The Implementation and Performance Modelling of an Agglomeration Strategy for a Parallel Multigrid Algorithm

Mark G. E. White

EPCC–SS94–15

The Implementation and Performance Modelling of an Agglomeration Strategy for a Parallel Multigrid Algorithm

Mark G. E. White

EPCC–SS94–15

September 1994

Abstract

An agglomeration strategy is implemented in the parallel multigrid application, supplied by the UK Meteorological Office, as part of the Summer Scholarship programme. A performance model is developed to determine how many processes should be used on each grid to minimise the execution time. Measurements are made using up to 16 i860 processors on the Meiko CS-1. Theory is found to agree qualitatively with experiment. Agreement is poor for significant communication times. A general configuration is found that minimises execution times. Inter-processor communications use MPI message passing and the EPCC's PUL-RD regular domain decomposition libraries.

1 Introduction

Multigrid methods hold a wide range of applications in numerical analysis. Their general approach can overcome the limitations of existing techniques. The method of solution to partial differential equations can take advantage of a multigrid. Ordinarily their solution can be computationally expensive, but using the multigrid approach the problem can be reduced to a more manageable size.

The UK's Meteorological Office is interested in the parallelisation of their existing serial multigrid code. The program which implements this is intended only as a research tool. The previous year, the alterations required for a parallel code were completed and a short preliminary investigation was conducted into its effectiveness [5]. This was achieved by a straight forward decomposition of the grids. However, it imposed constraints on the number of grids that could be used. For a large number of processes the coarsest grid would be relatively fine. This can have the effect of increasing the number of cycles that are required.

In research conducted into the parallelisation of multigrid methods, Vandewalle and Piessons compared several strategies to overcome these difficulties [8]. They favoured using a decomposition strategy called agglomeration, which would reduce communication time while allowing many more processors to be used. It is this technique that is implemented and investigated.

The following sections gives a brief account of the multigrid method, applied to partial differential equations. Section 2 illustrates the various agglomeration strategies and builds a theoretical framework for a performance model so that optimization may be achieved. Following this, section 3 discusses the issues involved in writing the code to implement agglomeration. Performance experiments are described which are used to obtain the necessary parameters for optimization. The practical and theoretical aspects are united in the section 4 which details the results of some experiments and uses them to predict the optimum configuration for the multigrid application. Section 5 discusses the issues involved in modelling communications and how they affect the performance model that is derived. In the final section the results are summarised and conclusions are drawn.

2 Theory

2.1 The multigrid method

A physical domain can be discretised using finite differences, such that the solution to a partial differential equation is represented by a grid of points. Let u be the exact solution and v an approximation to the solution. Let A be the discrete differential operator such that

$$A\,u = f \tag{1}$$

This is the equation that must be solved. The error in the approximation, v, is defined as

$$e = u - v \tag{2}$$

Another measure of the accuracy of the approximation v is the quantity called the residual, r, which is defined as

$$r = f - A\,v \tag{3}$$

Two of the most common techniques to solve eqn(1) either do so directly or use an iterative method. For example, Gaussian elimination is a well-known method that produces the solution directly. An iterative method makes an approximation to the solution which it repeatedly improves until the approximate solution is sufficiently accurate. The multigrid algorithm makes use of such an iterative method.

The multigrid approach utilizes a hierarchy of grids that represents the physical domain to different degrees of granularity. The finest grid can be thought of as lying on top, with the coarsest grid placed at the bottom, as in figure 1. The method first produces an approximation, v, on the finest grid using

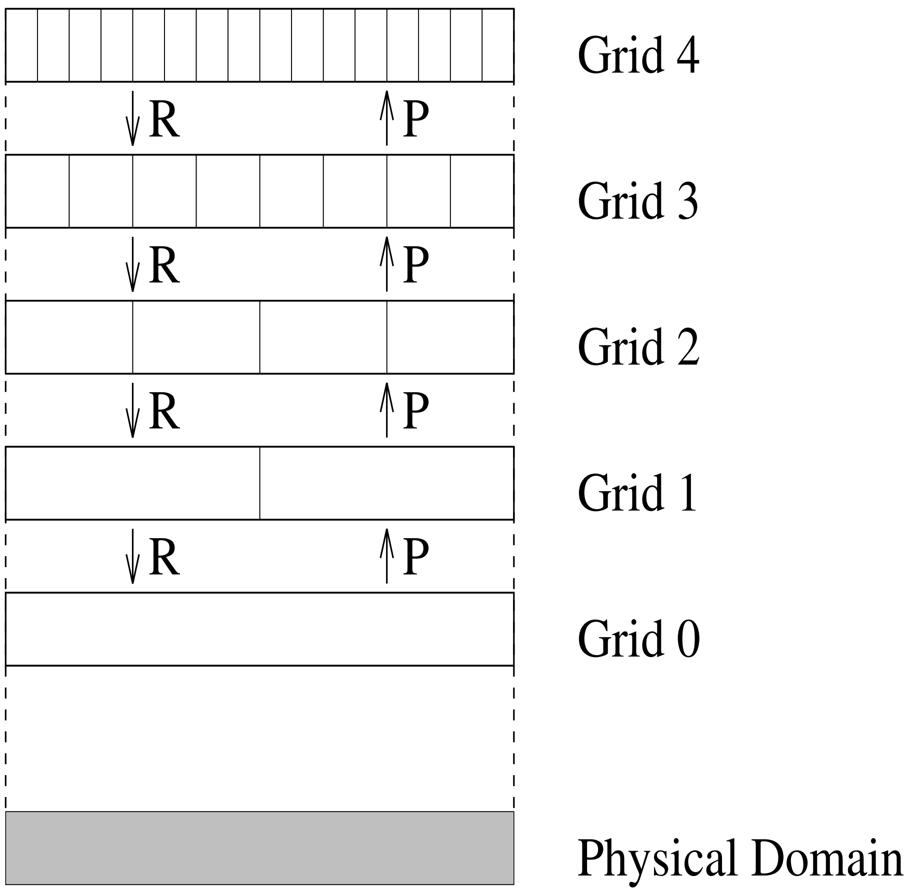


Figure 1: **The hierarchy of grids used in the multigrid method. Every grid represents the physical domain to varying degrees of granularity. The operators** R **and** P **represent the inter-grid transfers called restriction and prolongation respectively. See text.**

an inexpensive smoother. Further smoothing eliminates the high frequency or oscillatory components in the error. The new approximation is then transferred to subsequent coarser grids and this procedure, that forms the descent-phase, is repeated. The reverse process happens in the ascent phase, when these approximations are used to adjust the approximations on the finer grids. The V-cycle, as it is known, is repeated until the approximate solution on the finest grid is sufficiently accurate.

The fine-to-coarse inter-grid transfer is achieved using an operation called restriction. To begin the transfer, the residual is calculated. Using the restriction operator, R, defined by

$$r_{\text{coarse}} = \mathsf{R}\ r_{\text{fine}} \ , \tag{4}$$

the residual grid points are translated onto the coarse grid. The residual equation, defined as

$$\mathsf{A}\ e = r \tag{5}$$

is represented on this grid instead of the original eqn(1). Instead of v, the error, e, is smoothed. The smoothing begins with an initial error of zero. The error, e, is exactly equivalent to the approximation to the solution, v. The residual of the error can be transferred to next coarsest grid and the procedure repeated until finally the coarsest grid is reached. The objective is to obtain a good approximation to the error, e and by using eqn(2) correct the approximation v.

The purpose of the ascent phase is to transfer the error from the coarsest grid back to finest grid. The smoothed error is used to correct all the equivalent approximations to the solutions in the intervening grids, so that the approximation on the finest grid is corrected. The inter-grid transfer that is used is called prolongation; it is the reverse of the restriction operation. The prolongation operator, P, is defined as

$$e_{\text{fine}} = \mathsf{P}\ e_{\text{coarse}} \tag{6}$$

The correction to the approximation on the finer grid is made by

$$\begin{aligned} v_{\text{fine}} &\leftarrow v_{\text{fine}} + \mathsf{P}\ e_{\text{coarse}} \\ &= v_{\text{fine}} + e_{\text{fine}} \end{aligned}$$

This approach of eliminating the error in the solution is called the Coarse-Grid Correction Scheme, which is illustrated in figure 2.

The algorithm used by the Meteorological Office in the multigrid application is more complicated. It is the called the Full Approximation Storage multigrid algorithm. It is more suited to non-linear equations, such as the elliptic equation, defined by eqn(7) in section 3. The procedure outlined here gives the general approach that is used. For a good introduction to multigrid methods, see [1]. For a more detailed discussion, that includes the FAS algorithm, see [7].

2.2 Meteorological Office problem

The serial code supplied by the Meteorological Office solves an elliptic equation of the form

$$\frac{1}{acos\phi}\frac{\partial}{\partial\lambda}\left(\frac{A(\lambda,\phi)}{acos\phi}\frac{\partial Q}{\partial\lambda}\right) + \frac{1}{acos\phi}\frac{\partial}{\partial\phi}\left(\frac{B(\lambda,\phi)cos\phi}{a}\frac{\partial Q}{\partial\phi}\right) - C(\lambda,\phi)Q = R(\lambda,\phi) \tag{7}$$

where the physical domain is the surface of a sphere, namely the Earth. The equation is written in spherical polar coordinates where λ represents the longitude and ϕ the latitude. Eqn(7) can be written concisely as Poisson's equation, $\nabla^2 Q = R$. The coefficients are subject to the following constraints,

$$C > 0 \quad , \quad AB > 0$$

The physical domain is discretised by angular divisions. This causes significant anisotropies near the poles, such that there is strong coupling in the east-west direction. This is exploited by the exclusive use of longitudinal-solvers and is particularly suitable to the parallel decomposition of the global array over the north-south direction. For further details regarding this meteorological problem see [3] and [4]. For a discussion of the issues involved in the decomposition see [5].

The Coarse-Grid Correction Scheme

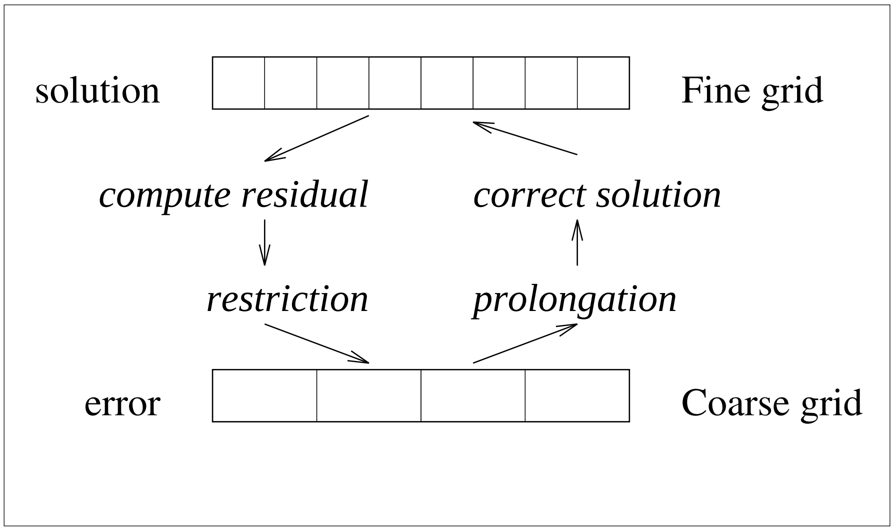


Figure 2: **The coarse grid correction scheme. The diagram represents the inter-grid transfers between two arbitrary grids. The finer grid can be thought of as the approximate solution. The coarse grid holds the residual, which is equivalent to the error. The approximation to the error is improved on this and subsequent grids in the descent phase. On the ascent, the error is used to correct the solution on the finer grids.**

2.3 Agglomeration strategies

The decomposition strategy used by McInnes is referred to in the literature as fixed partitioning with limit coarsening [8]. It has some disadvantages. When considering a large physical domain it is desirable to use as fine a grid as memory and computational power will allow. Generally the more processes that are used on the finest grid the better, since demands on storage space and computation time can be minimized. However, when coarser grids are subsequently used, there is an obvious limit to which coarsening can go. The coarsest grid has one grid point per process.

