Optimising Overlapped and Non-Overlapped Parallel Algorithms Using Bottleneck Analysis: A Case Study

M.J. Zemerly, J. Papay and G.R. Nudd

Parallel Systems Group, Department of Computer Science,
University of Warwick, Coventry, UK
Email: {jamal, yuri, grn}@dcs.warwick.ac.uk

Abstract

In this paper a methodology for parallel algorithm selection and optimisation based on performance prediction and bottleneck analysis is presented. The performance prediction method used here is based on an instruction set characterisation technique which assumes the availability of the sequential code and the compiler for the target machine. A "Processor Activity Graph" (PAG) which provides a graphical representation of the parallel system runtime behaviour is used to extend the applicability of the characterisation technique to overlapping parallel algorithms. The bottleneck analysis assumes that a bottleneck is caused by the slowest component of a computing system. These components are: memory, processor, communication and I/O. Three dimensionless ratios are used to identify and quantify bottlenecks, these are: the B-ratio, the communication-computation ratio and the memory-processing ratio. The optimisation methodology is illustrated and validated using a communication intensive linear solver algorithm (Givens rotations) which was implemented on a mesh connected distributed memory parallel computer (128 T800 Parsytec SuperCluster) using non-overlapping and overlapping parallelisation techniques.

1. Introduction

One of the main concerns of parallel computing is to port sequential programs onto a multiprocessor platform. This process requires to take into account the resource limitations of the target machine such as the number of processors, size of memory, latency and bandwidth of the communication network. Performance is one of the key factors that must be taken into account in design, development and tuning of a parallel algorithm. The overall performance is affected by the characteristics of the target machine and the software. Performance prediction allows to change the parallelisation technique and machine parameters and to observe the results of the modifications without porting the code or changing the existing hardware. Typical performance measures for parallel systems are execution time, speedup, system utilisation, throughput, MIPS, MFLOP, etc. Execution time was selected as a performance metric because it is a meaningful measure and widely used in parallel computing. Two approaches are usually used for evaluation of parallelisation techniques (see Figure 1). The traditional approach relies on the actual evaluation following the parallel code development and implementation which is a time consuming operation. The other approach uses an early performance prediction and optimisation step to prevent excessive software development costs .

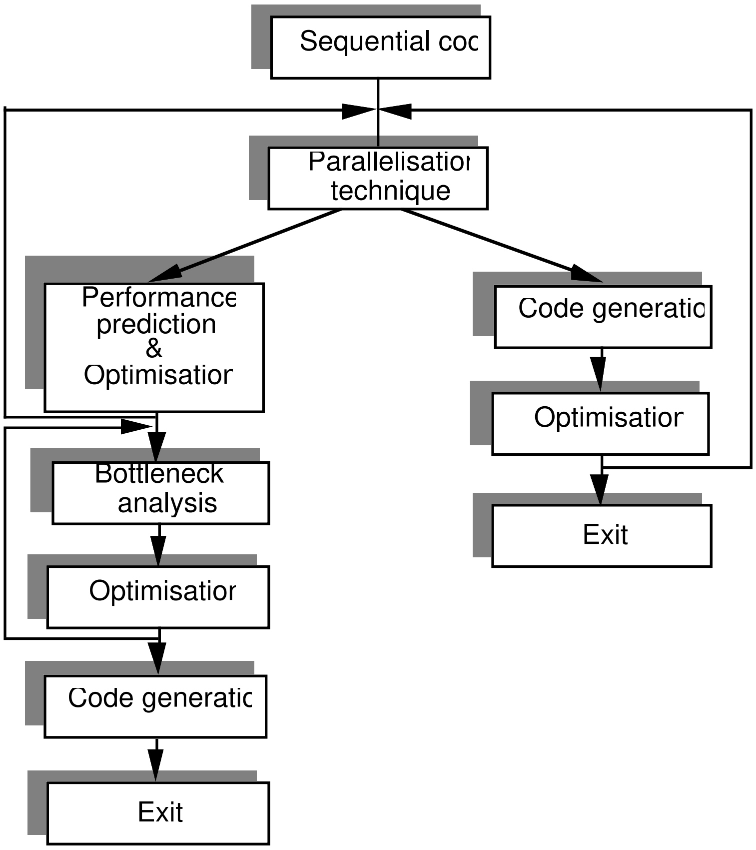


Figure 1 - Parallel algorithm development cycles

In this paper a recently developed early performance prediction method based on instruction set level characterisation is used [13]. This method has been extended by using a "Processor Activity Graph" (PAG) which enables to predict the performance of parallel algorithms exhibiting computation and communication overlap. A PAG provides a graphical representation of the parallel system runtime behaviour, shows the overlap between computation and communication, the level of parallelism, distribution of workload and load imbalance and facilitates performance prediction of parallel algorithms.

The optimisation task is concerned with reducing the time the algorithm spends in computation and communication. The computation can be optimised by selecting an appropriate parallelisation strategy which yields good load balancing. The reduction of the communication time can be achieved by using techniques such as the locality of communication, avoidance of hotspots in the communication network, overlap of computation and communication, etc.

A characterisation based bottleneck analysis is used to identify and quantify software and hardware problems before the actual code generation. The identification of bottlenecks within parallel systems is an important aspect of hardware and software design. This process usually involves examining the system behaviour under various load conditions which is a time and resource consuming operation. Bottlenecks are defined here as the parts of the system, either hardware or software which consume the maximum time or the slowest component of the system.

The bottleneck analysis involves the following steps: predict the execution time components of a certain workload, identify the time component responsible for the bottleneck (the slowest part), break down this component into its constituents and identify the sub-components causing the bottleneck. Optimisation of the software subroutines and/or hardware utilisation causing the bottleneck can improve the system performance. This operation can be iterated until no further optimisation is possible.

This paper is organised in the following way: in section 2 a background to the performance prediction and bottleneck analysis is given; section 3 introduces a methodology for characterisation and bottleneck analysis which are based on instruction level analysis; section 4 provides a case study to illustrate and validate the proposed methodology. The case study selected is a communication intensive linear solver algorithm (Givens rotations) which was implemented on a transputer-based distributed memory parallel computer (128 T800 Parsytec SuperCluster).

2. Related Work

Performance prediction gives an indication on the expected capabilities of a parallel system executing a certain application or a benchmark. Many researchers have characterised parallel systems on the component level (e.g. cache, CPU, etc.) with or without combining the effects of all the components [6]. Hennessy and Patterson [6] provided a comprehensive characterisation of a single micro-processor through characterising its components. Namely, the CPU, the memory hierarchy and the I/O. They argued that the execution time is the most meaningful performance measure for computer systems and they centred their processor characterisation on this measure. For each component they derived a performance equation based on the characteristics of that component.

Hockney [7] developed a model for SIMD and MIMD machines based on three program and hardware parameters. The program parameters are: s^s - total number of scalar floating point operations, s^c - total amount of data transferred and q^c- total number of messages. The characteristics of processor, memory system, interconnection network and operating system are given by parameters such as: r^s_∞ - asymptotic performance of scalar operations, r^c_∞ - asymptotic communication bandwidth, $n^c_{1/2}$ - half performance message length. The output of the model is a timing formula which is the sum of times for scalar operations communications. The timing formula does not consider overlapping of computation and communication.

Saavedra-Barrera et al. [11] described a way to predict performance by combining benchmarking and hardware characterisation. The idea is to develop and use a machine characteriser which measures the performance of a given system in terms of a Fortran abstract machine. This analysis yields a set of parameters which characterises the system and spotlights its strong and weak points. Each parameter provides the execution time for some primitive operations in Fortran. The authors characterised a large number of machines ranging from workstations to supercomputers.

