve the problem of partitioning with the band model is detailed in section 3. Section 4 presents analysis on the performance increase attainable using the band model. Conclusions and direction for future work are offered in Section 5.

2. Building The Model

The construction of the performance band model requires a number of steps. It is built from two components: a performance function and a pair of load functions. The performance function consists of a number of experimentally obtained benchmarks (execution times) for problems of increasing size. The load functions represent the maximum and minimum average load experienced by a machine over an increasing period of time. For each point in the performance function, using the load functions, we find the maximum and minimum load it might encounter during the problem's time executing. These loads are used to adjust the performance function for worst case performance, contending with a high level of load, and best case performance, contending with a low level of load. In the following subsections we provide more detail on the constituents of the band model, how they are obtained and how they are combined.

2.1. Performance Functions

The first step in the creation of the performance band for a machine is to build a piecewise linear function of optimal execution speed with respect to problem size. The optimal execution speed $s_o(x)$ is the rate at which a problem of size x is solved on an idle processor. The function requires a number of experiments to measure the optimal running time $t_o(x)$ of the application as x increases. $t_o(x)$ can be measured in any UNIX or UNIX-like environment using the **time** utility or the **getrusage()** system call. These functions return statistics maintained by the operating system kernel on the resources used by an application including the time a process spent actively executing on the CPU. This is equivalent to its optimal running time, $t_o(x)$. $s_o(x)$ is equal to the volume of computations for a problem size x divided by $t_o(x)$. We will not consider the construction of $s_o(x)$ any further in this paper, taking it as already provided. Choosing an optimal set of problem sizes with which to measure the execution speed is the subject of a paper pending publication. figure4a.eps

The performance band is built from $s_o(x)$ by adjusting each experimentally obtained point for two scenarios: worst and best predicted performance. Our prediction of performance is based on a history of observed load averages. Using the observations we build two load functions: $l_{max}(t)$ and $l_{min}(t)$. These functions represent the maximum and minimum average load over a period of t time units. We use these functions to find the average loads that would have occurred during the execution of a problem of size x. The maximum and minimum averages are used as factors to adjust $s_o(x)$, resulting in two functions that define the limits of the performance band: $s_{max}(x)$ and $s_{min}(x)$.

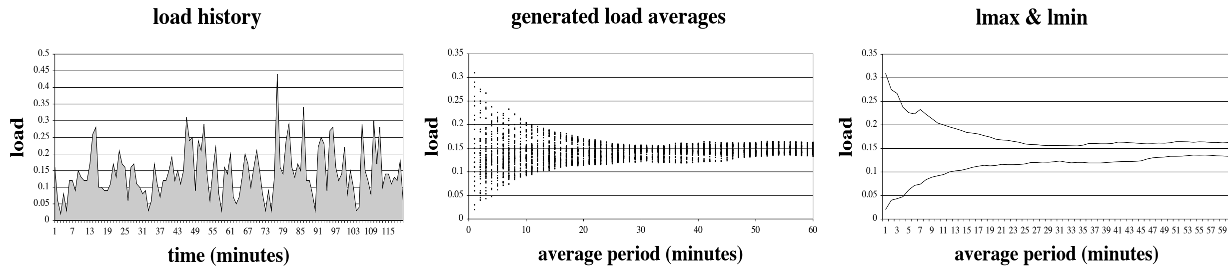

Figure 1. Steps in creation of load functions: first collect a history of load observations, then calculate all load averages of increasing periods, finally extract maximum and minimum piece-wise linear functions: l_{min} and l_{max}.

2.2. Load Functions

The load average on a UNIX or UNIX-like machine is described as: 'the number of processes in the system run queue averaged over various periods of time'.[2] The kernel of the operating system maintains averages with time periods of one, five and fifteen minutes. The averages are exponentially-damped, sampling the run queue length at an OS dependent frequency. They are available through system utilities such as **uptime** and library calls such as **getloadavg()**. A history of load fluctuations is kept by sampling the one minute load average every δ time units. A load average of a time period t can be calculated from the observations by averaging a sequence of $\frac{t}{\delta}$ observations.

$l_{max}(t)$ and $l_{min}(t)$ represent load averages of increasing period t up to some limit w. This limit may be defined as the running time of the largest problem permitted to run on the machine that the functions represent. The size of the history of load averages observations used is defined by h. A sliding window of length w is passed over the h observations and at each position of the window a set of load averages, with periods from δ increasing to w are calculated. A one minute average would be given by the first load observation in the window, the two minute average would be calculated from the average of the first and second observation, and so on. As the window moves over h observations it creates load averages with periods of $\delta, 2 \times \delta, ..., w$ until it begins to slide over the edge of the recorded history, at which point the range of load averages calculated decreases. From these calculated averages a maximum and minimum average for each time period $\delta, 2 \times \delta, ..., w$ are extracted and these values are used to build the piecewise linear functions $l_{max}(t)$ and $l_{min}(t)$.