The first implication of this is that the coarsest grid may be relatively fine. To understand the possible consequences of this, consider the following situation. Once the program completes a V-cycle its tests for convergence on the finest grid. Suppose that the condition is not satisfied. But, if on the other hand, there had been further coarser grids, then perhaps the condition would have been satisfied. A smaller number of grids will require a greater the number of cycles. It is desirable to minimize the number of cycles, when we consider the significant amount of time it takes to complete one cycle.

The second implication concerns inter-processor communication. At every stage boundary swaps[1] are made. There can be noticeable time overheads involved. From this consideration alone it would be better to have less processes so that there would be less communication. A balance must be struck between the need for sharing the computation as widely as possible, and with the need to minimize the communication times. On a fine grid the balance is firmly shifted in favour of using more processes, but on a coarse grid the balance is shifted in the opposite direction in favour of fewer processes. The fixed partitioning strategy is unable to take account of this. It must apply the same number of processes to *every* grid. This

[1] The term boundary swap refers to the communications operation that exchanges data between processes. If the elements in an array are updated according to the their own value and their nearest neighbours' values, then those elements which are on the edge of the local array need the elements of another neighbouring local array to be updated. These elements are sent by the neighbouring processor. At the same time this processor must also send its boundary values to the other, so the other may complete its own update operation as well. Hence the elements at the edge of the local arrays, or boundaries are swapped. Note that some authors refer to the same procedure as a flush operation. [8]

ignores the differing requirements between the fine and the coarse grids.

These difficulties can be overcome by permitting a degree of plasticity in the partitioning of the grid. There are two possible strategies. The first is called fixed partitioning with processor idling, or more concisely, reduced-type agglomeration [8]. This allows there to be fewer grid points than there are processes. Consequently, some processes must become inactive or idle.

Consider the level at which there is one point per process. This is not a realistic situation for the reasons given above, but it will serve to introduce the strategy. In order to move to the next coarsest grid, half the processes must send their data to their neighbours and then become idle. This event is called agglomeration. Their neighbours are now the only processes working on the problem. At subsequent agglomeration stages further data must be sent so that a quarter of all the original processes remain active, then an eighth, then a sixteenth, and so on. Ultimately, this can only continue until there is one grid point and one process. On the ascent phase, data must be returned so that the idle processes can rejoin the computation. There is a substantial amount of message passing required by this strategy.

The previous paragraph suggested that we wait until there is one point per process and before agglomerating. This suffers from excessive communication overheads. But there is no reason not to agglomerate much earlier and thus maintain a larger local array on each process. The problem of which grid levels to agglomerate on must be solved for optimization.

The second strategy is a variation on the first, and is called exchange-type agglomeration. It follows the same pattern as before, but instead of leaving processes idle, they continue to work on a duplicate grid simultaneously with their neighbours. This is achieved by the *two-way* exchange of data at each agglomeration stage, instead of the one-way send and receive. This has the advantage of requiring no data to be exchanged in the ascent phase because each process has all the information it requires already. Moreover, the implementation becomes easier because no consideration need be made to whether a process is active or not. In this sense, exchange-type agglomeration appears favourable.

2.4 A framework for performance

The basis for agglomeration has been outlined qualitatively so far. In order to find an optimal configuration[2] it is necessary to build a theoretical framework. This will permit an exploration of the issues and define in concrete terms the conditions for optimization.

Let us first consider an abstraction of the general procedure involved in a parallel program. The simplest situation to consider is a linear array of N elements decomposed evenly over p processes. This implies that each process holds N/p elements in its local array. Suppose that each element is updated by computing the average of its neighbours and so that the stencil size is three elements. The details of the update function are not important, but they will serve to think about the general situation.

Let τ_u be the update time required to re-compute each element according to the update rule. There will be an overhead time, t_0 which stems from the 'serial' parts of the code which cannot be distributed across the processes. This code must be executed by every process. Therefore, the total computational period is given by

$$t_u = \frac{N}{p}\tau_u + t_0 \qquad (8)$$

Assume that blocking communication is used and that messages are exchanged between nearest neighbours. Further assume that each processor is connected *directly* to its immediate neighbours in a communications network. Communications can then take place simultaneously across these dedicated connections. The time taken to send one message is the same for sending ten messages. This implies that there is no contention across the network.

The transmission time is dependent on the size of the message being sent. The relationship between size and time is assumed to be linear. Hence, for larger messages the communication time is proportionately

[2] The term configuration will refer to the arrangement which specifies the grid levels to agglomerate.

larger. Let s be the message size such that the communication time, t_c, is given by

$$t_c(s) = a + sb \tag{9}$$

where a and b are communication constants. The constant a represents the time to prepare for a message to be sent, which is independent of the size; b represents the rate of increase of the communication time with increasing message size.

The number of messages sent is proportional to the number of processes. Due to the dedicated connections, the transmission time does not depend on the number of messages and is therefore independent of the number of processes. There is one exception, however. Whenever there is more than one process eqn(9) applies but when there is only a single process there is no need for communication and therefore $t_c = 0$. Modifying eqn(9) to take this into account we obtain

$$t_c(p,s) = \begin{cases} a + sb & , \quad p > 1 \\ 0 & , \quad p = 1 \end{cases}$$

For brevity let us define the function θ, such that

$$\theta(p) = \begin{cases} 1 & , \quad p > 1 \\ 0 & , \quad p = 1 \end{cases} \tag{10}$$

so that the equation for communication time can be expressed as

$$t_c(p,s) = \theta(p)\,(a + sb) \tag{11}$$

For blocking communication the times are additive so the total time for one iteration is given by

$$\begin{aligned} t(p) &= t_u + t_c \\ &= \frac{N}{p}\tau_u + t_0 + \theta(p)\,(a + sb) \end{aligned}$$

Consider an idealised multigrid algorithm without agglomeration. The first distinction to be made between this and our abstract model is that there are two dimensions rather than one. In light of the north-south decomposition, the problem can still be considered in terms on one dimension.

Let n be the particular grid number, such that $n = 0$ is the coarsest grid and $n = M$ is the finest grid. The terms in the above equations now become functions of n. Let us re-express the execution time in terms of n:

$$\begin{aligned} t(n,p) &= t_u(n) + t_c(n) \\ &= \frac{N(n)}{p(n)}\tau_u + t_0 + \theta(p)\,(a + s(n)b) \end{aligned} \tag{12}$$

This is a completely general expression for the execution time for one iteration. Expressions for each of the quantities must be determined in terms of n and then substituted into the expression. In its compact form it allows us to consider the general situation without getting involved in complicated expressions.

If we are to incorporate agglomeration into the model then we must draw a distinction between boundary swap messages and agglomeration messages. Boundary swaps are always required during the computation for every iteration. Agglomeration, on the other hand, is not necessarily required for every inter-grid transfer. The relative sizes of these two types of messages differ as well. A boundary swap requires a row of data to be exchanged between processes, while an agglomeration message contains the entire local array maintained by the process. The expression for $s(n)$ will vary but the basic form of eqn(11) remains the same.

Consider the boundary swap message. There are $N(n)/p(n)$ elements in each process's local array. Let $r(n)$ be the number of rows over the entire nth grid. The number of elements in each row, and hence the size of the message, is given by

$$s_{\text{bs}}(n) = \frac{N(n)}{r(n)}$$

The agglomeration message size however, is the total number elements in the local array given by

$$s_{\text{a}}(n) = \frac{N(n)}{p(n)}$$

Hence, the agglomeration message is $r(n)/p(n)$ times larger.

A comparison between the time for a boundary swap and an agglomeration must take in account the number times these communications are required. Let v be the number of boundary swaps for each iteration. The total communication time, t_{bs}, over one iteration is

$$\begin{aligned} t_{\text{bs}}(n,p) &= v\theta(p)\,(a + s_{\text{bs}}b) \\ &= v\theta(p)\left(a + \frac{N(n)}{p(n)}b\right) \end{aligned} \tag{13}$$

The number of agglomeration stages must be considered in terms of the whole cycle. Adopting the same formalism as for communication time, let us define the function γ as

$$\gamma(n) = \begin{cases} 1 & , \quad n = \text{agglomeration stage} \\ 0 & , \quad \text{otherwise} \end{cases} \tag{14}$$

so that the agglomeration time can be expressed as

$$\begin{aligned} t_a(n) &= \gamma(n)\,(a + s_{\text{a}}b) \\ &= \gamma(n)\left(a + \frac{N(n)}{p(n)}b\right) \end{aligned} \tag{15}$$

We must further develop our model by extending it over all grids in the hierarchy. The optimization condition can be expressed in more specific terms: the execution time is minimized over the complete cycle. This implies summing $t(n)$ over n and finding the values of the coefficients to obtain its lowest value. The total time required for half a cycle is given by

$$T_{1/2} = \sum_{n=0}^{M} t(n) = \sum_{n=0}^{M} (t_u + t_c)$$

Using eqn(8) for the update time and combining eqn(13) and eqn(15) for the communication time we have

$$T_{1/2} = \sum_{n=0}^{M} \left\{ \underbrace{\frac{N(n)}{p(n)}\tau_u + t_0}_{\text{computation}} + \underbrace{v\theta(p)\left(a + \frac{N(n)}{r(n)}b\right)}_{\text{boundary swaps}} + \underbrace{\gamma(n)\left(a + \frac{N(n)}{p(n)}b\right)}_{\text{agglomeration}} \right\} \tag{16}$$

To evaluate the summation over n we need to define the functions $N(n)$, $r(n)$ and particularly $p(n)$ explicitly. Recall that the Meteorological Office problem is in two dimensions. The total number of grid points on the nth grid is the product of the number of rows and columns. Given that $r(n)$ is the number of rows and $s_{\text{bs}}(n)$ is equivalent to the number of columns, the total number of points is

$$N(n) = r(n)s_{\text{bs}}(n)$$

For purposes of this model assume that the number of rows is given by

$$r(n) = 2^n \tag{17}$$

For convenience, assume that the grid is square[3] so that $s_{\text{bs}}(n) = r(n)$ and hence

$$N(n) = 2^{2n} = 4^n \tag{18}$$

The function $\gamma(n)$ specifies the configuration for agglomeration. The function giving the number of processors for the nth grid, $p = p(n)$, must be complementary. There are constraints imposed on the values $p(n)$ can hold. The values of the function $p(n)$ must lie within a sensible range, such that $p(n)$ is not less than one and is not more than the maximum number of processes available. An agglomeration stage is signalled by $\gamma(n)$ equalling unity. To reflect this $p(n+1)$ must be double $p(n)$. Therefore, $p(n)$ must equal an integer power of two for integer values of n. This simply means in part that you cannot have a fractional number of processes. If it is not an agglomeration stage however, $\gamma(n) = 0$ and $p(n+1) = p(n)$. For increasing n, $p(n)$ must either remain the same or be double its previous value. These constraints can be expressed more mathematically:

$$\begin{array}{rcl} 1 \leq & p(n) & \leq P \\ p(n+1) & \geq & p(n) \\ \log_2 p(n) & \in & \{1, 2, 3, \ldots\} \\ n & \in & \{0, 1, \ldots, M\} \end{array}$$

where P is the maximum number of processes available.