Basu et al. [2] derived simple analytical execution models for evaluating the performance of parallel message passing systems. They modelled the execution of an algorithm as repetitive steps of computation and communication. Both overlapped and non-overlapped computation-communication execution models have been used. In a non-overlapping model all processors perform computation modules followed by synchronisation and exchanging of data. In overlapping models the computation and communication are completely or partially overlapped. Basu et al. used measurements of overlapping fractions between communication and computation from the parallelised code and extrapolated the results to larger systems. The overlapping occurs on he same processor (asynchronous communication) and does not deal with overlapping on different processors.

Desprez et al[4] describe Gaussian elimination with pivoting using column scattered data decomposition implemented on Intel iPSC/860 with 16 processor and Paragon with 64 processor machines. The authors investigated the pipeline and the broadcast algorithms for fully connected and ring topologies. Also synchronous and asynchronous message passing protocols have been analysed. The authors provide a complexity analysis and execution time predictions and measurements for the algorithms with various matrix sizes.

Gustafson [5] argues that almost every computer is limited not by the speed of the arithmetic unit but the memory bandwidth and latency. Using several examples the author showed that this problem is becoming visible even for workstations not just for parallel computers. Gustafson introduced the idea of characterising the system performance not by the Mflop rate but by the number of delivered Mword/s.

Hollingsworth [8] in his thesis describes a monitoring based bottleneck analysis methodology using an iterative process of refining the answers to three questions concerning performance problems. These three questions are why is the application performing poorly, where is the bottleneck, when does the problem occur. The why answer identifies the type of bottleneck (e.g. communication, I/O etc.). The where answer isolates a performance bottleneck to a specific resource used by the program (e.g. disk, memory etc.). Answering the when question isolates a specific phase of the program execution.

3. Characterisation and Bottleneck Analysis Methodology

The instruction level characterisation and bottleneck analysis are based on identification of the following components of the total execution time [14]:

- processing (CPU, FPU)
- memory (internal, external),
- communication (initialisation, distribution, collection, broadcast etc.)
- I/O (disk read and write).

These components can be further analysed and potential bottleneck problems identified as will be shown later.

3.1. The Computation Component

The instruction level characterisation method described in [13] is used here to predict the computation component time (processing and memory), which is given by:

$$T_{comp} = t_0 + (n_{ex} + n_{mem}) \times t_{cyc} \qquad (1)$$

For the Parsytec system studied here equation (1) becomes:

$$T_{comp} = (\underbrace{n_{cpu} + n_{fpu} - n_{fpu\_over}}_{n_{ex}} + \underbrace{n_{intmem} + n_{extmem} + n_{inst}}_{n_{mem}}) \times t_{cyc} \qquad (2)$$

where

t_0	start-up time for vector processors
n_{ex}	total number of cycles to execute the instructions
n_{mem}	total number of memory cycles
n_{intmem}	additional time spent to access internal memory
n_{exmem}	additional time spent to access external memory
n_{inst}	total number of cycles required to fetch instructions from external memory
n_{cpu}	total number of CPU cycles
n_{fpu}	total number of FPU cycles
n_{fpu_over}	total number of FPU cycles overlapping the CPU
t_{cyc}	processor cycle time

The values of variables in equations 2 can be obtained from the analysis of the assembly code of the sequential program as will be shown later.

3.2. The communication Component

In this section various communication models used in this paper are presented for the Parsytec SuperCluster. The analytical models cover the following classes of communication patterns which are typical for scientific applications: individual point-to point and collective communications such as broadcast, collection and data distribution. The total communication cost of a parallel algorithm T_{comm} can be written as:

$$T_{comm} = T_{cini} + T_{dist} + T_{coll} + T_{cst} \qquad (3)$$

where T_{cini} is the total initialisation cost of all the communication links (T_{ini}(d)) required between processors. The cost of establishing a link between two processors d distant away is given by:

$$T_{ini}(d) = \alpha + \beta d \tag{4}$$

For the Parsytec SuperCluster α=939.86 and β=272.05.

T_{dist}, T_{coll} and T_{cst} are the cost of initial distribution, final collection of data and total communication in all stages of the algorithm respectively.

3.2.1. Point-to-point communication

The time of the point-to-point communication depends on both the message length (l) and the distance (d) the message has to travel (the number of hops). An analytical communication model for quiet networks is described by Tron [12]. In this report a simpler communication model based on the work of Bomans [3] and Norman [10] is used and given by:

$$T_{com}(l,d) = \begin{cases} \alpha_1 + \beta_1 l + \gamma_1 d + \delta_1 ld & l \leq p_s \\ \alpha_2 + \beta_2 l + \gamma_2 d + \delta_2 ld & l > p_s \end{cases} \tag{5}$$

For the Parsytec SuperCluster the packet size, p_s, is equal to 120 bytes, and the parameters of equation (5) obtained by least square fitting of measured communication times are given in Table 1.

	α	β	γ	δ
$l \leq p_s$	51.1	0.27	8.86	0.68
l £ p_s	-81.5	1.33	93.04	0.05

Table 1 - Parameters of the quiet network communication model

3.2.2. Collective communications

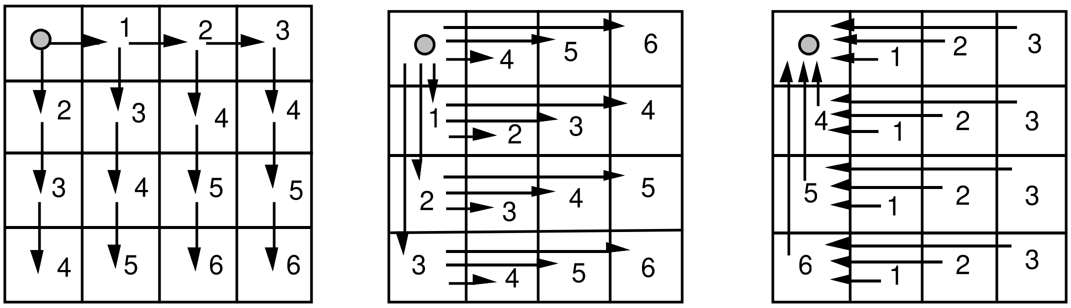
Three types of collective communication models are studied here: broadcast, distribution and collection.

3.2.2.1. Broadcast

Broadcast is a frequently occurring communication pattern in numerical applications. A simple broadcast implementation is a sequential loop in which the data is sent from the source processor to all the other processors one by one. The time cost of non-optimised broadcast is given by:

$$T_{nobroad}(l,p,k)=\sum_{i=0}^{p-1}T_{com}(l,d_{k,i}) \qquad (k\neq i) \qquad (6)$$

where l is the message length and $d_{k,i}$ is the distance between the broadcasting processor k and the i-th processor. In order to improve the performance of the broadcast subroutine several communication links should be activated and used in parallel. This solution enables full utilisation of the message passing capacity of the communication network and avoids long distance communications. Figure 2 shows the scheduling of communication steps for optimised broadcasting, distribution and collection on a 4x4 processor grid is given in Figure 3, the sequence of communication steps is marked by numbers.