More formally, if we have a sequence of observed loads $l_1, l_2, ..., l_h$, then a matrix A of calculated load averages is defined as follows:

$$A = \begin{pmatrix} a_{1,1} & . & . & a_{1,h} \\ . & . & . & \\ . & . & & \\ a_{w,1} & & & \end{pmatrix} \text{ where } a_{i,j} = \frac{\sum_{k=j}^{i+j-1} l_k}{i \times \delta} \text{ for all } i = 1...h; j = 1...w \text{ and } i + j \leq h + 1$$

$l_{max}(t)$ and $l_{min}(t)$ are then defined by the maximum and minimum calculated loads from a row t in the matrix A. Points are connected in sequence to give a continuous piecewise linear function. The tth period load averages are given by:

$$l_{max}(t) = \max_{i=1}^{h}(A_{i,t})$$

$$l_{min}(t) = \min_{i=1}^{h}(A_{i,t})$$

[2]From the 'getloadavg' man page entry, section 3 (library calls), Linux Programmers Manual.

The load statistic is maintained by the operating system continuously, there is no additional computation involved our measurement of the load at each δ interval. In comparison to the cost of the initial generation of the performance function, computing the load functions and the resulting performance band is trivial. The benefit of the performance band can be had for almost not cost at all.

2.3. Performance Bands

Given the functions relating to optimum speed of execution: $s_o(x)$ and $t_o(x)$ and the load fluctuation functions $l_{max}(t)$ and $l_{min}(t)$, we now wish to calculate the performance band. An application will execute for a certain period of time, the minimum being $t_o(x)$ when there is no external load on the executing machine. As load increases the executing time also increases. We need to find the points at which the executing time under a given load matches the predicted maximum or minimum load for that time period. The executing time of an application $t_e(x)$ can be calculated by dividing $t_o(x)$ by a measure of the CPU availability. We may estimate CPU availability for a single threaded application based on a load average with the equation (1), where n is the number of processors on a machine. This estimate is not perfect as it's simplicity does not reflect the complex job a kernel does in scheduling processes. In most cases however, the conversion from load average to CPU availability is accurate enough for the purposes of problem partitioning [7].

$$a(l) = \begin{cases} 1 & (l \geq n-1) \\ \frac{n}{1+l} & (l > n-1) \end{cases} \tag{1}$$

Using the availability, we may plot a function of execution time: $t_e(l) = \frac{t_o(x)}{a(l)}$ for a particular problem size x. The points where this line intersects the load fluctuation functions $l_{max}(t)$ and $l_{min}(t)$ correspond to the factors by which we must adjust the optimal speed of execution to create the performance band (Figure 2).

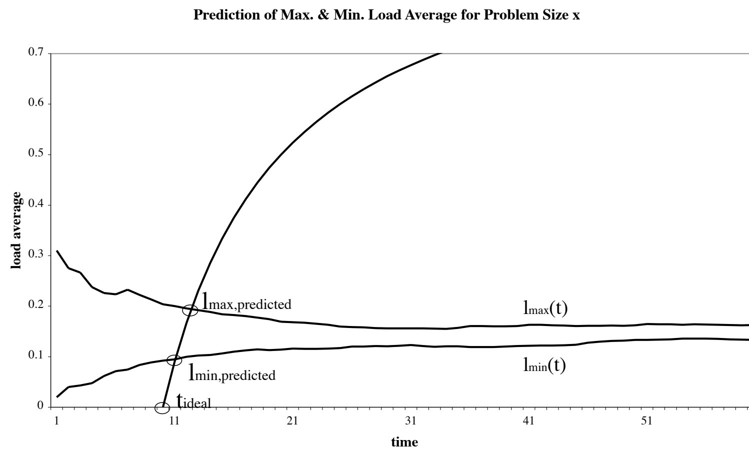

Figure 2. Finding the average load that effects an application over the duration of it's execution.

