33

Communication Latency Hiding in a Parallel Conjugate Gradient Method

Kevin McManus[1], Steve Johnson[2], Mark Cross[3]

Introduction

The diagonally preconditioned Conjugate Gradient Method (CGM) is commonly used in a range of applications for the iterative solution of $\mathbf{Ax} = \mathbf{b}$. The method lends itself to parallelisation by Domain Decomposition (DD) using a Single Program Multi-Data (SPMD) mapping onto Distributed Memory (DM) parallel platforms. While it is possible to achieve good parallel performance it is often the case that performance is limited by the communication performance of the parallel system. Performance of the parallel CGM method can be improved by reducing the impact of communication start-up latency and finite inter-processor communication bandwidth without modification of the algorithm. This invites an examination of the factors affecting parallel performance.

Unstructured mesh applications using Finite Element or Finite Volume techniques to solve differential equations with iterative solution procedures are commonly parallelised using overlapping DD methods based on mesh partitioning. Decomposition of the mesh into P sub-domains distributed over P processors allows concurrent loop execution with global consistency maintained through the exchange of the data between the sub-domains as required to satisfy the dependencies of the integration

1 Centre for Numerical Modelling and Process Analysis, University of Greenwich, London, SE18 6PF, UK email: k.mcmanus@gre.ac.uk
2 Address same as above.
3 Address same as above.
Eleventh International Conference on Domain Decomposition Methods
Editors Choi-Hong Lai, Petter E. Bjørstad, Mark Cross and Olof B. Widlund

method. Extension of the sub-domains to include all required data leads to each sub-domain being surrounded by a layer that overlaps adjacent sub-domains. Message passing is used to communicate data from the processor on which it is assigned into the overlap of the processor(s) on which it is accessed [RL90]. Renumbering of each sub-domain from global to local numbering permits an implementation without global data structures and hence a scalable mapping onto DM parallel hardware [MCJ95].

For many applications, execution time is dominated by the execution of loops in the solution of $\mathbf{Ax} = \mathbf{b}$. Amdahl's law [Amd67] gives the maximum theoretical speed-up for a parallel code running on an ideal machine. With a large problem on each processor Amdahl's law predicts near linear speed-up for parallel solvers when scaling with a constant problem size per processor. But linear speed-up is seldom achieved in practice. The parallel overhead resulting from the time needed to communicate the necessary data between processors restricts the achieved performance. Two strategies that can help to hide the communication overheads are examined in a parallel conjugate gradient method.

Parallel Conjugate Gradient Method

CGM is an established non-stationary iterative method for symmetric positive definite systems providing rapid convergence and $O(m)$ computational cost per step, where m is the number of non-zero components of $\mathbf{A}$. Preconditioning may be used to improve the condition number of the matrix $\mathbf{A}$ [GL89, BBC+94]. For a positive definite matrix preconditioner $\mathbf{M}$

$$\mathbf{Ax} = \mathbf{b} \quad \equiv \quad \mathbf{M^{-1}Ax} = \mathbf{M^{-1}b} \tag{1}$$

If the eigenvalues of $\mathbf{M^{-1}A}$ are clustered better than the eigenvalues of $\mathbf{A}$ then the preconditioned problem may converge in fewer iterations than the original problem. The Jacobi preconditioner is the diagonal of the $\mathbf{A}$ matrix that has the effect of scaling the quadratic form along the coordinate axes [She94]. While not the most effective preconditioner in reducing iterations its simplicity provides computational efficiency. A modification [LL94] which transforms of $\mathbf{Ax} = \mathbf{b}$ into $\check{\mathbf{A}}\check{\mathbf{x}} = \check{\mathbf{b}}$ where the components of $\check{\mathbf{A}}$, $\check{\mathbf{x}}$ and $\check{\mathbf{b}}$ are

$$\check{a_{ij}} = \frac{aij}{\sqrt{a_{ii}a_{jj}}}, \quad \check{x_i} = x_i\sqrt{a_{ii}}, \quad \check{b_i} = \frac{b_i}{\sqrt{a_{ii}}} \tag{2}$$

results in the diagonal of $\check{\mathbf{A}}$ being the identity matrix $\mathbf{I}$ and so the Jacobi preconditioner $\mathbf{M} = \mathbf{I}$. The CGM iteration can then be expressed as

$$\mathbf{u^{(k)}} = \mathbf{Ap^{(k-1)}} \tag{3}$$

$$\alpha^{(k)} = \frac{\rho^{(k-1)}}{\mathbf{p^{(k-1)T}u^{(k)}}} \tag{4}$$

$$\mathbf{x^{(k)}} = \mathbf{x^{(k-1)}} + \alpha^{(k)}\mathbf{p^{(k-1)}} \tag{5}$$

