Estimating Computer Performance for Parallel Sparse QR Factorisation

David J. Miron* and Patrick M. Lenders

School of Mathematical and Computer Sciences, University of New England,
Armidale NSW 2350, Australia

Phone: 61 2 67732298 Fax: 61 2 67733312

david@turing.une.edu.au pat@turing.une.edu.au

Abstract. Performance estimates of a parallel computer during sparse matrix factorisation aid in the identification of overheads and the tuning of software. This paper proposes a technique which allows the computer parameters of computation speed, communication speed, latency and parallel efficiency to be estimated. The technique is based upon the use of mathematical models derived from a model problem in conjunction with experimental results. By combining the mathematical models with the experimental results, sets of simultaneous equations can be derived which can be solved for the above computer parameters. The technique is explained in the context of sparse QR factorisation.

1 Introduction

The primary focus of this work is the factorisation $A=QR$, where A is a sparse real $m \times n$ ($m \geq n$) matrix, Q is orthogonal, R is upper triangular and the number of nonzero entries in any row or column is less than 10%.

The sparse QR factorisation is important as it can be used to solve sparse linear least squares problems. These problems occur in areas such as geodesy [7], data mining [9], animal breeding [8] and finite element problems.

A sparse matrix is factorised by storing and operating only on the nonzeros in the matrix, thereby realising significant savings in storage requirements and factorisation time. However, sparse matrices can be so large that factorisation on single processor computers is not practical, therefore requiring the use of parallel computers.

Pozo [13] provides a theoretical study of the distributed memory multifrontal LU factorisation. His analysis is based upon *completion time*, being the time taken by the *multifrontal method* [3] to perform the numeric factorisation of a sparse matrix. Using some example matrices from the Harwell Boeing Collection [2], Pozo was able to give estimates for the completion time and parallel speedup of various parallel distributed memory computers.

* This work was done as part of a PhD dissertation with the CRC for Advanced Computation in the Research School of Information Sciences and Engineering at the Australian National University.

This work extends that of Pozo, applying a similar theoretical analysis to the sparse QR factorisation. A method is proposed for estimation of the computer parameters computation cost γ (seconds/operation), communication cost β (seconds/byte), latency α (seconds) and the parallel efficiency of the factorisation ϕ ($0 < \phi \leq 1$). In this context the computation cost γ is the average time for a computer to perform one operation, while the communication cost β is the time for the computer to transfer one byte of data between two processors.

The target architecture is a Fujitsu VPP300 computer at the Australian National University. This computer is a distributed memory vector parallel computer with 13 processors with each processor having a peak speed of 2.2 GFlops. Of the 13 processors 5 have 2 GB of RAM while the other 8 processors have 416MB of RAM with the processors connected by a crossbar switch that has a transfer rate of 570 MB/sec and a latency between processors of 570μ seconds.

To estimate the computer parameters computation cost γ , communication cost β , latency α and the parallel efficiency of the factorisation ϕ requires the construction of a model problem that is based upon a special case of a *relaxed supernodal elimination tree* [1, 3]. Using the model problem an equation can be derived that gives the theoretical completion time for parallel sparse QR factorisation using a multifrontal method. Actual completion times can then be found using an implementation of the parallel factorisation. This results in a set of simultaneous equations that can be solved for the above computer parameters.

In Section 2 a brief overview of multifrontal methods is given before proceeding to define the model problem in Section 3. Completion times are then presented in Section 4 with results given in Section 5. In Section 6 some conclusions are presented.

2 Multifrontal Methods

The multifrontal method of Duff and Reid [3] reduces the factorisation of a sparse matrix to the factorisation of a sequence of small submatrices called *frontal matrices*. A frontal matrix is associated with each column of the matrix A , with the factorisation of a frontal matrix corresponding to the elimination of a column in A . A factorised frontal matrix is dense and consists of a contribution to R and an *update matrix*, where the update matrix is made up of the rank-1 updates produced by the frontal matrix factorisation. The update matrix is stored on a stack to be later retrieved and summed into other frontal matrices.