We have derived the execution time for half a V-cycle. To find the time for the whole cycle eqn(16) must distinguish between the two types of agglomeration. For reduced-type agglomeration the number of messages sent is the same for the ascent and descent phases. Therefore, the execution for one cycle is double that of the half cycle. For exchange-type agglomeration, the descent and ascent phases are distinct, however. In the descent phase agglomeration messages are exchanged, but no de-agglomeration messages are required for the ascent. Compared to exchange-type agglomeration, reduced-type agglomeration requires double the number of agglomeration communication time, when taken over the whole cycle. The computation time and the boundary swap times are the identical so that distinguishing between both strategies, the execution time over the whole cycle is given by

$$T = \begin{cases} 2\sum_{n=0}^{M} \{t_u(n) + t_c(n) + t_a(n)\} & , \text{ reduced-type} \\ 2\sum_{n=0}^{M} \{t_u(n) + t_c(n)\} + \sum_{n=0}^{M} t_a(n) & , \text{ exchange-type} \end{cases} \tag{19}$$

The condition for the optimum configuration is when the execution time is at a minimum. An analytic approach to this problem must first re-structure the summation in eqn(16) so that the $\gamma(n)$ is eliminated. Secondly, a general analytic expression for $p(n)$ must be defined using some configuration-defining parameters such as k and l, say. Once this is done the expression's derivative must be found with respect to the parameters used, equated to zero and solved. The algebra can become extremely involved and once the derivative is found, it is impossible to obtain an analytic expression for the parameters involved. The simplest option is to evaluate the summation numerically on a computer for different configurations, plot the results and determine the minimum.

The theoretical framework has now been constructed. Some measurements that are of interest, with regard to parallel algorithms, can be calculated using this framework. These are the utilization of the processes, the speed-up of parallel implementation, and from this, the efficiency of the configuration.

The agglomeration strategies already discussed require that either some processors become idle on the one hand, or that computation is duplicated across a number of processes on the other. This implies that

[3] In Meteorological Office problem, the number of rows is not quite given by eqn(17), but it is similar. The adjustment which must be made is left until the model in complete. The grid is not square but these assumptions simplify the expressions which follow. For the true number of points per grid see eqn(23) on page 14.

the processing power of the machine is not being fully exploited. The utilization is a measure of the proportion to which the the full computational power is being used. It is simply the ratio of p/P. For the multigrid algorithm it is more convenient to obtain a value for the utilization over the whole cycle.

On fine grids, for example, many processes work on the computation since there are a large number of points. But on a coarse grid few processes will be at work, since there are relatively few points to update. A comparison will show that the fine grid computation will take longer. It is therefore more sensible to attach greater importance to the utilization on the fine grid rather than the coarse grid; the longer the computation the greater this weighting should be. This is embodied in the following expression:

$$\langle U \rangle = \frac{1}{T_{1/2}} \sum_{n=0}^{M} \frac{p(n)}{P} t(n) \tag{20}$$

where the denominator is a normalising factor, so that $\langle U \rangle = 1$ means 100 per cent utilization. It is defined in terms of a half-cycle because the expression is simpler.

The 'speed-up' expresses how much faster the parallel implementation is over the serial implementation. It is the ratio of the serial execution time to the corresponding parallel execution time. For the nth grid the speed-up is given by

$$\sigma(n,p) = \frac{t_u(n,1) + t_0}{t_u(n,p) + t_c(n) + t_0}$$

Note that the update time in the numerator is for one processor, which gives the serial computation time. Using the same approach as for utilization, the mean speed-up over the whole cycle, averaged over time, is given by

$$\langle \sigma \rangle = \frac{1}{T_{1/2}} \sum_{n=0}^{M} \sigma(n) t(n) \tag{21}$$

Initially for the small numbers of processes, the speed-up is directly proportional to the number of processes. The efficiency of the implementation is a measure of the departure from this linear relationship. Efficiency is defined for the nth grid as

$$\epsilon(n) = \frac{\sigma(n)}{p(n)}$$

Therefore, the mean efficiency of the cycle, averaged by time, is given by

$$\langle \epsilon \rangle = \frac{1}{T_{1/2}} \sum_{n=0}^{M} \epsilon(n) t(n)$$

Using these measures, a better insight can be gained in to the relative merits of each particular configuration. However, it should be emphasised that the optimum condition is defined in terms of the execution time only. These measures provide additional information only; it is the minimisation of the execution time that is important.

3 Implementation

The parallel version of this multigrid algorithm decomposes the solution to Poisson's equation over a spherical surface. Each process is allocated a fixed range of latitudes in the physical domain. To adopt agglomeration, this fixed partitioning of the domain must become flexible.

There are a small number of differences with the performance model. To accommodate the poles and the equator, the number of points in the latitude direction, ϕ, is an odd number. Each successive finer grid has double the number of rows in the north-south direction, minus one. In this preliminary implementation, there is only one altitude level, making the grids effectively two dimensional. For a more detailed description see [4] and [5].

3.1 Agglomeration

The program takes advantage of the Parallel Utilities Library for Regular Decomposition, known as PUL-RD [2]. These routines handle the arithmetic required to decompose the grids over a specified number of processes. They also offer a boundary swap routine for the exchange of haloes required during the computation. This library uses MPI, the Message Passing Interface [6], that is emerging as the accepted standard for communication libraries on parallel machines. For the agglomeration routines, calls are made to the MPI library to send or receive the necessary data between neighbouring processes.

Each grid requires a separate instance of PUL-RD. This must take into account the number of processes working on that grid. MPI's implementation of communication groups are exploited in the program for this purpose. These are defined in the initialisation.

The default intra-communicator, `MPI_COMM_WORLD`, is used for the finest grid and for each successive grid prior to the first agglomeration stage. The communicator is sub-divided into two new intra-communicators using the routine `MPI_COMM_SPLIT` for agglomeration. If reduced-type agglomeration is being used then one communicator contains all the processes that remain active, while the other contains the idle processes. For exchange-type agglomeration both communicators contain active processes. This notional division highlights the separate but duplicate computations. For each agglomeration stage, the sub-communicators are sub-divided once more. Finally, a tree of communicators is produced.

Some important modifications must be made to the variables which define the configuration and the data decomposition on each grid. This generally involves the extension of single variables to array references, permitting the variation that is required from one grid to the next. For example, the number of processes that was held by one variable, must now become an array holding the number of processes on every grid. This array is directly equivalent to the processor function, $p(n)$. The variables which inform the process which part of the physical domain it is allocated to, such as the north pole, must also become array references.

The partitioning of the solution and other data arrays takes advantage of PUL-RD. A single array is used to store all the data of a particular kind for every grid. The indices specifying the partition are calculated using the storage requirement of each grid. If reduced-type agglomeration is used, the grids for which a process is idle have no storage space allocated.

It is faster for all active processes to share the restriction or prolongation computation. The restricted arrays form the agglomeration messages. In the descent phase the fine grid is restricted by all the active processes. Then active processes receive the restricted grids from their neighbours which become idle. For the ascent phase the procedure is reversed: the coarse grid arrays are first sent in de-agglomeration messages, then the shared task of prolongation begins. This approach, illustrated in figure 3, minimizes the size of the communication messages, by sending the smaller coarse grid data, and maximizes the number of processes working on the inter-grid transfer.

A special grid must be created in place of the coarser grid. In reduced-type agglomeration a process which is idle over a certain number of coarse grids will not require any storage space for them. The processes which become idle needs some extra storage space to restrict their arrays into. Hence, extra space must be allocated for the transfer onto an additional grid. The processes which remain active however, have double the storage space they require for restriction. The agglomeration data received from another process slots into this surplus space[4].

During an inter-grid transfer, a boundary swap operation is required on the coarser grid. The initialisation provides the instances of PUL-RD for the normal boundary swapping that is required on *agglomerated* grids. This is not the right decomposition for the inter-grid transfer, which uses an *un-agglomerated* coarser grid. A special instance of PUL-RD is declared which is used solely for the inter-grid transfer operation.

The division of the communicator group follows a simple rule. The processes which have a even rank number form one sub-communicator group, and the odd rank numbered processes make up the other.

[4] For exchange-type agglomeration, the space is the correct size.

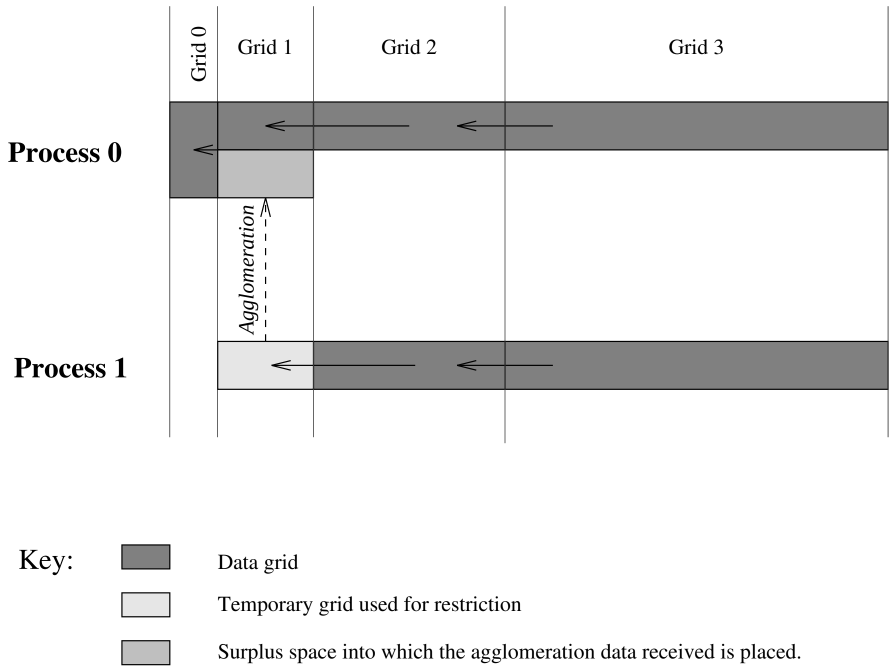


Figure 3: **The partitioning of the decomposed data arrays. There is an agglomeration stage on the grid 1, represented by $\gamma(1) = 1$. Both processes restrict their data from grid 2 into grid 1. Note that during the restriction a boundary swap is required which must use a special instance of PUL-RD. Process 0, which remains active, has extra storage space on grid 1 which is not used in the restriction operation. Process 1, which becomes idle, passes its restricted data to process 0 which places it in the vacant storage space. Process 0 is then free to begin the smoothing on grid 1.**

Each process receives a new rank number for its new communicator group. For each group the numbering always starts from zero, which allows the rule to be applied recursively[5].