Using these points, $l_{max,predicted}$ and $l_{min,predicted}$, we adjust the optimal speed to give us the upper and lower limits of the performance band:

$$s_{max}(x) = s_o(x) \times a(l_{min,predicted}) \tag{2}$$
$$s_{min}(x) = s_o(x) \times a(l_{max,predicted}) \tag{3}$$

3. Using The Model

In [3] the partitioning of a problem using a functional performance model was introduced. It was demonstrated that the proportional partitioning of a problem occurred when a line through the origin intersected the performance functions of each machine at points corresponding to their assigned workload (figure 3). Performance bands may be considered as a sets of performance functions occurring between between the maximum and minimum performance levels $s_{max}(x)$ and $s_{min}(x)$. Given a particular distribution of work there may be many combinations performance levels across machines in the NOC that result in a perfectly proportional distribution. Figure 4 illustrates two combinations of performance levels within a band, for which some distribution is optimal. The aim in partitioning a problem with the band model is to find the distribution that maximises the amount of possible combinations of performance where the distribution will remain optimal. These performance levels correspond to levels of workload created on the a machine due to it's non-dedicated status. The distribution allows the maximum amount of fluctuation in load without harming the balance of the distribution.

Calculating the Distribution Arc As can be seen in figure 4 there is an common arc α between which the distribution of work remains proportional. In finding the best possible distribution we wish to maximize the size of this arc. For the two processor example in figure 5, the angle of the arc is determined by lines drawn through the origin to points on the bands. These points given by the work assigned to the respective processors. The arc is always defined by the shallowest line which intersects a maximum speed function, subtracted from the steepest line which intersects a minimum speed function (4).

$$\begin{aligned}\alpha = \min&\left(\arctan\left(\frac{s_{max,0}(w_0)}{w_0}\right), \arctan\left(\frac{s_{max,1}(w_1)}{w_1}\right)\right)\\ -\max&\left(\arctan\left(\frac{s_{min,0}(w_0)}{w_0}\right), \arctan\left(\frac{s_{min,1}(w_1)}{w_1}\right)\right)\end{aligned} \tag{4}$$

Maximising this equation via differential analysis was considered too complex so a heuristic algorithm was used to find the optimal solution. Observation of the problem space has shown no occurrence of local maxima close to the global maximum. Assuming this to be true, a standard hill climbing algorithm [5,6] was chosen to search for this maximum. Experiments with the algorithm have not conflicted with the assumption, though further work would be required to prove it true for all situations. To speed the search for a maximum α the algorithm is guided by initially distributing the workload according to the the average speed of each workstation: a midpoint between $s_{max}(x)$ and $s_{min}(x)$ at an arbitrary value of x. From this point the hill climbing algorithm quickly finds the optimal distribution.

4. Analysis

In our analysis we compare the theoretical execution time of a job partitioned using a single benchmark with a job partitioned using our band model. We suppose that the single benchmark is taken directly before the execution of the parallel job. Such benchmarks are short and subject to load fluctuations. Given the load fluctuation functions of a machine we calculate a set of benchmarks, ranging from ones that may be measured during a high load to a low load. Combinations of these benchmarks are used to partition the job and a running time is calculated. The load combinations in the figures presented are shown on the x and y axes. The speed up gained by using the band model instead of the load-effected benchmarks is shown on the z axis.

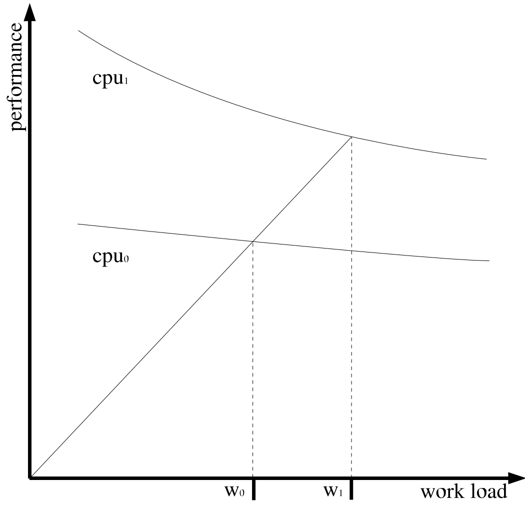

Figure 3. Proportional problem partitioning with performance functions.