The order in which frontal matrices are factorised and update matrices retrieved from the stack is governed by a graph structure called the *elimination tree*. An elimination tree is a tree of n nodes with each node corresponding to a column in the matrix A . The columns of the matrix A corresponding to the leaves of the elimination tree can be eliminated independently. The order in which columns are eliminated is determined by a postorder traversal of the tree.

Frontal matrices can be small, resulting in extensive overheads in stack operations and frontal matrix construction. These overheads can be reduced on vector computers by merging nodes of the elimination tree into “relaxed su-

pernodes". Thus factorisation of a relaxed supernode involves elimination of all the columns associated with that node from the frontal matrix. Frontal matrices associated with relaxed supernodes explicitly allow zeroes into the factorised frontal matrix [1, 3]. In this work the term elimination tree shall refer to a relaxed supernodal elimination tree.

3 The Model Problem

The model problem is based upon a relaxed supernodal elimination tree with the following structure.

Firstly, the elimination tree T is a balanced binary tree consisting of $l = 1 + \log_2 p$ levels and p leaves. A node $t \in T$ at level i of the elimination tree has a *granularity* $\kappa_t = \frac{\kappa}{\nu^i}$ where κ is the number of columns to be eliminated at the root node and $\nu > \sqrt{2}$ is the rate at which the granularity of the nodes in the tree decreases between levels and is a constant.

Secondly, all the frontal and update matrices have magnitudes:

$$|\mathcal{F}^t| = \frac{c_1 \kappa^2}{\nu^{2i}} \quad (1)$$

and

$$|\mathcal{U}^t| = \begin{cases} 0 & \text{if } i = 0 \\ \frac{c_2 \kappa^2}{\nu^{2i}} & \text{if } i > 0 \end{cases} \quad (2)$$

respectively where the magnitude is the number of elements in the matrix M . The parameters c_1 and c_2 are constants and their ratio represents the over-determinism of the frontal and update matrices respectively. An example of the elimination tree for the case $p = 4$ is given in Fig. 1.

Fig. 1. The elimination tree for $p = 4$

The elimination tree in Fig. 1 corresponds to a *separator tree* generated by applying the Recursive Bisection [4] reordering to matrices arising from finite

element problems. For finite element problems based on a 2 dimensional grid the granularity of the top level node κ would be proportional to $\sqrt{n}$ [5] while the rate at which the granularity changes between levels ν would be proportional to $\sqrt{2}$. For finite element problems arising from 3 dimensional grids it would be expected that κ would be proportional to $\sqrt[3]{n}$ and ν would be proportional to $\sqrt[3]{2}$.

Figure 1 also shows the processor to elimination tree mapping, the processors being labelled P_0, P_1, P_2 and P_3 respectively. In general terms the parallel factorisation of the elimination tree exploits the natural parallelism in the tree while at the same time distributing the factorisation of a frontal matrix $\mathcal{F}^t$ at level i over 2^{l-i} processors.

For each node $t \in T$ there is a *processor group* and for each processor group there is a *lead processor*. The lead processor is responsible for distributing a frontal matrix over the other processors within its group for parallel factorisation and the gathering of the resulting R contribution and the update matrix. On completion of the gathering, the lead processor then sends the update matrix to the lead processor of its parent node. It should be noted that R will not be distributed evenly over the processors, with processor P_0 containing the larger portion of R .

4 Completion Times

In this section equations for determining the completion times of the single processor and parallel factorisation are derived. Using these equations an equation for a two step factorisation is presented. A two step factorisation is used since in practice the number of leaves of the elimination tree is usually much greater than the number of processors available.

4.1 Single Processor Factorisation

On a single processor the multifrontal QR factorisation involves a postorder traversal of the elimination tree T . On visiting each node $t \in T$ a frontal matrix $\mathcal{F}^t$ is assembled and factorised.