The similarities of reduced- and exchange-type agglomeration allows the program to offer both strategies. A switch variable is available to allow the user to choose which particular strategy he or she desires.

3.2 Experiments

The objective, following the implementation of agglomeration, is to find the optimum process configuration. The performance model developed in the previous section sets out to do this. The assumptions which underlie it must the tested to see if they are valid.

The final judgement of any theory is its ability to make accurate quantitative predictions. The time parameters which specify the computation communication times must be measured. The time for a V-cycle can then be calculated theoretically, using eqn(16). This time must then be compared with the actual time measured for the execution of the V-cycle.

Times must be obtained for a number of different configurations. The configuration which holds the shortest execution time is the optimum. Hence the absolute times obtained are not the issue; it is the comparison between them that is important. When the performance model is compared with actual experiments, the correctness of the model is measured by how well it predicts the optimum configuration, not the degree of accuracy of the execution times.

A few extensions must be made to the performance model to accommodate the actual implementation of the multigrid algorithm. The program handles the descent and ascent phases slightly differently. The coarsest grid is also handled separately. The smoothing routine that does the computation is called a certain number of times in each iteration. The parameters `NPRE, NPOST` and `NCOARSE` specify the number of times the smoothing function is called in the descent, ascent and coarsest grid respectively. This can be incorporated in the model by making τ_u an integer multiple of the time for one smoothing operation, τ_{u_0}. This can be generally expressed as

$$\begin{aligned} \tau_u(n) &= c(n)\tau_{u_0}, \\ c(n) &\in \{1, 2, 3, \ldots\} \end{aligned} \tag{22}$$

For example, in the program `NPRE=2`, `NPOST=2` and `NCOARSE=3`. This arrangement is referred to as (2, 3, 2) [3]. The function $c(n)$ for this arrangement is

$$c(n) = \begin{cases} 3 & , \quad n = 0 \\ 2 & , \quad n > 0 \end{cases}$$

To obtain values for the parameters τ_{u_0} and t_0, the individual routines must be timed over a large number of repetitions. Each iteration in the descent phase of the cycle is composed of the follow routines:

- `I_LINE`
- `J_POLE`
- `RESIDUAL`
- `LEFT_HAND_SIDE`
- `RESTRICT`[6]

All the routines' execution times are proportional to the number of points allocated to each processor, except for the routine `J_POLE` which is proportional to the number of points in a row. This much can be gathered by examining the program statements in each routine. An additional term must be added to the

[5] This approach takes no account of some architectural issues that are discussed in section 5.
[6] For the ascent phase, the routine `RESTRICT` would be replaced with `PROLONG`

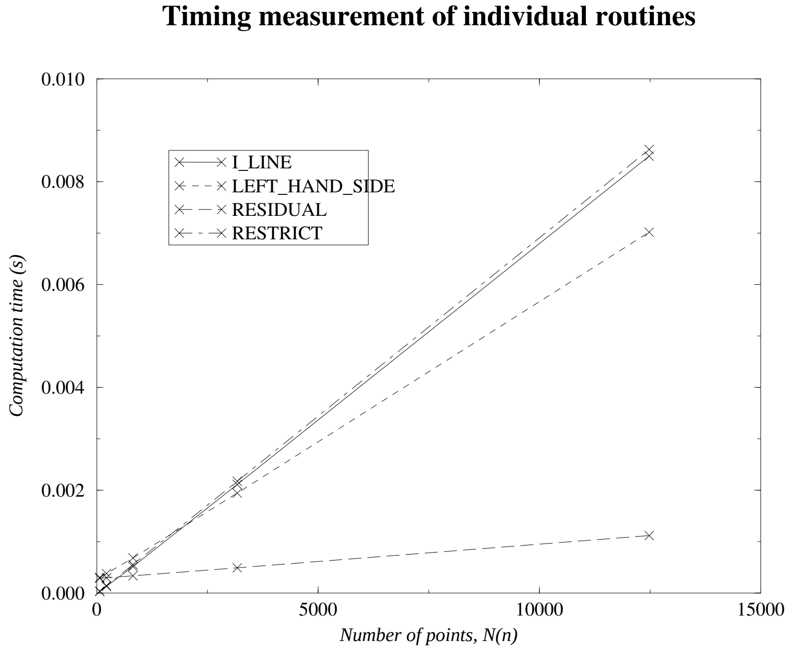


Figure 4: **The timing measurements made for the routines used in the computation. As expected the relationship to the number of points processed is linear. The gradient gives $\tau_{u\,\text{<routine>}}$ and intercept gives $t_{0\,\text{<routine>}}$. The costs of the routines `I_LINE` , `LEFT_HAND_SIDE` , and `RESTRICT` are similar. The routine `RESTRICT` also includes the time to restrict the arrays other than the solution and the right-hand-side. In the next V-cycle the cost to restrict will be much less.**

equation for the update time to account for the computation that is proportional to the number of rows. Therefore eqn(8) becomes

$$t_u = \frac{N}{p}\tau_u + s\tau_{u_{\text{row}}} + t_0$$

where $\tau_{u_{\text{row}}}$ is the computational update time per row element, in the routine `J_POLE`.

The execution times of these routines, *excluding* communications, can be measured for different grids. Plotting the execution time against the number of points processed, shown in figure 4, gives a straight line. The gradient gives $\tau_{u\,\text{<routine>}}$ and the intercept on the time axis gives $t_{0\,\text{<routine>}}$. These values can be combined, taking into account the number of executions of each routine per iteration, to give the overall values for τ_u and t_0.

The number of points on each grid, $N(n)$, is not quite given by eqn(18). To include the equator there must be an odd number of rows. The coarsest grid has 5 rows with 6 columns. The dimensions of the first few grids are

$$\begin{aligned} 6 \times 5 &= 30 \\ 12 \times 9 &= 108 \\ 24 \times 17 &= 408 \\ 48 \times 33 &= 1584 \\ &\vdots \end{aligned}$$

and so on. The general equation for the number of points on each grid is

$$N(n) = (6 \times 2^n)(4 \times 2^n + 1) \tag{23}$$

For communications, the parameters a and b must also be measured. A simple timing experiment can be conducted to determine their values. The assumptions made in the performance model stated that communication was between nearest neighbours. For the experiment two neighbouring processors are chosen. One process sends a message of a certain size to the other. When the other processor receives the message it immediately returns it to the first. The procedure is repeated many times and an average value of the time to execute one send and receive can be found. It is important that only the first process times the operation so that one clock is used.

The time for execution of the whole cycle must be measured as well. The program handles the descent and ascent phases in separate loops. In a similar way the performance model considers only half a cycle. It is therefore convenient to measure the execution time for the descent phase. The average period for execution can then be determined from a large number of repetitions[7].

The time obtained using the extended performance model can be compared with the time measured by running the actual program. The quantitative and qualitatively effects of both can be observed and from this the model may or may not be validated.

4 Results

The values obtained through the previously described experiments are dependent on the parallel machine that is used. The processors' type will have direct effects on the computation period, t_u. The hardware and topology will effect the communication constants a and b[8].

For this set of experiments the machine used is the Meiko CS1. It consists of 16 i860 processors connected in a two dimensional mesh. Each node has sufficient memory to hold data for up to 7 grids. The language used is FORTRAN 77 with some commonly accepted extensions. The arrays are defined to be of the type real, such that one floating point number is stored in four bytes of memory. The optimization option was used in the compilation procedure. The smoother uses the alternating-line zebra method. Full-weighting restriction is used.

4.1 Time parameters

The computation parameters are determined by repeating the appropriate routines one thousand times each, for grids zero to four. The standard time function, `SECNDS`, was used. One processor was used to eliminate communications, and to include the handling of the polar values in one measurement. Using linear regression the gradient and the intercept can be found for each routine. The gradient is equivalent to the computational update time and the intercept gives the additional overhead required each operation. The results are summarised in Table 1.

For a benchmark compare the heavy computation done by `I_LINE` , for example, with the calculation done by `RESIDUAL` which is just a straight-forward matrix subtraction. The overhead requirement is substantial for the routines `LEFT_HAND_SIDE` and `RESIDUAL` . This will prove to be significant for the processing of the coarser grids.

The communication parameters are determined using the method outlined in the previous section. The results are shown in figure 5. As we assumed in the model, the dependency is linear. There is, however, an unexpected transition in the communication time, which further analysis revealed to occur when the message size equals 256 elements or one kilobyte. In practice, most message sizes will be less than this. Therefore, it is appropriate to confine a linear regression to sizes below 256. The results of which are given in Table 2.

[7] For a comparison between the two agglomeration strategies the time for the full cycle must be measured.

[8] The effects due to the architecture are wide reaching. The effects will be seen in the communication time *and* the computation time as well. These issues are discussed in the section 5.