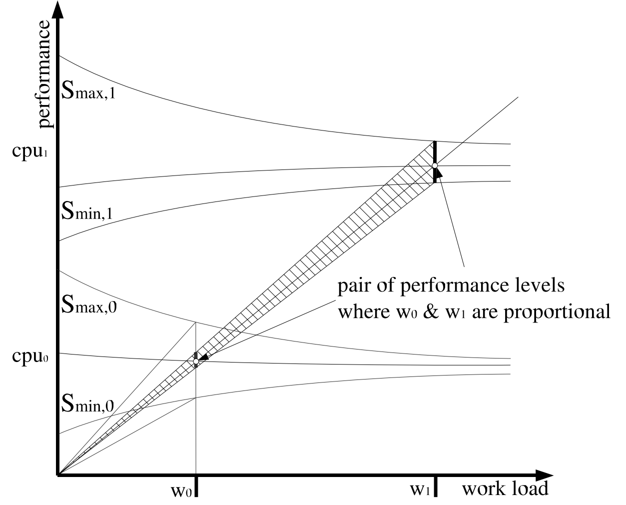

Figure 4. Two pairs of performance levels for which distribution is proportional.

Calculating the Distribution Arc

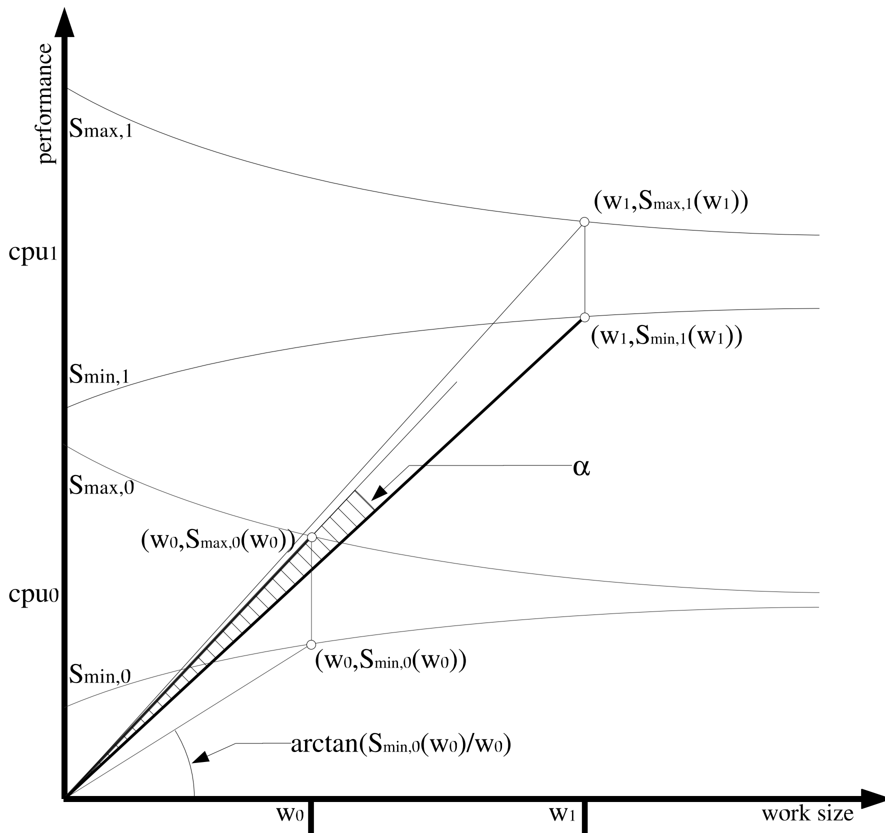

Figure 5. Calculating the size of the common arc α for a data distribution between two processors.

The comparison of the single benchmark and band model is done for three scenarios. A job is partitioned between two machines where: both machines are experiencing: a low level of load fluctuation (figure 6), a high level of load fluctuation (figure 7) and a combination of low and high level of load fluctuation (figure 8). The machines themselves are identical. This homogeneity exists so that we may focus on the effects of heterogeneity in the workloads on the machines. From these plots we can see the conditions under which the band model out performs the benchmark, namely when higher levels of load fluctuation exist on the NOC. We found that for scenarios where both machines have similar levels of load fluctuation, the band model is does not offer very much benefit over a single benchmark. An average 1% speedup was calculated. As the variance between the load fluctuations increases the speed up increases. With two busy machines an average 7% speedup was calculated and 9%, ranging as high as 35%, when an idle machine was paired with a busy machine.

5. Conclusions

In these paper we have presented a method of representing load fluctuation using a performance band. We have demonstrated the construction of the performance band and using load observations and suggested a measure for the optimality of a distribution using this performance band: that of the widest angle α.

Our analysis has shown that a scheduling created by maximizing α outperforms one created using single benchmarks under circumstances where load fluctuations on some of the machines in a NOC are high. When the loads on all machines participating in the computation are relatively steady the performance gains are unclear. In such situations, using the band model only adds complexity to the scheduling. A hybrid model of performance could identify situations where the band is unnecessary and use simpler methods in those circumstances.