(a) (b) (c)

Figure 2 - Sequence of communication steps for optimised broadcast (a), distribution and collection (c)

The analytical model in equation (7) gives the cost of optimised broadcast. The model considers only the communication to the neighbours and therefore the distance is 1.

$$T_{obroad}(l,p,k)=\sum_{i=0}^{neighbours(p,k)}T_{com}(l,1) \qquad (7)$$

where k is the broadcasting processor and neighbours(p,k) is the number of active neigbours of processor k in a p processor array.

3.2.2.2. Data distribution and collection

The data distribution and data collection are normally the initial and the final steps of parallel processing respectively. In the distribution step, the data is read in by a single processor which distributes it to the rest of the processor array. In the collection step, the results of all processors in the processor array are gathered by one processor. In a non optimised data distribution or collection the cost is equal to the sum of individual communications.

$$T_{nodist}(l,p,k) = T_{nocoll}(l,p,k) = \sum_{i=1}^{p-1} T_{com}(l_i, d_{k,i}) \qquad (8)$$

where l is the message length and $d_{k,i}$ is the distance between the source processor, k and the destination, i.

The optimised data distribution and collection (Figure 2(b) and 2(c)) activate several communication links in parallel. The time cost of optimised data distribution and collection is given by:

$$T_{odist}(l,p,k) = T_{ocoll}(l,p,k) = \sum_{j=y(k)+1}^{dimy-1} T_{com}(l_{dimx}, d_{y(k),j}) + \sum_{i=x(k)+1}^{dimx-1} T_{com}(l_i, d_{x(k),i}) \qquad (9)$$

where $x(k)$ and $y(k)$ are the x and y coordinates of processor k, dimx and dimy are the x and y dimensions of the processor grid and l_{dimx} is the size of data for processors in the x direction of the grid.

3.3. The I/O Component

The cost of I/O assuming a dedicated system can be expressed by:

$$T_{input}(l) = \alpha_{read} + \beta_{read} l \qquad (10)$$

$$T_{output}(l) = \alpha_{write} + \beta_{write} l \qquad (11)$$

The coefficients of the formulas above can be obtained by fitting a linear model to measurements provided on the target machine (for the Parsytec machine these coefficients are α_{read}=642.16, β_{read}=3.73, α_{write}=1583.73 and β_{write}=2.2).

3.4. Parallel execution time

The execution time of a parallel algorithm is given as:

$$T_{xp}(N,p)=T_{seq}(N)+T_{par}(N,p)+T_{comm}(N,p)+T_{I/O}(N) \qquad (12)$$

where T_{par} is the execution time of the algorithm part that can be parallelised and can be derived from PAG as will be shown in the case study in Section 4, T_{seq} is the execution time of the serial part of the algorithm including initialisation time, T_{comm} is the total communication overhead and $T_{I/O}$ is the input/output time.

3.5. Bottleneck analysis

A prediction of the components constituting the execution time to identify the slowest part of the system is first carried out using the characterisation method described before. Also three metrics are used to identify bottlenecks in the system; these are the B-ratio, the communication to computation ratio and memory to processing ratio. The B-ratio (B stands for bottleneck) of an execution time component (i.e. processing, memory, communication and I/O) is the ratio of the component time to the sum of all other components' times.

$$B_{component}=\frac{T_{component}(N,p)}{T_{xp}(N,p)-T_{component}(N,p)} \qquad (13)$$

where $T_{component}=T_{proc}$, T_{mem}, T_{comm} or $T_{I/O}$.

The B-ratios are dimensionless and a simple comparison between them will clearly identify a bottleneck problem when visualised in the same plot. The best performance (where bottlenecks are minimal) is obtained when all the B-ratios are equally balanced. The min-max value of all the B-ratios, provided its value is less than 1, is used to determine the optimum number of processors for the application and problem size in question. The maximum B-ratio value at each number of processors is first determined as follows:

$$\max(B\text{-}ratio)=\max_{i=1}^{p}(B_{proc},B_{mem},B_{comm},B_{I/o}) \qquad (14)$$

The min-max B-ratio value is the minimum of the $\max(\text{B-ratio})$ function and its position on the graph shows the optimum number of processors. Another measure to obtain the optimum number of processors that takes into account the execution time as well as the B-ratios is the min-max B-time. The $\max(\text{B-time})$ is given by:

$$\max(\text{B-time}) = \max_{i=1}^{p}\left(\text{B}-\text{ratio}_i \times T_{xp}(i)\right) \tag{15}$$

The ccr and mpr identify bottlenecks, without considering the effects of I/O. If communication time exceeds computation time, or memory access time exceeds processing time, then the ratio in question will exceed 1, indicating a bottleneck. Ideally, both ratios should be less than unity.

$$\text{ccr} = \frac{T_{comm}(N,p)}{T_{comp}(N,p)} \tag{16}$$

$$\text{mpr} = \frac{T_{mem}(N,p)}{T_{proc}(N,p)} \tag{17}$$

Once an execution time component is identified as a bottleneck further analysis of its sub-components can be carried out to highlight any software or hardware related problems causing the bottleneck. These sub-components are: FPU, CPU, external memory, internal memory, initialisation of communication links, data distribution, data collection, other communication overheads, disk read time and disk write time.

The bottleneck analysis methodology consists of the following steps:

- 1. Prediction of execution times for a range of processors and the problem sizes.
- 2. Comparison of the main execution time components.
- 3. Analysis of the B-ratios, ccr and mpr. In this step the dominant time components (bottleneck) are identified for any problem size and the results are extrapolated for any number of processors.
- 4. Analysis of each sub-components can be carried out to highlight any software or hardware related problems involved in the bottleneck. This will allow optimisation of software or hardware use where possible.

Note that in step 1 only the execution time predictions are required for the bottleneck analysis and the measurements are provided for validation purposes.

4. Case study: Givens linear solver

4.1. Description of Givens rotations

Givens methods are most commonly used for finding the eigenvalues of matrices and in that case, there is a requirement for the matrix to be symmetric [1, 9]. For solving linear equations, the Givens method is applicable to general matrices. Givens transformation is defined by a 2×2 rotation matrix:

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \qquad (18)$$

where $c^2 + s^2 = 1$

A Givens rotation is used to eliminate the elements of a vector or a matrix as follows:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \qquad (19)$$

where

$$c = \frac{a}{\sqrt{a^2 + b^2}} \quad \text{and} \quad s = \frac{b}{\sqrt{a^2 + b^2}}$$