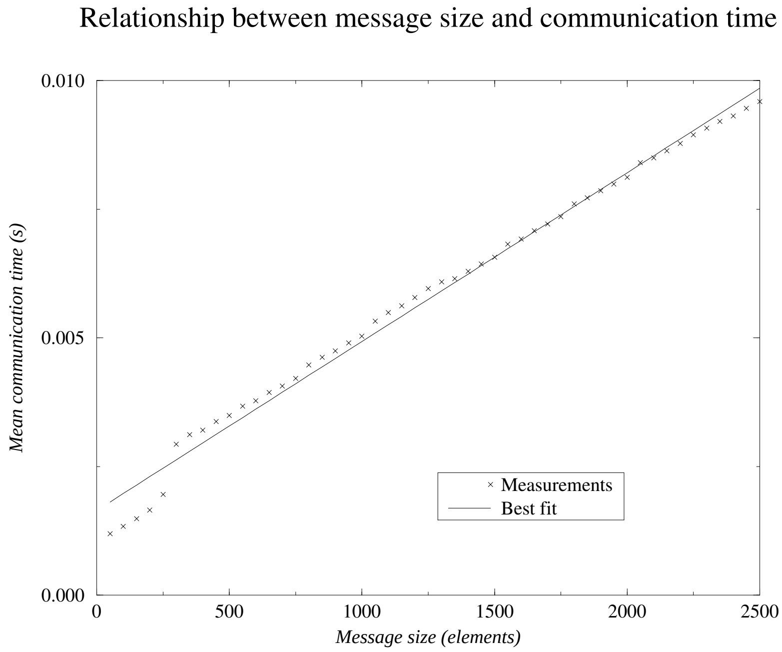


Figure 5: **The approximate linear relationship between the message size and the mean communication time. For a given message size, 1,000 messages are sent and returned in succession. The sizes used begin at 50 elements and increase by steps of 50. The average time is taken for one message to be passed from one processor to the other. In other words, the measured time is divided by two thousand. The best fit straight line gives the constants *a* and *b*, shown in Table 2. (Note the slight discontinuity at *s*=256, equivalent to 1Kb).**

Routine	Description	$\propto$	$\tau_u(\mu s)$	$t_0(\mu s)$
`I_LINE`	Matrix solver	$N(n)$	6.818	-67.4
`J_POLE`	Special routine for handling the polar values	$s(n)$	5.380	0.839
`LEFT_HAND_SIDE`	Computes the vector on the left hand side of the equation. This is used to compute the residual.	$N(n)$	5.407	2720
`RESIDUAL`	Computes the residual according to eqn(3).	$N(n)$	0.670	2830
`RESTRICT`	Restricts a fine grid onto a coarse grid.	$N(n-1)$	6.919	-30.4
overhead	The so-called 'serial' code in the main routine, executed by every process.	n/a	n/a	0.33

Table 1: **The computational update time per point and the overhead incurred for each routine in the descent-phase. The routines were timed for 1,000 iterations to obtain these values. The negative values for the overhead for the routines `I_LINE` and `RESTRICT` are due to the special treatment given to the polar values.**

Constant	value (μs)
a	1649
b	3.281

Table 2: **The communication constants obtained by linear regression from the figure 5.**

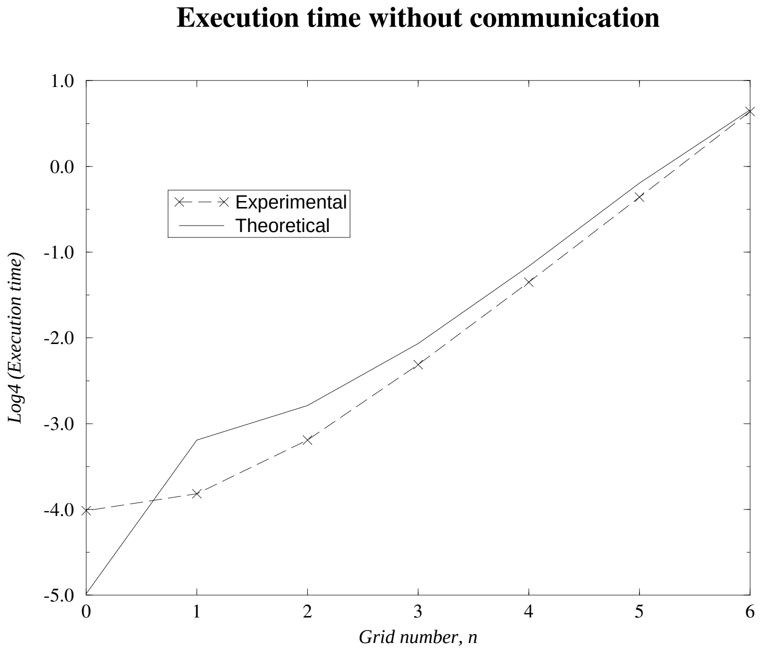


Figure 6: **The execution for each grid for a single processor. This implies no communication overheads. The experimental and theoretical times match closely, except for the coarsest grid. This demonstrates that the computation times calculated from the performance model, are approximately correct. The small disagreement on the coarsest grid has a negligible impact on the overall execution time. A logarithmic scale has been used to bridge the wide range in times. The 4th base was chosen to give theoretical gradient of unity.**

4.2 Various process configurations

With the results obtained above, the execution time for the extended performance model can be evaluated. This allows a comparison to be made between the model and the actual program.

To ensure that the model is correct for the computation only, excluding communications, the times for one processor for different grids are compared. If proper account has been taken in the timing of every routine then the overall times should match up closely. The results are plotted in figure 6. The discrepancies arise due to the different treatment that the polar values receive. For higher grid numbers these become less significant. Satisfied that the model provides the correct computation time, we can proceed to investigate some different process configurations.

The first experiment examines the effect on the execution time for different numbers of processors without agglomeration. The actual time and the performance model time, hereafter referred to as the experimental and theoretical times respectively, are plotted in figure 7. The theoretical curve matches the experimental one quite closely. The assumptions made in the model for communications are mainly responsible for the divergence observed for greater numbers of processors. This is because the communication overheads are more significant, *a priori* the approximations made are worse and the error is more prominent.

The speed-up can be calculated by taking the execution time for p processors and dividing by the

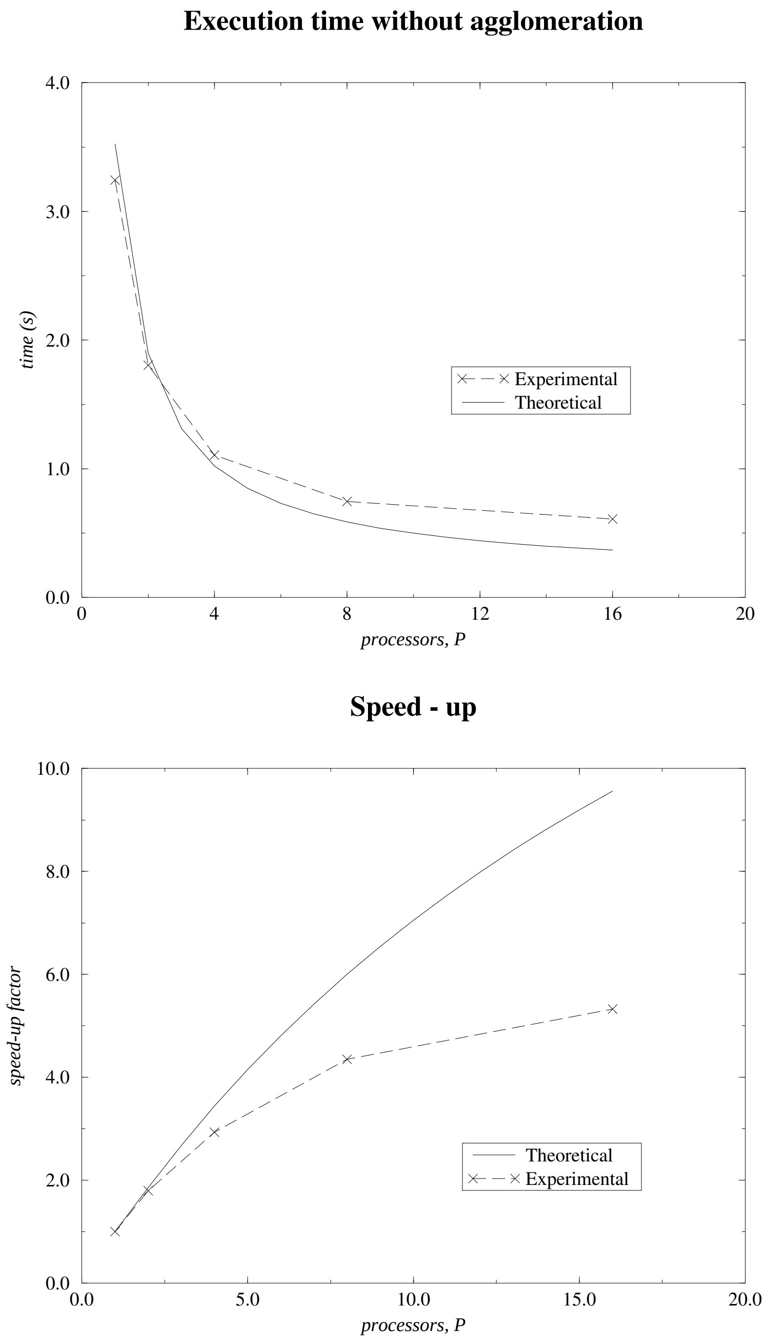


Figure 7: **The execution time and the equivalent speed-up obtained for the descent phase of the cycle over five grids. The number of processes remains constant for each instance. There is a slowed reduction in execution time for higher numbers of processors, which is more clearly seen in the speed-up. The theory gives a reasonable prediction of the execution time.**

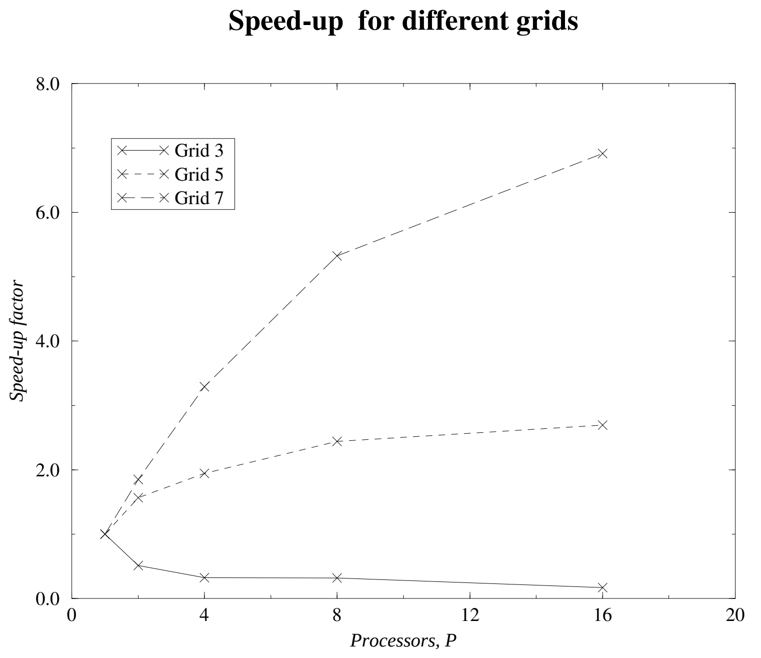


Figure 8: **The speed-up measured experimentally for the third , fifth and seventh grids. The speed-up for the finest grid, namely grid 7, is as one would expect. The curve flattens out quite quickly, due principally to the communications overheads. The coarse grids, 3 and 5, show severe reductions in speed-up because communication times form a major proportion of the execution time.**

execution time for a single processor. The results are also plotted in figure 7. The relationship is not linear, and shows a law of diminished returns. For every further increase in the number of processors, the improvement in speed-up declines. The overhead time responsible for this includes a communication and a computational component. The differences between the theoretical and experimental values are more prominent. It indicates that insufficient communications time has been accounted for in the model[9].

The speed-up for the cycle can be broken-down to show how it varies from grid to grid. The results for three grids are shown in figure 8. For a fine grid we obtain a typical speed-up where the efficiency drops as the number of processes increases. The coarser grids however, show very poor or even *fractional* speed-up. This illustrates the important role that communication plays in the coarser grids. For this reason, it is better to choose a small number of processes for these grids.