The assembly involves the accumulation of update matrices $\mathcal{U}^s$ (for which s is a child node of t) into the frontal matrix $\mathcal{F}^t$. For each node $t \in T$ the assembly time is:

$$\mathcal{A}(t) = \gamma\lambda \left(\sum_{s \in \text{child}(t)} |\mathcal{U}^s| \right), \quad (3)$$

where $\lambda \geq 1$ relates to the efficiency of the indirect addressing associated with the assembly process, γ is the computation cost and $|\mathcal{U}^s|$ is the number of elements in the update matrix $\mathcal{U}^s$. In the ensuing discussion it is assumed that the cost of indirect addressing $\lambda = 1$.

After assembly the frontal matrix $\mathcal{F}^t$ is factorised. The factorisation involves the elimination of κ_t columns from the frontal matrix $\mathcal{F}^t$. This requires the computation of κ_t Householder vectors and rank 1 updates resulting in at least $4\kappa_t|\mathcal{F}^t|$ operations [6]. This gives an upperbound on the time to factorise a frontal matrix $\mathcal{F}^t$ as:

$$\mathcal{E}(t) = 4\gamma\kappa_t|\mathcal{F}^t|. \quad (4)$$

Thus an upperbound for the completion time for multifrontal QR factorisation on a single processor is the sum of the time required to factor each frontal matrix $\mathcal{F}^t$:

$$\mathcal{T}_{seq} = \sum_{t=1}^{\eta} (\mathcal{A}(t) + \mathcal{E}(t)), \quad (5)$$

where η is the number of nodes in the elimination tree. Substituting the frontal and update matrix sizes from (1) and (2) into (5) and simplifying gives a single processor completion time of:

$$\mathcal{T}_{seq} = 4\gamma\kappa^3c_1 \left[\frac{1}{1 - \frac{2}{\nu^3}} \left(1 - \left(\frac{2}{\nu^3} \right)^{l+1} \right) + \frac{c_2}{2\kappa c_1 \left(1 - \frac{2}{\nu^2} \right)} \left(1 - \left(\frac{2}{\nu^2} \right)^l \right) \right]. \quad (6)$$

4.2 Parallel Factorisation

The parallel factorisation involves sending update matrices from child nodes of the elimination tree to their parents. However this need only be done by one child. Further, the time to perform the parallel factorisation is governed by the height of the elimination tree $|h|$ where $|h| = \log_2 p + 1$ and h is a path from a leaf node to the root node of the elimination tree.

Let the time for sending k words of data from one processor to another be:

$$M(k) = \alpha + \omega\beta k, \quad (7)$$

where α is the startup cost, β is the per byte transmission time (communication cost) and ω the number of bytes per word. Therefore the cost of communicating an update matrix is:

$$\mathcal{C}(t) = \alpha + \omega\beta|\mathcal{U}^t|. \quad (8)$$

If it is assumed that the frontal matrix $\mathcal{F}^t$ is on one processor before distribution and gathered back onto that same processor after factorisation, then an upperbound for the distributed factorisation time for a frontal matrix $\mathcal{F}^t$ is:

$$\mathcal{E}(t, p) = 2(p-1) \left(\alpha + \omega\beta \frac{|\mathcal{F}^t|}{p} \right) + \frac{4\gamma\kappa_t |\mathcal{F}^t|}{\phi p}. \quad (9)$$

The first term of (9) reflects the communication cost of distributing and gathering the frontal matrix $\mathcal{F}^t$, while the second term is an upperbound on the time required to factor the frontal matrix $\mathcal{F}^t$ over p processors with the distributed factorisation having a parallel efficiency of $0 < \phi \leq 1$.