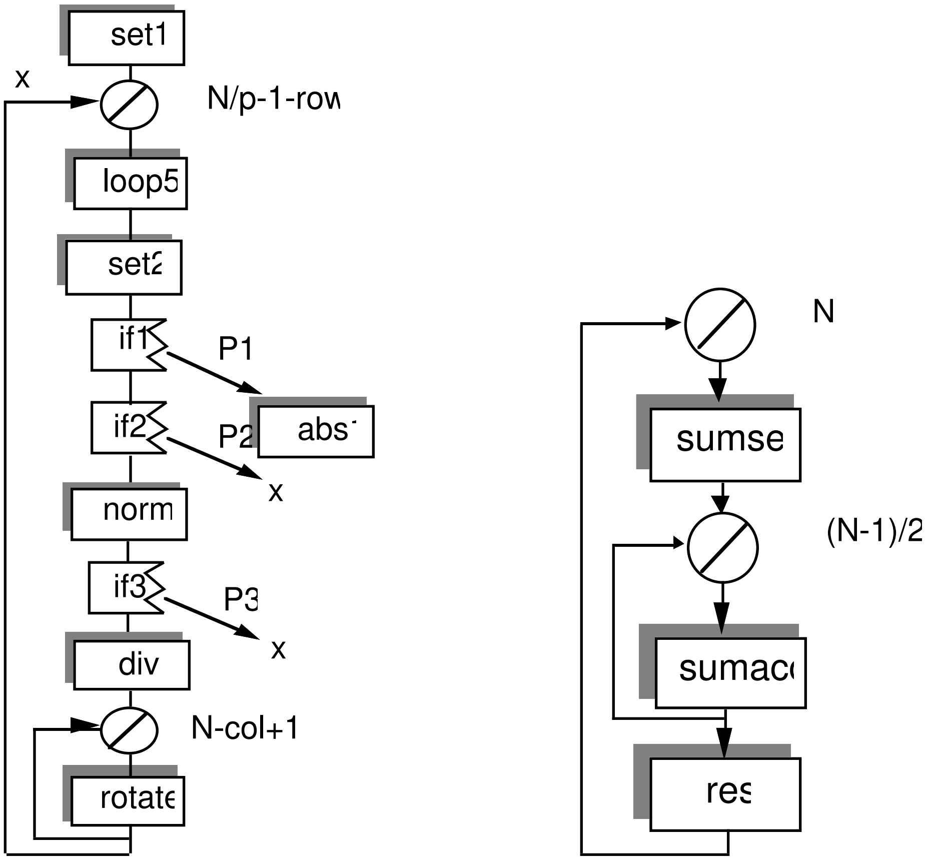
Unlike other linear solvers such as Gaussian elimination, Givens rotation does not require pivoting which is difficult to parallelise and has the advantage of being numerically more stable and inherently more accurate in theory than Gaussian elimination with pivoting. This follows from their properties of being orthogonal matrices. The disadvantage is that the Givens method requires 4 times more work than Gaussian elimination (4/3 N^3 as opposed to 1/3 N^3). However this problem is less important in parallel implementation. Applying a single Givens rotation reduces one element in the matrix to zero but simultaneously changes two rows. In the case of Gaussian elimination the row containing the element to be eliminated only is changed. Both these algorithms produce an upper triangular matrix which can be solved by back-substitution. The complexity of back substitution is $O(N^2)$ compared with the triangulation phase which is $O(N^3)$.

4.2. Characterisation of the sequential Givens algorithm

The Givens algorithm for solving a linear system with N equations can be decomposed into two computational stages. The first is the triangulation of the initial matrix, this stage is represented by the elimination block executed N times. The second stage is the substitution block which solves the triangular matrix. The total execution time of the sequential Givens algorithm can be given by:

$$T_x(N) = \sum_{i=0}^{N-1} T_{elim}(i,i,N,N) + T_{substitution}(N) \qquad (20)$$

where the function $T_{elim}(i,j,MN)$ represents the elimination of one column j, across M rows. and $T_{substitution}(N)$ represents the back-substitution of N columns. The computational modules of these blocks are given in Figure 3.



(a) (b)

Figure 3 - Computational modules of the elimination (a) and substitution (b) blocks

The sequential computational time of the elimination block is given by:

$$T_{elim}(\mathrm{row},\mathrm{col},y,x) = t_{set1} + (y-1-\mathrm{row}) \times (t_{loop5} + t_{set2} + t_{if\,1} + P1 \times t_{abs1} + t_{if\,2} + (1-P2) \times (t_{norm} + t_{if\,3} + (1-P3) \times (t_{div} + (x-\mathrm{col}) \times t_{rotate}))) \qquad (22)$$

The sequential computational time for the substitution block is given by:

$$T_{substitution}(N) = \sum_{i=n-1}^{0} \left(t_{sumset} + \sum_{j=n-1}^{i} t_{sumacc} + t_{res} \right) \qquad (22)$$

The instruction set characterisation which represents the computation component (see section 3.1) of the elimination and substitution blocks are given in Table 2. Note that the processor clock speed used here is 40 ηs.

	set1	loop5	set2	if1	abs1	if2	norm	if3	div	rotate	sumset	sumacc	res
n_{inst}	3	12	25	9	5	7	45	10	10	71	26	31	30
n_{cpu}	7	29	42	9	4	7	39	12	13	121	54	59	62
n_{fpu}	0	0	35	14	13	15	462	22	44	143	11	56	58
n_{in_mem}	0	0	4	2	2	1	12	1	3	7	1	4	3
n_{ex_mem}	12	36	103	25	23	14	148	30	45	231	83	106	110
n_{fpu_over}	0	0	3	8	3	4	15	5	3	20	0	3	18
Total cycles	19	65	181	50	39	33	646	60	102	482	149	222	215
Total µs	0.76	2.6	7.24	2.0	1.56	1.32	25.84	2.4	4.08	19.28	5.96	8.88	8.6

Table 2- Instruction Level Characterisation for elimination and substitution blocks

The probabilities P1, P2 and P3 for the three conditions if1, if2 and if3 obtained from measurements from the sequential code for different matrix sizes are given in Table 3.

N	P1	P2	P3
128x128	163.0 E-6	0	2.0 E-6
256x256	42.0 E-6	0	0
512x512	10.0 E-6	0	0

Table 3 - Probability values for elimination

4.3. Parallelisation methods

In the parallel Givens algorithm several orthogonal rotations can be applied simultaneously. The basic question is how to choose the sequence of rotations in order to minimise the amount of computations and the communication overhead. The triangulation of the initial matrix requires N algorithmic steps (k=1,N). The parallel linear solver based on Givens rotations can be obtained by extending the sequential algorithm with communication routines which provide distribution of data and the required message passing. Two parallelisation techniques are used here: non-overlapping and overlapping. The block-row data decomposition which divides the matrix horizontally and assigns the adjacent blocks of rows to the neighbouring processors is used in both methods.

4.3.1. Non-overlapping algorithm

In this method collective communications are used to collect the first rows from processors and distribute the rows back to the processors. The triangulation of the initial matrix requires N algorithmic steps ($k=1,N$) each consists of the sequence of the following operations:

- eliminate the elements of the actual column (this step will be referred to as A)
- the "master" processor collects the first rows from the processors
- eliminate the remaining elements of the column (this step will be referred to as B)
- distribute the resulting rows to the processors

The back-substitution is carried out on p processors where the last processor ($p-1$) substitutes its parameters and broadcasts them to all processors and then processor $p-2$ does the same and so on until processor 0 completes its work and writes the results. Figure 4(a) shows the PAG for the non-overlapping method for 4 processors each with 2 rows, the hashed areas represent computation, the empty areas represent idle processors and the arrows represent communication. It is worth noting that the non-overlapping algorithm uses optimised collective communication modules. The computational part of the parallel algorithm can be calculated from blocks of $T_{elim}(row,col,y,x)$. The parallel execution time derived from the PAG is given by:

$$T_{xp}(N,p) = T_{I/O}(N) + T_{cini}(p) + T_{odist}(N,p) + T_{triang}(N,p) + T_{parsub}(N,p) \qquad (23)$$