The following experiments are intended to investigate different configurations for reduced-type agglomeration. Consider the a single agglomeration stage which may be placed at different grids. The configuration function with its complementary general processor function is shown in figure 9. They are defined as

$$
\begin{aligned}
\gamma(n) &= \begin{cases} 1 & , \quad n = k \\ 0 & , \quad n \neq k \end{cases} \\
p(n) &= \begin{cases} 2^l & , \quad n \leq k \\ 2^{l+1} & , \quad n \geq k \end{cases} \\
k &\in \{0, 1, 2, \ldots, M\} \\
l &\in \{0, 1, 2, \ldots, \log_2(P-1)\}
\end{aligned}
$$

The experimental and theoretical results, plotted in figure 10, show broad agreement. The experimental values show that for every instance of l there is an optimum configuration k. The disparities in the

[9] The small errors appear larger in the speed-up graph because the error term appears in the denominator.

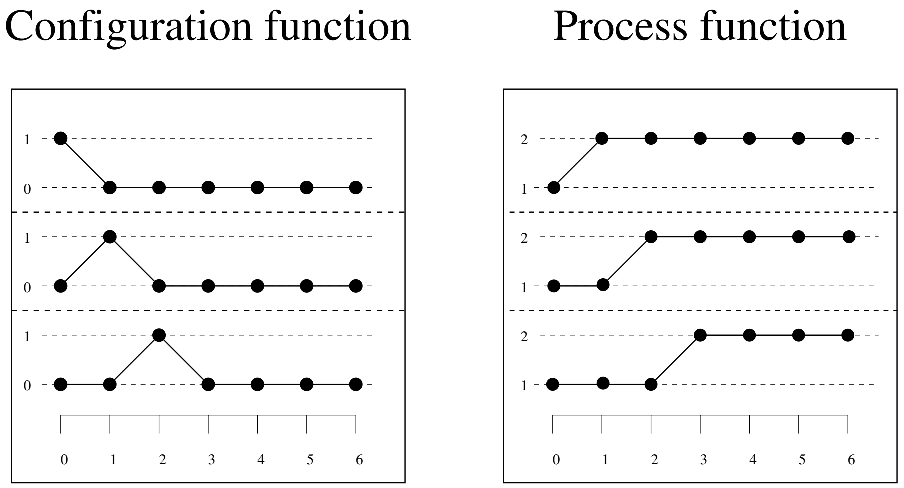


Figure 9: **The arrangement for the second experiment, called the 'step' configuration. The configuration function and the complementary process function are shown for three instances: *k* = 0, *k* = 1 and *k* = 2. The name comes from the shape of the process function. Note that these are not continuous functions. Only the integer values, given by small circles, are defined.**

theoretical values render the slight differences which gave the experimental minimum negligible. The performance model is not good enough to show fine qualitative behaviour of the actual program. These results provide some support to the view that it is better to agglomerate earlier than is necessary, but the reduction in the execution is marginal.

The general behaviour of the graph can be understood as follows. The coarse grids have few points to be processed. If many processors are working on these grids, the work load will be shared rather thinly. In such circumstances the communications become significant. The messages will be small making the set up time a very important. The contribution made by the linear relationship described by b is small. As the grids grow finer, the required computation rises dramatically. When the gains from distributing the work load over more processes out weighes the price demanded by communication, then the grid should be agglomerated.

There may be a better approach to minimize the execution time. Since the finest grid has the greatest number of points, it will take the longest time to process. Therefore, all processes should be assigned to the task. For the coarsest grid the previous experiment showed that it was better to have as few processes as possible. It follows from these facts that the configuration that yields the shortest time is the one that has the maximum number of processes allocated to the finest grid, while the coarsest grid is allocated a single process. The questions that must be addressed are: at which point should agglomeration begin?; and after how many grids should there be between each stage? As we are considering a small number of grids, there is little scope for varying how frequently to agglomerate. The configuration and process functions for this arrangement are shown in figure 11 and are defined as

$$
\begin{aligned}
\gamma(n) &= \begin{cases} 1 & , \quad k \le n \le k' \\ 0 & , \quad \text{otherwise} \end{cases} \\
p(n) &= \begin{cases} 1 & , \quad n \le k \\ 2^{n-k+1} & , \quad k \le n \le k' \\ P & , \quad n \ge k' \end{cases} \\
k &\in \{0, 1, 2, \ldots, M\} \\
k' &\in \{k, k+1, \ldots, \min(\, M, \log_2 P + k - 1)\}
\end{aligned}
$$

The results are shown in figure 12. Again we see that leaving agglomeration till the last moment, when there is very nearly one point per process, does not give the optimum configuration. It shows that holding

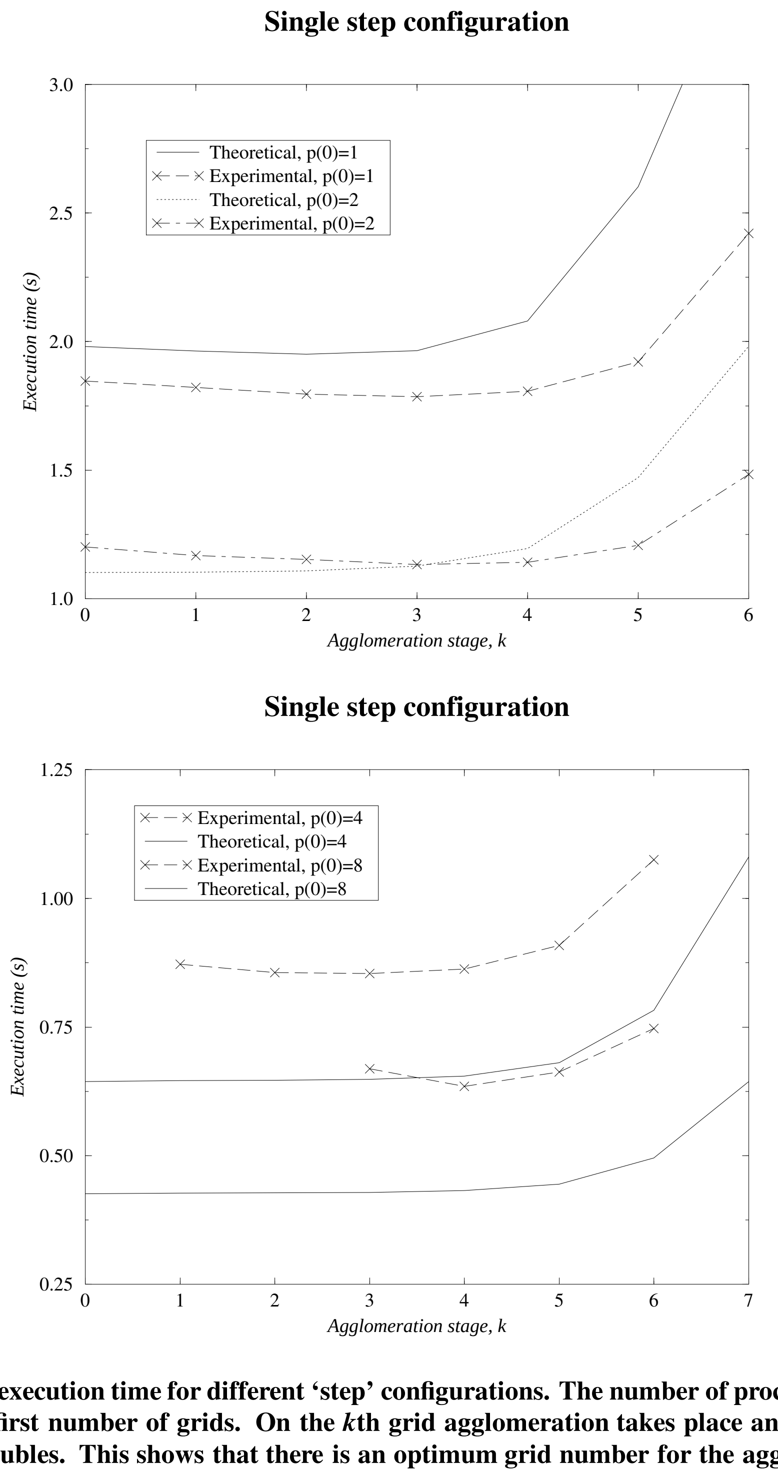


Figure 10: **The execution time for different 'step' configurations. The number of processes remains fixed over the first number of grids. On the *k*th grid agglomeration takes place and the number of processes doubles. This shows that there is an optimum grid number for the agglomeration to take place. The theoretical times agree broadly with the experimental times, but they do not show a minimum.**

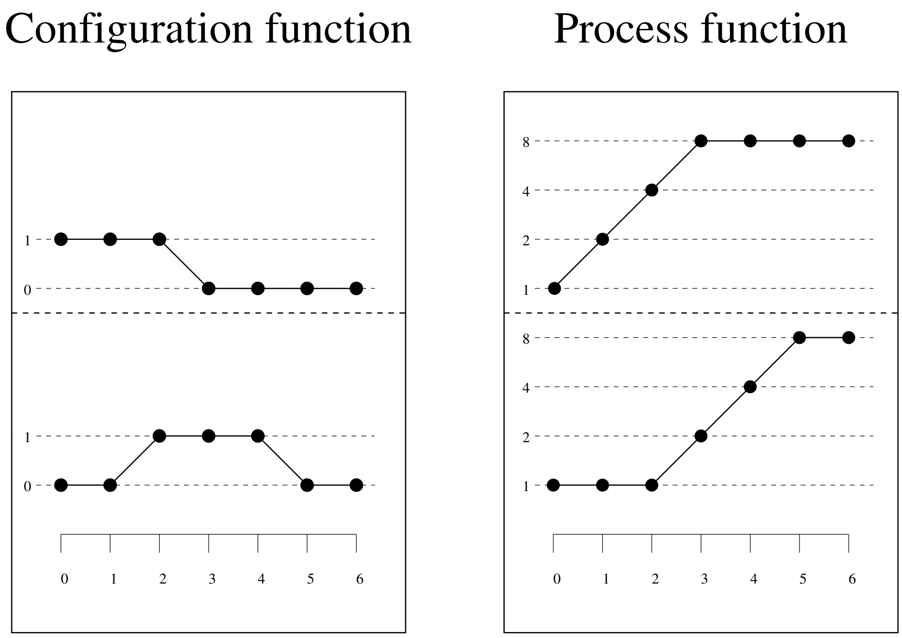


Figure 11: **The arrangement for the third experiment, called the 'staircase' configuration. The configuration function and the complementary processor function are shown for two instances: $k = 0$ and $k = 2$. This is an extension of the 'step' function, hence the name. Note that these are not continuous functions. Only the integer values, shown by small circles, are defined.**