From this point there are a number of directions our research may take. The measure of optimality α currently only considers the common arc between all processors. This may be adjusted to also maximise arcs that overlap between a subset of the total number of processors. Doing so would add value to distributions that allow fluctuation in parts of the NOC, but not all. As the number of processors in the NOC is increased, finding a distribution that allows a large load fluctuation on all processors is unrealistic, so this kind of measure may be required.

The load functions $l_{max}(t)$ and $l_{min}(t)$ are built from the extremes of observed loads, however between these extremes there exists a probability curve where load is more likely to occur. Currently the optimality measure gives equal importance to all parts of the band. It is probably more beneficial to impart greater value on distributions that allow fluctuation regions of the band where load is more likely to occur than where it is less likely, at the extremes.

Finally, for more complex measures of goodness the problem space may not suit the simple hill climbing method of optimising the distribution. More elaborate Evolutionary Algorithms should examined and implemented to efficiently maximise goodness factor.

References

[1] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. Technical Report CS-96-328, Knoxville, TN 37996, USA, 1996.

[2] A. Lastovetsky. Adaptive parallel computing on heterogeneous networks with mpc. *Parallel Comput.*, 28(10):1369–1407, 2002.

[3] A. Lastovetsky and R. Reddy. Data partitioning with a realistic performance model of networks of

heterogeneous computers. In *Proceedings. 18th International Parallel and Distributed Processing Symposium*, page 104, 2004.

[4] A. Lastovetsky and J. Twamley. Towards a realistic performance model for netowrks of heterogeneous computers. In *International Symposium on High Performance Computational Science and Engineering*, 2004.

[5] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New Yourk, 1971.

[6] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, New Yourk, second edition, 1991.

[7] R. Wolski, N. T. Spring, and J. Hayes. Predicting the CPU availability of time-shared unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.

Appendix

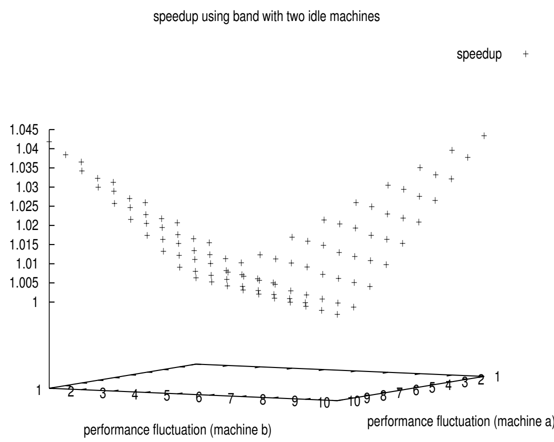

Figure 6. Speed up using two idle processors (1% average).

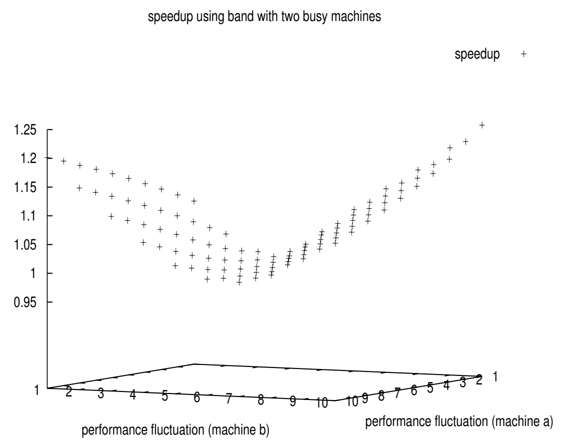

Figure 7. Speed up using two busy processors (7% average)

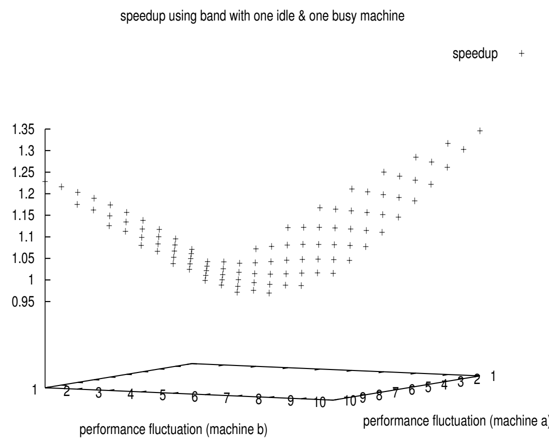

Figure 8. Speed up using one busy and one idle processor (8% average)