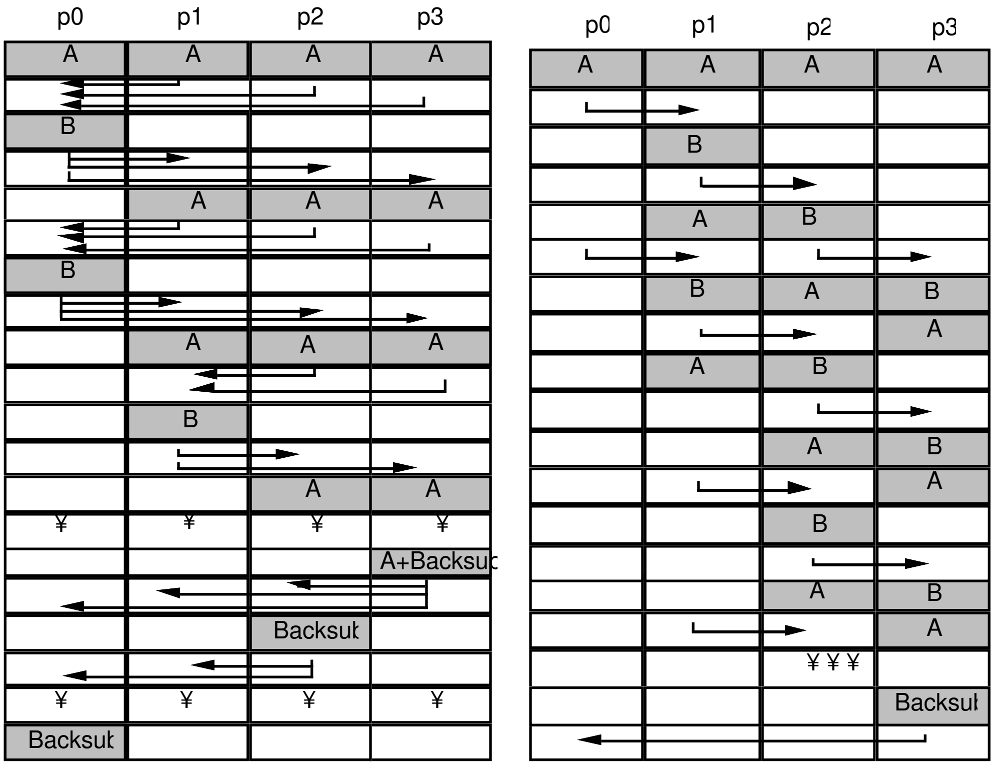
where

$$\begin{aligned} T_{triang}(N,p) = \sum_{k=0}^{p-2} \sum_{i=0}^{N/p-1} \Big(T_{elim}(0, \frac{kN}{p}+i, \frac{N}{p}, N) + T_{coll}(8\times(N+1-\frac{kN}{p}-i), p, k) + \\ T_{elim}(k, \frac{kN}{p}+i, p, N) + T_{odist}(8\times(N+1-\frac{kN}{p}-i), p, k)\Big) + \sum_{i=0}^{N/p-1} T_{elim}(i, N-\frac{N}{p}+i, \frac{N}{p}, N) \end{aligned} \qquad (24)$$

$$T_{parsub}(N,p) = T_{substitution}(N) + \sum_{k=p-1}^{1} T_{obroad}(\frac{8\times N}{p}, p, k) \qquad (25)$$

4.3.2. Overlapping algorithm (Pipeline)

The overlapping algorithm uses a pipeline technique which provides overlapping of computation and communication stages, as a result good scalability and speedup can be obtained. The triangular matrix is collected on processor p-1 where the back-substitution is performed. The results of the computation are sent to processor 0. The PAG of parallel pipeline Givens algorithm for four processors each with 2 rows is given in Figure 4(b).



(a) (b)

Figure 4 - Processor activity graphs for the parallel non-overlap (a) and pipeline (b) Givens algorithms.

The execution time prediction assumes that the last processor (p-1) of the pipeline is always busy after the pipeline is filled. Hence the execution time is equal to the time required to fill up the pipeline and the time taken by the last processor to complete its work. The execution time of the parallel pipeline Givens algorithm is given by:

$$T_{xp}(N,p) = T_{I/O}(N) + T_{cini}(p) + T_{initpipe}(N,p) + T_{pipe}(N,p) + T_{alone}(N,p) + T_{substitution}(N) + T_{send}(N,p) \qquad (26)$$

where $T_{initpipe}(N,p)$ is the initialisation time of the pipeline:

$$T_{initpipe}(N,p) = T_{elim}(0,0,\frac{N}{p},N) + \sum_{k=0}^{p-2}\left(T_{com}(8\times(N+1),1) + T_{elim}(0,0,2,N)\right) \qquad (27)$$

The time for the full pipeline is:

$$T_{pipe}(N,p) = \sum_{i=1}^{N-N/p}\left(T_{elim}(0,i,\frac{N}{p},N) + T_{elim}(0,i,2,N) + T_{com}(8\times(N-i),1) + T_{elim}(0,i,2,N)\right) \quad (28)$$

The elimination time for the data of the last processor (p-1) is:

$$T_{alone}(N,p) = \sum_{i=0}^{N/p} T_{elim}(i, N-\frac{N}{p}+i, \frac{N}{p}, N) \qquad (29)$$

The communication time required for processor p-1 to send the results to processor 0 is:

$$T_{send}(N,p) = T_{com}(8\times N, d_{0,p-1}) \qquad (30)$$

4.5. Performance prediction and bottleneck analysis

4.5.1 Performance Predictions

In this section the performance predictions of both algorithms are compared and the best algorithm is selected for bottleneck analysis in the following sections. Table 4 and Figure 5 show the execution times (in seconds) for both algorithms for different problems sizes and various number of processors.

N\p	1	2	4	8	16	32	64	128
non_overlap(128)	14.36	11.29	6.61	5.18	5.95	9.94	18.82	38.15
non_overlap(256)	110.09	84.56	46	30.16	27.85	40.07	71.07	140.21
non_overlap(512)	862.38	654.63	341.92	198.92	149.27	175.91	283.83	541.18
pipeline(128)	14.36	11.27	6.13	3.56	2.31	1.75	1.59	1.73
pipeline(256)	110.09	84.49	44.09	23.88	13.84	8.92	6.72	6.02
pipeline(512)	862.38	654.37	334.31	174.2	94.25	54.49	35.17	26.26

Table 4 - Execution time predictions for non-overlapped and pipeline Givens algorithms

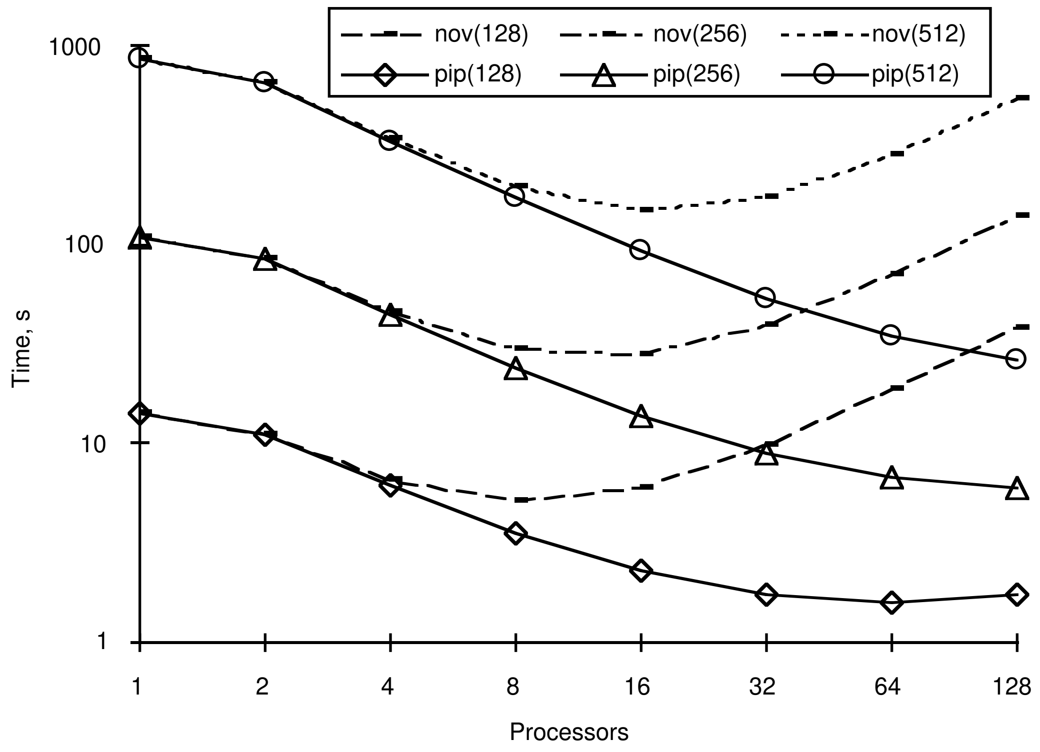


Figure 5 - Execution time predictions for non-overlapped and pipeline Givens algorithms