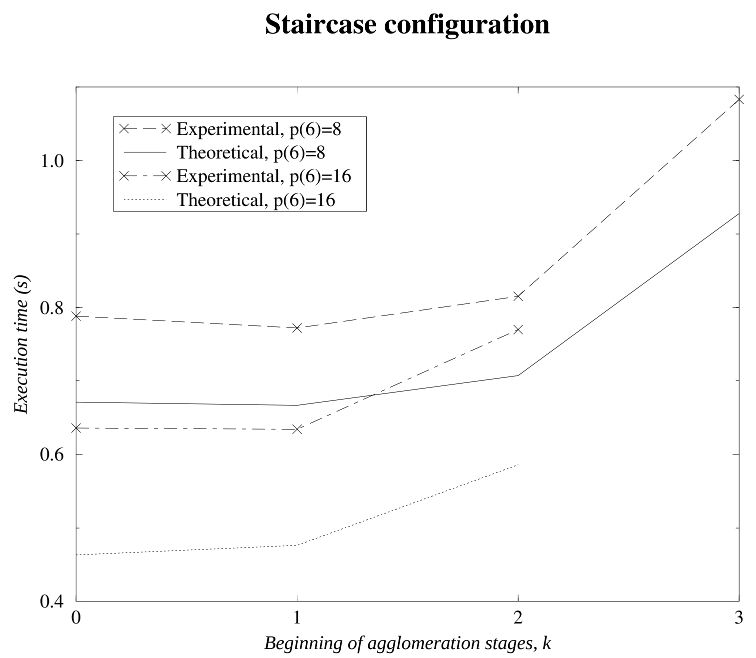


Figure 12: **The execution time required for the descent phase of the cycle on a 'staircase' configuration. The coarsest grid has one process, the finest has 8 in first case and 16 in the second. The horizontal axis gives the grid number at which the staircase begins. The theoretical value is less exact for the higher numbers of processes.**

Execution time over a staircase configuration

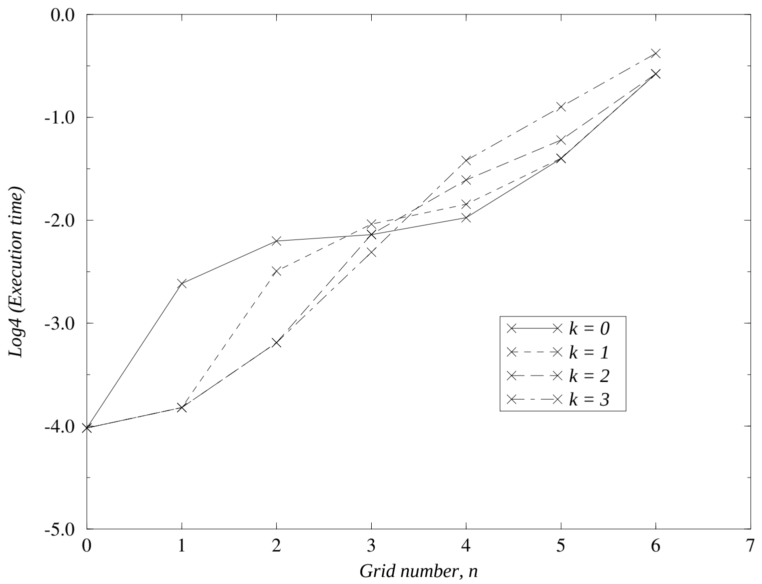


Figure 13: **The execution time measured experimentally for individual grids in a 1-to-8 staircase configuration. Around the grid, *n* = 3, there is little reduction in the execution time from one grid to the next. The agglomeration stages are revealed by the sharper reductions in execution time. (A logarithmic scale is used to bridge the large range of times. The 4th base was chosen because without the effects of communication, the time would drop by one unit for each grid.)**

all points on a single process for the first few coarser grids yields the shortest execution time.

A greater insight can be gained by examining the execution time required for each grid. The results for one staircase configuration are shown in figure 13. The graph indicates that the demands on computation and communication are balanced for a particular grid. The $k = 0$ configuration demonstrates the overhead cost incurred in delaying agglomeration until the last opportunity.

The graphs for speed-up and efficiency can be derived from the execution times measured for the individual grids. The results, in figures 14 and 15 reveal the extent to which communications dominate the processing on the coarser grids. Only when the grids become quite large do we obtain an acceptable values for speed-up and efficiency. The dramatically low values, particularly for efficiency, are striking.

All the results can be taken together to understand the reasons for the optimum configuration. For every increment in the parameter k, which specifies the staircase configuration, there are two implications. On the coarser grids the significant gain from the elimination of communications must be offset by the reduction in the computational power. On the finer grids the situation is similar, but not in degree. The reduction in communications is less noticeable but the gain in processing power is considerable. The minimum occurs when the overall reduction of communications equals the overall gain in processing power. This was revealed in the execution time and speed-up for grid $n = 3$.

Three different configuration strategies have been discussed: no-agglomeration, the single step configuration, and the staircase configuration. Using the best arrangement for each, their execution times can be compared, as shown in figure 16. The execution time without agglomeration does well in this situation. The staircase configuration strategy comes out with the longest execution time because it requires additional time for agglomeration communications.

The slow execution times on the coarse grids are more than offset by absence of agglomeration communications. In order to make a comparison the times were not taken to the coarsest grids. Therefore,

Speed-up over a staircase configuration

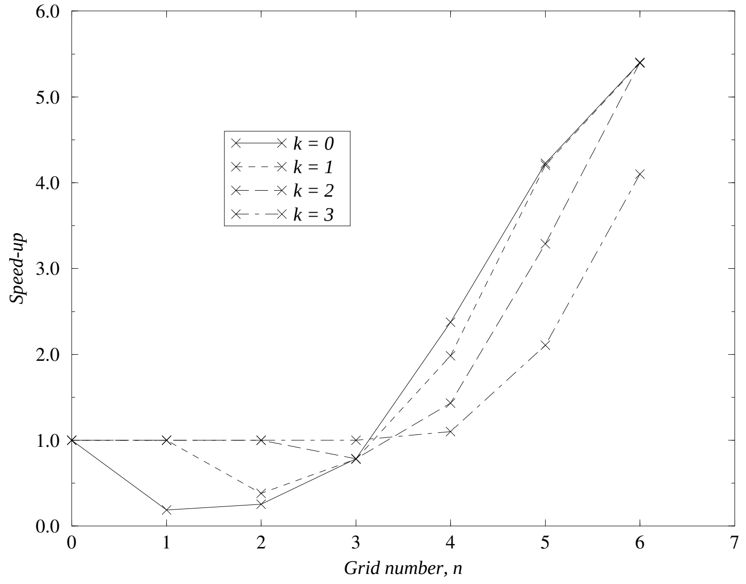


Figure 14: **The speed-up measured experimentally for individual grids in a 1-to-8 staircase configuration. A speed-up of unity indicates that only one process is active. The poor configurations are those which show a speed-up of less than one. This reveals that communications are a very important factor for coarse grids.**

Efficiency over a staircase configuration

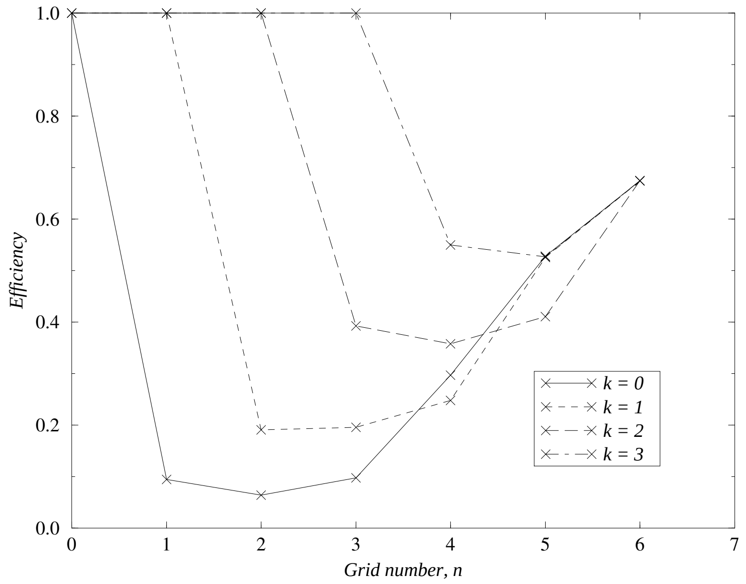


Figure 15: **The efficiency measured experimentally for individual grids in a 1-to-8 staircase configuration. The values are calculated from figure 14. The unitary values for efficiency show to what extent one processor was used. The sudden drop in efficiency reveals the extra demands made by communications. For *k* = 0 the efficiency can be very small.**

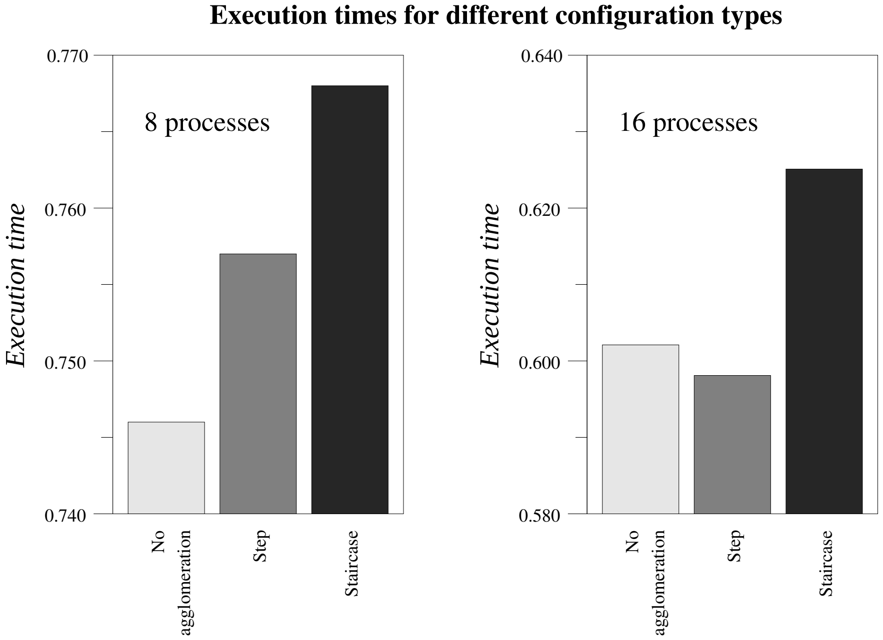


Figure 16: **A comparison of the different types of configuration functions. The execution time is taken over the grids n = 6 to n = 1 for the set of 8 processes, and over grids n = 6 to n = 2 for the set of 16 processes. The fastest configurations are chosen for the step and staircase configurations. For the step configuration 4-to-8 k = 3 and for 8-to-16 k = 4. For the staircase configuration 1-to-8 k = 1 and for 1-to-16 k = 2. The single step configuration does well when compared with the no-agglomeration configuration for 16 processes, but does poorly when compared for 8 processes. In both cases the no-agglomeration strategy does better than the staircase configuration strategy. The differences between the times for the different configurations types are exaggerated by the vertical scale used.**