The parallel factorisation has a completion time of:

$$\mathcal{T}_{par} = \sum_{t \in h} [\mathcal{A}(t) + \mathcal{E}(t, p) + \mathcal{C}(t)]. \quad (10)$$

Making the appropriate substitutions in (10) and simplifying gives the following completion time for the parallel factorisation:

$$\begin{aligned} \mathcal{T}_{par} = & 4\gamma\kappa^3 c_1 \left[\frac{1}{2^l \phi \left(1 - \frac{2}{\nu^3}\right)} \left(1 - \left(\frac{2}{\nu^3}\right)^{l+1}\right) + \frac{\omega\beta}{2\gamma\kappa} \left(\frac{1}{1 - \frac{1}{\nu^2}} \left(1 - \left(\frac{1}{\nu^2}\right)^{l+1}\right) - \right. \right. \\ & \left. \frac{1}{2^l \left(1 - \frac{2}{\nu^2}\right)} \left(1 - \left(\frac{2}{\nu^2}\right)^{l+1}\right) \right) + \frac{c_2}{2\kappa c_1 (\nu^2 - 1)} \left(1 - \left(\frac{1}{\nu^2}\right)^l\right) \left(\frac{\omega\beta}{2\gamma} + \nu^2\right) + \\ & \left. \frac{\alpha}{4\gamma\kappa^3 c_1} \left(2^{l+2} - l - 6\right) \right]. \end{aligned} \quad (11)$$

4.3 Two Step Factorisation

Since the number of processors p is usually small compared to the number of leaves in the elimination tree T , the parallel algorithm uses two steps, a local step and a communication step [12]. The communication step is used for factorising that part of the matrix A which is associated with the nodes of the tree from level $2^i = p$ to level $i = 0$, while the local step is the sequential factorisation of the local subtrees that are on each processor.

Using (6) and (11) the completion time for the two step parallel factorisation can be derived. The completion time for the communication step $\mathcal{T}_{par}^l$ of the two step parallel factorisation is calculated by setting the number of levels in the elimination tree $l = \log_2 p$ in (11), while the completion time for the local step is given by:

$$\mathcal{T}_{local}^{l \rightarrow \infty} = 2 \frac{4\gamma\kappa^3 c_1}{\nu^3 \log_2 p} \left(\frac{1}{\nu^3 - 2} + \frac{c_2 \nu^{\log_2 p}}{2\kappa c_1 (\nu^2 - 2)} \right). \quad (12)$$

Equation (12) is derived from (6) by setting the rate at which the granularity decreases between levels of the elimination tree to $\frac{\kappa}{\nu^{\log_2 p + 1}}$. The quantity $\frac{\kappa}{\nu^{\log_2 p + 1}}$ is the granularity of the nodes of the separator tree at level $i = \log_2 p + 1$. The

factor of 2 in (12) is required since there are two subtrees at level $i = \log_2 p + 1$ who share a parent node at level $i = \log_2 p$. The time to factorise a frontal matrix of a node at level $i = \log_2 p$ is not included in (12) as this time is accounted for in (11). Hence the completion time for the two step combined parallel factorisation is:

$$\mathcal{T}_{2step} = \mathcal{T}_{par}^{l=\log_2 p} + \mathcal{T}_{local}^{l \rightarrow \infty}. \quad (13)$$

5 Results

The computer parameters computation speed γ^{-1} , communication speed β^{-1} and latency α , in addition to the parallel efficiency ϕ can now be estimated by performing a least squares fit of the model to the measured data.

The two step algorithm was implemented on the 13 processor Fujitsu VPP300 at the Australian National University using the Fujitsu Fortran 90 compiler with default options. The numerical factorisation was performed using the numerical linear algebra libraries of LAPACK [11] and ScaLAPACK [14]. LAPACK is used for the factorisation of frontal matrices associated with the nodes of the local subtrees while ScaLAPACK is used for the distributed factorisation of frontal matrices.

5.1 The Example Matrices

The number of rows m , columns n and nonzeros nz of the example matrices used to gather completion times are given in Table 1.