Figure 5 shows that the pipeline algorithm provides much better performance and scalability than the non-overlapped algorithm, hence the optimised pipeline algorithm will be selected for bottleneck analysis in the following sections.

4.5.2. Comparison of measurements and predictions for the pipeline algorithm

The Givens linear solver algorithm was implemented on the Parsytec machine and the execution times were measured to validate the predictions. The results of the predictions and the measurements for various task sizes (128, 256 and 512 equations) and number of processors are given in Table 5 and Figure 6.

N\p	1	2	4	8	16	32	64	128	256	512
measured(128)	14.15	10.12	6.8	3.53	2.31	1.83	1.7	1.97		
measured(256)	108.91	76.45	50.83	24.12	14	9.33	7.27	6.93		
measured(512)	854.44	593.74	393.11	177.5	95.58	56.19	37.51	29.67		
predicted(128)	14.36	11.27	6.13	3.56	2.31	1.75	1.59	1.73		
predicted(256)	110.09	84.49	44.09	23.88	13.84	8.92	6.72	6.02	6.6	
predicted(512)	862.38	654.37	334.31	174.2	94.25	54.49	35.17	26.26	23.79	25.49

Table 5 - Predictions and measurements for the pipeline Givens algorithm

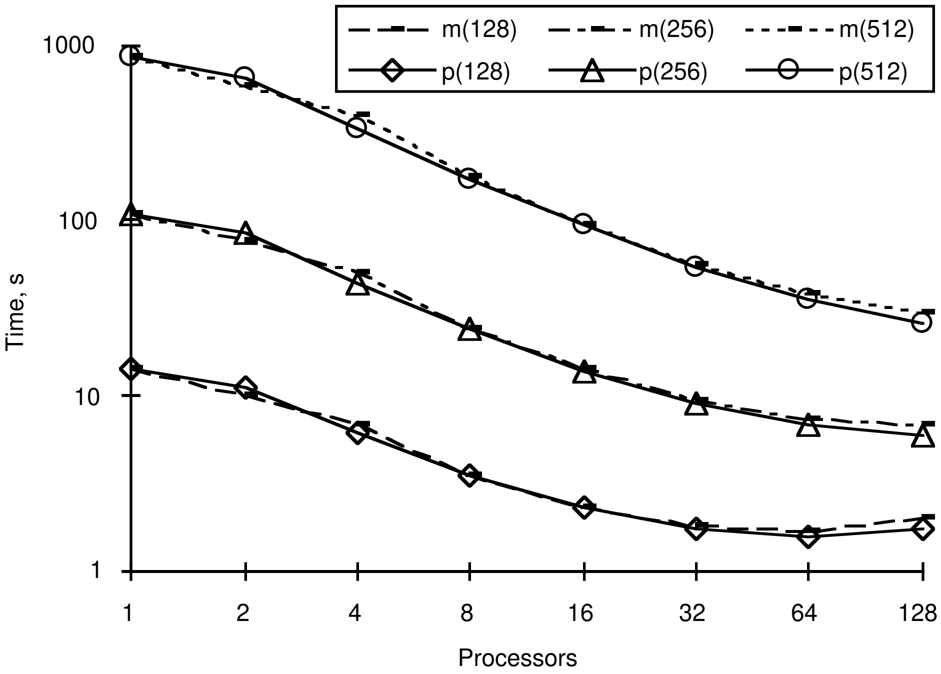


Figure 6 - Predictions and measurements for the pipeline Givens algorithm

Figure 7 shows the absolute percentage differences between the predicted execution times and the measurements for the optimised linear solver.

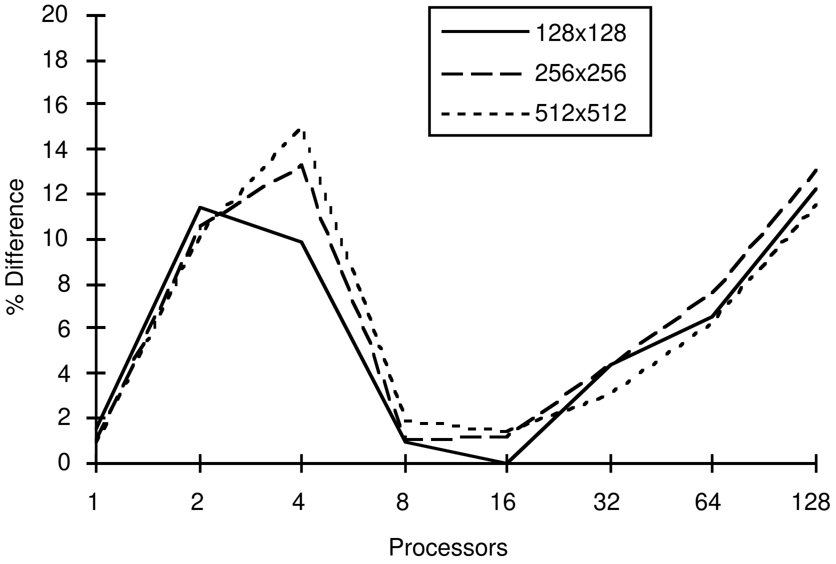


Figure 7 - The absolute percentage differences between the predictions and measurements

Figure 7 shows that the maximum absolute error between the predictions and measurements for the pipelined Givens linear solver is about 15%. A comparison between the measurements and predictions has been carried out on the non-overlapped algorithm and showed a maximun error of 12% [14].

4.5.3. Comparison of the main execution time components

Table 6 shows the execution time components (namely, processing, memory, communication and I/O times) for the pipeline Givens algorithm for 512 equations

N\p	1	2	4	8	16	32	64	128	256	512
Tproc(512)	436.86	329.35	165.0	82.87	41.78	21.3	11.19	6.35	4.42	4.41
Tmem(512)	417.34	314.29	157.5	79.09	39.89	20.34	10.68	4.24	4.24	4.23
Ti/o(512)	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17	8.17
Tcomm(512)	0	2.54	3.55	4.06	4.39	4.66	5.13	5.64	6.95	8.67

Table 6 - Execution time components for the pipeline Givens algorithm

Figures 8 shows the percentage execution time components for the pipeline Givens linear solver for 512 equations.