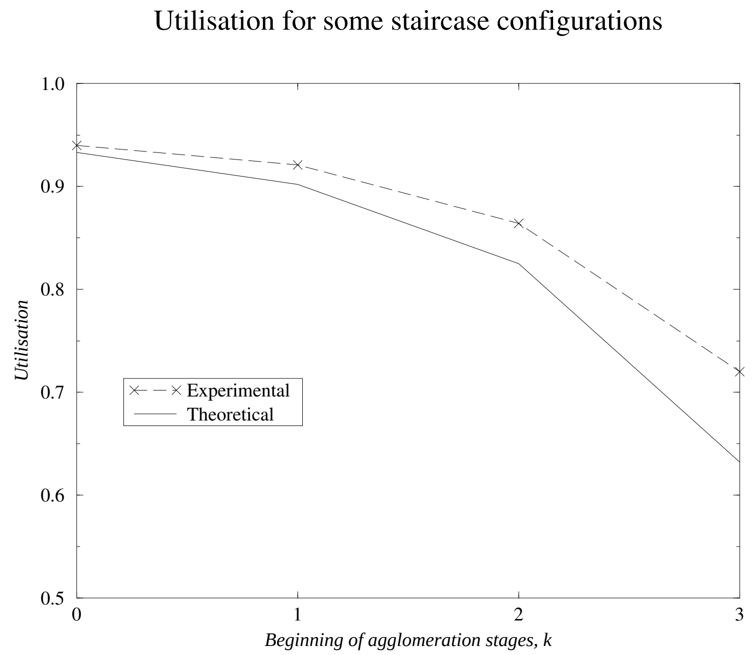


Figure 17: **The time averaged utilization over the descent phase of some 1-to-8 staircase configurations. The utilization is high, in spite of the fact that one process is used on the coarser grids, because the greatest proportion of processing time in spent on the finer grids, where most processes are active.**

the favourable times shown mask to inadequacy of this strategy, since the advantages of the other configuration strategies on the coarsest grids are not taken into account.

In more realistic situations there will be larger finer grids requiring as many processes work as possible work on these grids. Without agglomerating, this requirement restricts the number of coarse grids that can be used. This also reduces the convergence rate considerably. Therefore, more cycles will have to executed, making a significant impact on the total execution time required to converge the approximate solution to a certain tolerance.

The staircase configuration strategy provides the best of both worlds: for the coarse grids it gives the shortest execution, when one process is used; for the fine grids, it allows the maximum number of processes to be used, minimising the execution time. Therefore, it can use the maximum number of grids, so that the convergence is not hindered. It provides the shortest time for convergence of the solution, since it requires less cycles to do so.

The important issue of utilization can also be addressed. It may be of concern to some readers that the whole approach of agglomeration is flawed since it 'wastes' valuable processing power. It is undoubtably true that the staircase configuration does lay emphasis on a single process for the coarser grids, leaving all fifteen others idle. However, by doing so that execution time is minimized. It is surely the overall processing time for the whole program that is important, because processing is allocated on the basis of time. It is beside the point if some processes must lie idle during this time. Nevertheless, some may be unsatisfied by this, after all it does appear to be inefficient. To lay these concerns to rest, recall that the greatest proportion of time spent is on the finest grids when most or all processes are working. The time averaged utilization can be determined over the whole cycle using eqn(20), the results of which are plotted in figure 17. We see that for these staircase configurations that the utilization is mostly above 80 per cent. This demonstrates that the agglomerating to one process does not have a noticeable impact, in terms of utilization, but does have the effect of reducing the overall execution time.

5 Discussion

The comparisons made between the actual times measured and the predicted time calculated using the performance model showed that when communications become significant agreement is poor. This is not surprising, considering the simplifying assumptions that were made in the model's derivation. There are a large number of issues which have been ignored. It is important that they are addressed.

The principal assumption made was that communications would be between immediate neighbours. For the initial arrangement on the finest grid, this assumption probably holds true. But following the agglomeration stages it is unlikely to be the case.

The choice between which processes continue to participate in the computation, and those which become idle, has important implications. A rule is chosen which defines the sub-division of the intra-communicator groups. At the heart of this lies the relative communication times which apply to different implementations. Under certain arrangements, boundary swap messages can be between nearest neighbours, while agglomeration messages are not. Alternatively for another rule, the opposite may be the case, or even a mixture of both. With the objective of optimization in mind the communication time must be minimized. The underlying architecture of the parallel machine exerts an important influence in this regard.

If a message is not sent to an immediate neighbour, it will have to be routed through the network. This opens up a Pandora's box of problems, complications, and many uncertainties . It *cannot* be represented in the simple way given in the performance model. The assumptions made in its derivation required that *all* communications be between nearest neighbours; that this should be through direct dedicated connections; and that there should be no contention. We must consider communications, other than those between nearest neighbours, which implies rejecting the simplifying assumptions that were made.

The way in which *processes* are mapped on to *processors* is not certain. MPI uses a rank number to refer to different processes. Each process has its own rank number in a given communicator group. If process zero sends a message to process one, it does *not* follow that since the ranks differ by one, this implies nearest neighbour communication. It may be the case in fact, but it cannot be assumed.

The connectivity of the network has important implications as well. The more processors a given processor is directly connected to, then the greater likelihood that the message will be passed to a neighbour. Moreover, a larger number of different routes are available for a message to follow if it is not sent to a neighbouring process. A six dimensional hypercube, for example, has its has six nearest neighbours. Other architectures may only have four nearest neighbours per processor.

Suppose a message a processor sends is not addressed to one of its immediate neighbours. The message must pass from one node to the next, until it reaches its destination. Under certain architectures, some processors must pause in their computation to deal with such a message. If the message is not addressed to itself then it must send the message on to the next node in the routing chain. This has effects on the time to complete the computation for every processor, as almost certainly each processor will have to stop at various different times to pass on different messages.

If more than one message is required to be sent along the same network line, there is contention. Various different routing algorithms exist which seek to alleviate these problems as much as possible. When contention arises messages can be sent along other routes to reach the same destination. Consider two messages which are sent by one processor to the same destination at the same instant. Through the vagaries of communication, the message which passes along more nodes than its partner may reach the same destination *sooner* than the other, which is passed along the direct route!

To begin to attempt to accommodate some of these issues in a model for communications would be difficult, if not impossible, and perhaps even foolhardy. In many cases, experiments must be conducted using the actual architecture, without the use of any abstract model. These experiments involve many repetitions of the same communications in order to obtain good time averages. In fact, such data yield only the abstract concepts of mean, standard deviation and other such measures These statistical measures are not part of the intrinsic nature of the system; its nature is its *variability*. It is all very well to know the mean transmission time, but it says nothing about the transmission time for next message.

With such grave difficulties, what justification is there for using such a simple representation for communication time in the performance model? The results which have been obtained have shown that the theoretical times predicted by the model do agree to certain extent with the experimental ones. The qualitative behaviour is the broadly the same. Where the experimental results show a shallow minimum, however, the theoretical results do not. There is not the high accuracy required in the model which would reveal such a minimum.

It follows from these observations that the many factors which influence communications do not exert very significant effects. Some brief experiments have shown that the greatest factor affecting the communication time is the messages size. The standard deviation in the communication time, under different conditions, amounted to little more than 10 per cent. In comparison, the communication time required for a similar message on one grid would be approximately double the that of the same message on the following coarser grid. For this reason alone, the simplistic assumptions made about communications can be justified.

6 Conclusions

The strategy called agglomeration has been successfully implemented. New routines for agglomeration have been added to the existing parallel version of the Meteorological Office's code. The initialisation routine to set up the instances of PUL-RD, have been extensively modified to accommodate the inclusion of the communicator group hierarchy. Further small modifications where required throughout the code to handle the changing process assignments to different parts of the physical domain. A switch variable was added to allow the user to choose the particular kind of agglomeration.

The theoretical framework which was developed proved to be a reasonable predictor of the execution time. For higher numbers of processes, however, the theory did not agree very well with experiment. This is primarily due to the greater importance communications have on the execution time. The disparity was highlighted particularly in the speed-up graphs, which showed that insufficient account was taken of the communications. Nevertheless, as a predictor of execution time the model can prove useful for situations where the time for execution is unknown for very large grids. Time can be saved by finding the best configuration in advance, before implementing it in practice.

The results obtained on the Meiko CS1 show that a single process is fastest for the coarse grids. The computation involved is small so that any communications impose significant overheads. For the finer grids, the maximum number of processes should be used. Such grids can have a very large number of points. It is appropriate to share the computational work load across a number of processes, at the expense of communications. Although, the 'staircase' configuration strategy does not give the shortest execution time over one half-cycle, it requires less cycles for the approximate solution to converge. Hence, the 'staircase' configuration strategy provides the shortest time for convergence.

An analysis of a comparison between reduced-type and exchange-type agglomeration has not been conducted. According to the simple performance model their execution time for exchange-type agglomeration should be shorter. This does not take in account the complex issues involved in communications. These are likely to be more prominent in this case, making analysis difficult. Further timing experiments need to be conducted to measure the execution time required for the convergence of the solution. These results should show the advantages of the staircase configuration strategy quantitatively.

At present not all the data arrays used in the multigrid program are decomposed. This imposes a relatively low upper limit to the number of grids that can be stored by each process. The memory required to store these arrays is the same, regardless of the number of processes used. Further work may be directed towards decomposing these arrays to allow larger grids to be used.

References

[1] W. L. Briggs. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1987.

[2] S. R. Chapple. *PUL-RD Prototype User Guide*. Edinburgh Parallel Computing Centre, Edinburgh, United Kingdom, 1993.

[3] M. H. Mawson. Multigrid solution methods on the sphere. Meteorlogical Office's documentation accompanying the serial version of the algorithm, 1993.

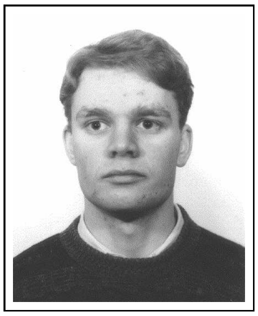
[4] B. McInnes. Met. office sequential multigrid code. Details of the routines and variables used in the serial version of the algorithm, 1993.

[5] B. McInnes. The parallelisation of a multigrid algorithm for the united kingdom meteorological office. Technical report, Edinburgh Parallel Computing Centre, Edinburgh, United Kingdom, 1993.

[6] Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee. *MPI: A Message-Passing Interface Standard*, April 1994.

[7] W. H. Press, Teukolsky S. A., W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, Cambridge, New York, Victoria, second edition, 1992.

[8] S. Vanderwalle and R. Piessens. A comparison of parallel multigrid strategies. Technical report, Katholieke Universiteit Leuven, Elsevier Science, B. V., North-Holland, 1989.



Mark White is currently in his fourth year studying Physics with Computer Applications at Heriot-Watt University, Edinburgh. Following his Honours graduation in 1995, he intends either to begin a PhD in some related area, or begin a career in industry.
email: phymgew@caledonia.hw.ac.uk
This project was supervised by Julian Parker (jp@epcc.ed.ac.uk) and Hon Yau (hwyau@epcc.ed.ac.uk).