Table 1. Statistics for test matrices

Identifier	m	n	nz
PIGS1	3140	1988	8510
PIGS2	9397	6119	25013
PIGS3	28254	17264	75018
RAND1	50000	10201	199818
RAND2	100000	22801	399883
RAND3	125000	40401	499101

The example matrices fall into two categories. The first category consists of the three overdetermined matrices prefixed with “PIGS”. These matrices are from animal breeding statistics [8]. The second category of matrices are the three matrices prefixed with “RAND”. The “RAND” matrices are randomly generated and are used in the study of additive models from “data mining” [9]. The normal equations matrices associated with the “RAND” matrices are block tridiagonal.

The Recursive Bisection used to generate the elimination tree was similar to that implemented by Karypis and Kumar [10] with the Recursive Bisection continuing until the frontal matrices had a maximum size of 50 columns.

5.2 The Numeric Phase

The times for the two step factorisation were found for an arrangement of the processors into a $p \times 1$ grid for the distributed factorisation. The frontal matrices were distributed by rows over the processors using a blocking factor of 200. On the VPP300 $n_{1/2} \approx 200$, where $n_{1/2}$ is the vector length that produces half the peak performance of the vector units.

Two runs of the numeric factorisation were made by seeding the random number generator of the Recursive Bisection implementation with two different values. For each run the best time for two trials of the factorisation was recorded for the processor configurations of $p = 1, 2, 4$ and 8. The results are given in Table 2.

Table 2. Timings for two step combined parallel factorisation.

	κ	c_1	p=1	p=2	p=4	p=8
PIGS1	122 109	10 12	0.32 0.33	0.27 0.28 (1.19) (1.19)	0.23 0.26 (1.39) (1.27)	0.22 0.28 (1.45) (1.18)
PIGS2	186 189	19 18	1.49 1.55	0.91 1.04 (1.64) (1.49)	0.63 0.78 (2.37) (1.99)	0.54 0.71 (2.76) (2.18)
PIGS3	172 295	65 38	5.30 0.33	3.32 4.46 (1.60) (1.52)	2.44 3.47 (2.17) (1.95)	2.05 2.52 (2.59) (2.69)
RAND1	159 136	251 293	10.23 8.99	6.61 5.86 (1.55) (1.53)	4.77 4.12 (2.14) (2.18)	**** ****
RAND2	275 382	281 203	40.66 45.25	27.16 30.99 (1.50) (1.46)	18.55 22.40 (2.19) (2.02)	**** ****
RAND3	281 268	301 316	69.14 56.76	38.84 34.62 (1.78) (1.64)	26.83 23.39 (2.58) (2.43)	**** ****

The results for Run 2 in Table 2 are given in italics with the corresponding speedups given in parenthesis. Entries denoted by “****” indicate insufficient memory was available for the factorisation. The lack of memory was due to the fact that only 4 of the available processors had more than 416MB available. In addition to recording the completion times, the granularity of the top level node κ for each of the matrices was recorded for each run. With these values of κ , estimates for c_1 were computed using:

$$c_1 = \frac{|\mathcal{F}|}{\kappa^2} = \frac{(m - n + \kappa)}{\kappa}, \tag{14}$$

since $|\mathcal{F}| = (m - n + \kappa) * \kappa$ at the root of the elimination tree. The values for κ and c_1 are given in Table 2.

Using the values in Table 2 a set of four simultaneous equations can be constructed for each of the matrices for each run. For example, the equations (15), (16), (17) and (18) are those constructed for Run 1 of the matrix “Pigs1”:

$$\mathcal{T}_{seq}^{l \rightarrow \infty} = 0.32 \quad (p = 1), \tag{15}$$

$$\mathcal{T}_{2step}^{l=1} = 0.27 \quad (p = 2), \tag{16}$$

$$\mathcal{T}_{2step}^{l=2} = 0.23 \quad (p = 4), \tag{17}$$

$$\mathcal{T}_{2step}^{l=3} = 0.22 \quad (p = 8). \tag{18}$$

In the case of the “RAND” matrices only three equations are available since the results for the case $p = 8$ ($l = 3$) are not available. Therefore an extrapolation is used to estimate the completion time for this equation. The extrapolation is based on the fact that the times are decreasing by approximately $\frac{1}{\sqrt{2}}$. Therefore to compute the time for $p = 8$ for each of the “RAND” matrices the times for $p = 4$ are multiplied by $\frac{1}{\sqrt{2}}$.