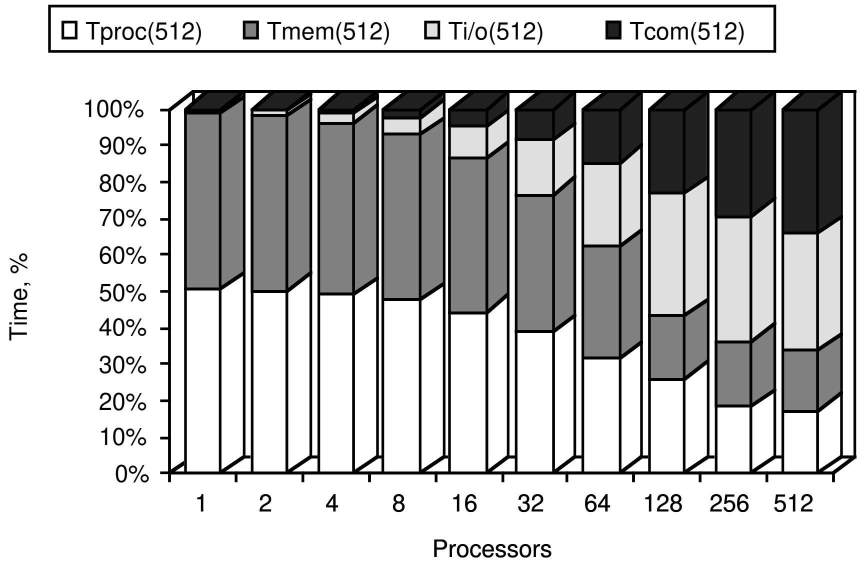


Figure 8 - Percentages of execution time components for 512 equations

Table 7 shows the communication-computation, the memory-processing and the B-ratios for the execution time components (processing, memory, communication and I/O) for the linear solver for 512 equations and different number of processors. The highlighted values are the maximum B-ratio (see equation 16). Figures 9 and 10 show a graphical representation of these ratios for 512 equations. Figure 9 shows that the communication-computation ratio exceeds unity after 512 processors indicating the start of a communication bottleneck. The memory-processing ratio is constant throughout due to the absence of a cache. This ratio will show the effect of a cache memory in systems where it exists.

N\p	1	2	4	8	16	32	64	128	256	512
mpr(512)	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.96
ccr(512)	0	0.004	0.01	0.02	0.05	0.11	0.23	0.45	0.8	1
Bproc(512)	1.02	1.01	0.97	0.9	0.79	0.64	0.46	0.32	0.22	0.21
Bmem(512)	0.93	0.92	0.89	0.83	0.73	0.59	0.43	0.3	0.21	0.19
Bcomm(512)	0	0.003	0.01	0.02	0.04	0.09	0.17	0.27	0.41	0.51
Bi/o(512)	0.01	0.012	0.02	0.05	0.09	0.17	0.3	0.45	0.52	0.47

Table 7 - mpr,ccr and B-ratios for processing, memory, communications and I/O for 512 equations

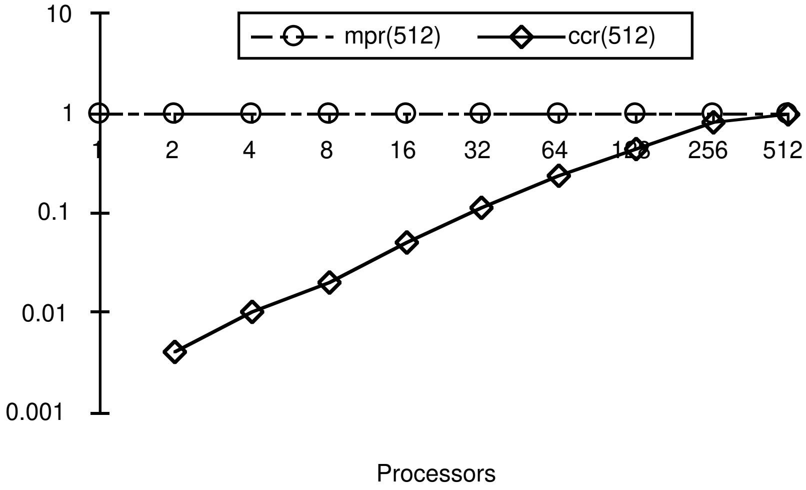


Figure 9. Communication-computation and memory-processing ratios for 512 equations

Figures 10 shows the B-ratios of the linear solver for 512 equations for various numbers of processors.

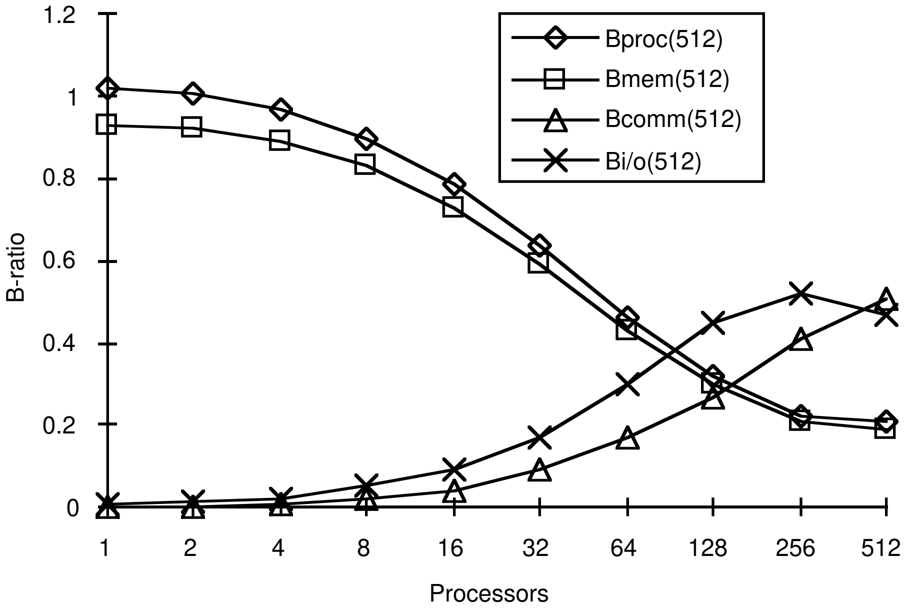


Figure 10 - B-ratio for processing, memory, communication and I/O for 512 equations

As can be seen from Figure 10 the I/O becomes a bottleneck after 128 processors while the CPU time is a bottleneck between 1 and 64 processors, for more than 512 processors the communication takes over. The optimal number of processors indicated by Table 10 and Figure 10 using the min-max(B-ratio) is 128 processors. Figure 11 Shows the max(B-time) for 512 equations yielding the same optimal number of processors at 128.

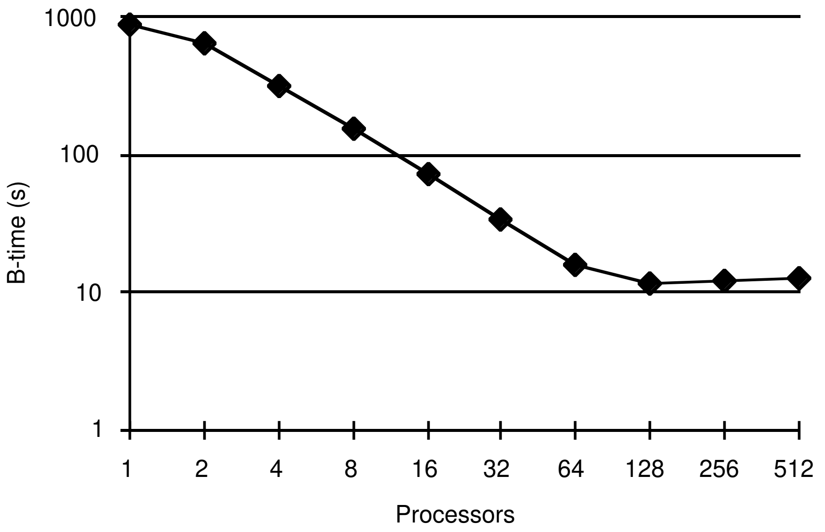


Figure 11 - The max(B-time) function for the parallel Givens algorithm for 512 equations

4.5.5. Breakdown of execution time components

Figure 12 shows the breakdown of execution time components for 512 equations and various number of processors.