For each matrix the appropriate values for the top level node granularity κ and the frontal matrix constant c_1 from Table 2 were substituted into the four simultaneous equations. The update matrix constant c_2 had a range of $0 \leq c_2 \leq c_1$. Inspection of the size of the update matrices generated from both runs showed a wide variance in size, therefore a value of $c_2 = \frac{c_1}{2}$ was chosen. The rate at which the granularity decreased between levels of the separator tree ν was set to 1.5 (since $\nu > \sqrt{2}$). The sets of equations were solved using Mathematica [15] with the results for Run 2 given in Table 3.

Table 3. Estimated VPP300 parameters using Run 2 results.

	γ^{-1} (Mflops)	β^{-1} (MB/sec)	α (μ secs)	ϕ
PIGS1	466	27	3430	0.47
PIGS2	799	125	11130	0.49
PIGS3	1417	69	22589	0.55
RAND1	810	128	3209	0.53
RAND2	1328	95	< 0	0.97
RAND3	1056	190	107629	0.55

The value in Table 3 denoted by “< 0” indicates a value less than zero and consequently an infeasible solution. The values in Table 3 are in effect an average over all processor configurations for each matrix with the results for Run 1 omitted since infeasible solutions were generated for half the matrices indicating that the model is extremely sensitive to parameter values.

Half of the matrices exhibited a computation speed $\gamma^{-1} = 1100$ Mflops which is consistent with the setting of the vector length to $n_{1/2}$. The latency α is unexpectedly high, increasing with problem size. It is conjectured that the high latency α is associated with overheads in ScaLAPACK.

6 Conclusions

Using the mathematical models developed in Section 4 in conjunction with the experimental results in Section 5, estimates for the computer parameters computation speed γ^{-1} , communication speed β^{-1} , latency α and parallel efficiency

ϕ were made. The high latency was the only unexpected result. It was also found that the parameter values are problem dependent. This work has laid the foundation for estimating a parallel computer's performance during sparse QR factorisation and can also be adapted to Cholesky and LU factorisations. This method has also been applied in a forward manner to predict the completion time of parallel sparse QR factorisation [12].

References

- [1] C. Ashcraft and R. Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Transactions on Mathematical Software*, 15(4):291–309, December 1989.
- [2] I. Duff. Harwell Boeing sparse matrix collection release I, 1992.
- [3] I. Duff and J. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.
- [4] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, pages 345–363, 1973.
- [5] J. Gilbert and R. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50:377–404, 1987.
- [6] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 2nd edition, 1989.
- [7] G. Golub and R. Plemmons. Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition. *Linear Algebra and its Applications*, 34:3–27, 1980.
- [8] M. Hegland. On the computation of breeding values. In *Proceedings of the CONPAR-90-VAPP IV, Joint International Conference on Vector and Parallel Processing*, pages 232–242, Zurich, 1990.
- [9] M. Hegland. Personal communication, February 1997.
- [10] G. Karypis and V. Kumar. Multilevel k -way partitioning scheme for irregular graphs. Technical Report 95-064, University of Minnesota, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, August 1995.
- [11] LAPACK users' guide. SIAM, May 1992.
- [12] D. Miron. *The Parallel Solution of Sparse Linear Least Squares Problems*. PhD thesis, Computer Science Laboratory, Research School of Information Sciences and Engineering, Australian National University, 1998.
- [13] R. Pozo. Performance modeling of sparse matrix methods for distributed memory architectures. In *Proceedings of Parallel Processing: CONPAR 92 - VAPP V*, number 634 in Lecture Notes in Computer Science, pages 677–688. Springer-Verlag, September 1992.
- [14] ScaLAPACK user's guide. Preliminary Draft, May 1996.
- [15] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.