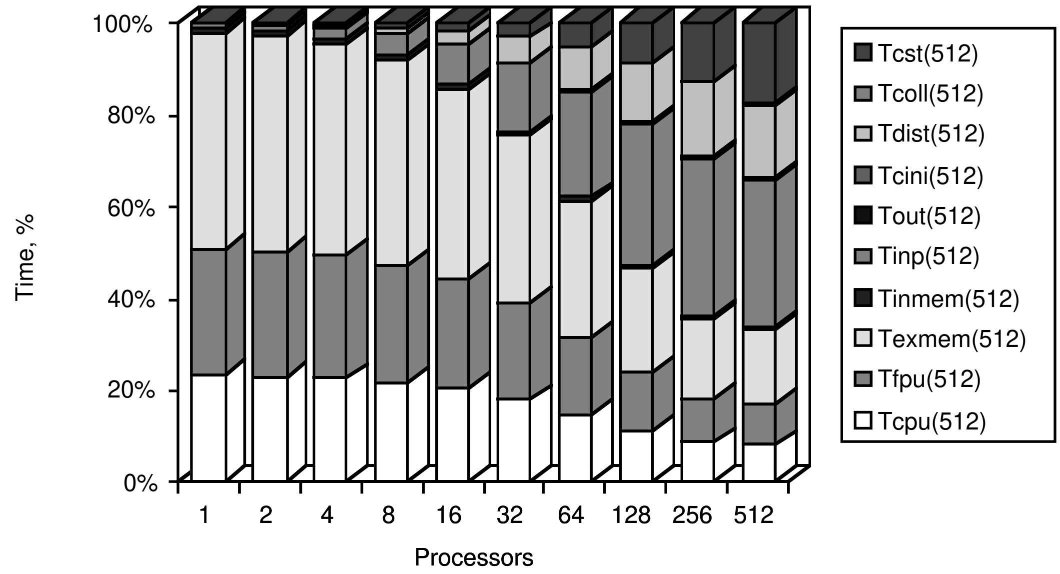


Figure 12 - Percentages of execution time sub-components for 512 equations

The breakdown of execution time components Figure 12 shows that within the processing time the CPU is about 45% , the external memory is about 98% of the total memory time, the communication time is dominated by T_{cst} (communication between neighbours at each stage) and T_{dist} (initial distribution of data) and the I/O time is dominated by disk read 99%.

5. Summary

In this paper we present a methodology for optimising parallel algorithms using characterisation based performance prediction [13] and bottleneck analysis. The bottleneck analysis assumes that a bottleneck is caused by the slowest component of a computing system. These components are: memory, processor, communication and I/O. Three dimensionless ratios are used to identify and quantify bottlenecks, these are: the B ratio, the communication-computation ratio and the memory-processing ratio. A new performance measure, B-time, is introduce which can be used to select the optimal number of processors. A "Processor Activity Graph" (PAG) has been introduced which is a useful graphical representation of the parallel system runtime behaviour. The PAG is used in the execution time prediction for both non-overlapping and overlapping algorithms. A linear solver based on Givens rotations is selected as the application and parallelised using two different techniques on a distributed memory parallel computer (128 T800 Parsytec SuperCluster). The first is a non-overlapping algorithm using collective communication modules such as broadcast, distribution and collection. The second is a pipeline (overlapping) algorithm which uses only simple point-to-point communication between neighbouring processors. The two algorithms use the same computational modules which were identified and extracted from the sequential code. We show that the pipelining technique is superior to the non-overlapping technique one achieving (for $N=512$) a maximum speedup of 30 for 128 processors. Whereas for the non-overlapping technique the maximum speedup achieved was 4 at $p=16$. A bottleneck analysis for the pipelined Givens linear solver algorithm is also provided and shows how the bottleneck varies with different problem sizes and numbers of processors.

Acknowledgement

This work is funded by the ESPRIT project 6942 (PEPS) - Performance Evaluation of Parallel Systems. The authors wish to thank Mr. Trevor Chambers and the Parallel Systems Group at Warwick University for the valuable discussions of this work.

Bibliography

1. S. Barnett Matrices Methods and Applications, Clarendon Press, Oxford 1990.

2. A. Basu, S. Srinivas, K. G. Kumar, A. Paulraj. A Model for Performance Prediction of Message Passing Multiprocessors Achieving Concurrency by Domain Decomposition. Proc. of the Joint Int. Conf. on Vector and Parallel Processing, (editor) Burkhart, H., Lecture Notes in Computer Science, 457, 10-13 September, Zurich, Switzerland, Springer-Verlag, pp. 75--85, 1990.

3. I. Bomans and D. Roose. Benchmarking the iPSC/2 Hypercube Multicomputer. Concurrency: Practice and Experience 1 (1), pp 3-18, 1989.

4. F. Desprez, B. Tourancheau, J.J. Dongarra. LAPACK Working Note 67: Performance Complexity of LU Factorisation with Efficient Pipelining and Overlap on a Multiprocessor. University of Tennesse, Knoxville, March (1994), To be published in Parallel Processing Letters

5. J.L. Gustafson. Computer Intensive Applications on Advanced Computer Applications, D .J. Evans, G. R. Jouber and H. Lidell (Editors), Parallel Computing, pp. 75-89, 1991.

6. J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, 1990.

7. R. W. Hockney, C.R. Jesshop: Parallel Computers 2, Architectue, Programming and Algorithms., Second edition, Institute of Physics Publishing, Bristol, 1988.

8. J. K. Hollingsworth. Finding Bottlenecks in Large Scale Parallel Programs. PhD Thesis, Dept. of Computer Science, University of Wisconsin-Madison, 1994.

9. J. J. Modi. Parallel Algorithms and Matrix Computation, Clarendon press, 1988.

10. M. G. Norman and P. Thanish. Models of Machines and Computation for Mapping in Multicomputers. ACM Computing Surveys, vol 25, no.3, September, pp 263-301, 1993.

11. R. H. Saavedra-Barrera, A. J. Smith, and E. Miya. Machine characterisation based on an abstract high-level language machine, IEEE Trans. on Computers 38 (12) (1989) 1659-1679.

12. C. Tron and B. Plateau. Modelling Communication in Parallel Machines Proc Performance Evaluation of Parallel Systems, University of Warwick, UK, pp 110-117, December 1993.

13. M. J. Zemerly, D. J. Kerbyson, E. Papaefstathiou, R. Ziani., J. Papay, T. J. Atherton and G. R. Nudd. Characterising Computational Kernels to Predict Performance on Parallel Systems. Proc. World Transputer Congress 94, Como, Italy, 5-7 September 1994, pp.105-119.

14. M. J. Zemerly, J. Papay and G. R. Nudd. Analysis of Bottlenecks. ESPRIT III PEPS Project (6942) Deliverable D5.4, University of Warwick, April 1995.