$$\mathbf{r^{(k)}} = \mathbf{r^{(k-1)}} - \alpha^{(k)}\mathbf{u^{(k)}} \tag{6}$$

$$\rho^{(k)} = \mathbf{r^{(k)T}r^{(k)}} \tag{7}$$

$$\beta^{(k)} = \frac{\rho^{(k)}}{\rho^{(k-1)}} \tag{8}$$

$$\mathbf{p^{(k)}} = \mathbf{r^{(k)}} + \beta^{(k)}\mathbf{p^{(k-1)}} \tag{9}$$

An overlapping DD implementation of this method requires three communications within the iterative loop. One overlap update is required for the matrix multiply and each of the two inner products require a global summation. The integration stencil results in the distributed $\mathbf{A}$ matrix on each processor being non-square, having coefficients that address $\mathbf{x}$ in the overlap. Values of $\mathbf{x}$ must therefore be copied into the overlap before they are addressed by the matrix-vector multiplication. Such an overlap update communication only requires communication between adjacent subdomains and so may be expected to scale well with increasing P. A distributed inner product loop results in each processor calculating a local inner product which must be summed to produce a global inner product. A binary tree communication pattern is used to minimise the number of messages required to calculate a global summation.

Substituting for $\mathbf{r^{(k)}}$ in Equation (7) gives

$$\rho^{(k)} = (\mathbf{r^{(k-1)}} - \alpha^{(k)}\mathbf{u^{(k)}})^{\mathbf{T}}(\mathbf{r^{(k-1)}} - \alpha^{(k)}\mathbf{u^{(k)}}) \tag{10}$$

which can be expanded to produce

$$\rho^{(k)} = \mathbf{r^{(k-1)T}r^{(k-1)}} + \alpha^{(k)\mathbf{2}}\mathbf{u^{(k)T}u^{(k)}} + \mathbf{2}\alpha^{(k)}\mathbf{r^{(k-1)T}u^{(k)}} \tag{11}$$

substituting $\mathbf{r^{(k-1)T}r^{(k-1)}}$ from Equation (7) now gives

$$\rho^{(k)} = \rho^{(k-1)} + \alpha^{(k)}(\alpha^{(k)}\mathbf{u^{(k)T}u^{(k)}} + \mathbf{2r^{(k-1)T}u^{(k)}}) \tag{12}$$

Calculation of $\rho^{(k)}$ now requires two inner products instead of one but no longer requires $\mathbf{r^{(k)}}$ and so may be moved forward in the loop to the same point at which $\alpha^{(k)}$ is calculated.

$$\mathbf{u^{(k)}} = \mathbf{Ap^{(k-1)}} \tag{13}$$

$$\alpha^{(k)} = \frac{\rho^{(k-1)}}{\mathbf{p^{(k-1)T}u^{(k)}}} \tag{14}$$

$$\rho^{(k)} = \rho^{(k-1)} + \alpha^{(k)}(\alpha^{(k)}\mathbf{u^{(k)T}u^{(k)}} + \mathbf{2r^{(k-1)T}u^{(k)}}) \tag{15}$$

$$\mathbf{x^{(k)}} = \mathbf{x^{(k-1)}} + \alpha^{(k)}\mathbf{p^{(k-1)}} \tag{16}$$

$$\mathbf{r^{(k)}} = \mathbf{r^{(k-1)}} - \alpha^{(k)}\mathbf{u^{(k)}} \tag{17}$$

$$\beta^{(k)} = \frac{\rho^{(k)}}{\rho^{(k-1)}} \tag{18}$$

$$\mathbf{p^{(k)}} = \mathbf{r^{(k)}} + \beta^{(k)}\mathbf{p^{(k-1)}} \tag{19}$$

The three inner products in Equations (14) and (15) can now be calculated using only a single commutative operation to perform the three global summations [BJG94]. If it takes less time to calculate an inner product than the time required to calculate a global summation then this algebraic modification will improve parallel performance.

Performance Estimation

A simple cost model can provide a predictor for parallel performance. The time t_{iter} required for each iteration is the sum of the calculation time t_{calc} and the

communication time t_{comm}. An estimate for the calculation time can be calculated from the number of floating point operations in each iteration and the floating point rate of the processor. Unfortunately such an estimate is unlikely to be an accurate reflection of practice as the time for each individual floating point operation is often highly dependent upon the success or otherwise of cache memory systems and superscalar pipeline processor architectures.

Communication time can be characterised in terms of the communication start-up latency which is the time required to send a single word message and the communication bandwidth or rate usually quoted in megabytes per second. The communication time consists of the time required for an overlap update t_{update} and the time required for a global commutative t_{global}. The communication time for the original CGM

$$t_{comm} = t_{update} + 2t_{global} \tag{20}$$

is reduced by the modification to

$$t_{comm} = t_{update} + t_{global} \tag{21}$$

The global commutative time can be approximated as

$$t_{global} \approx 2nt_{latency} \tag{22}$$

where n is the number of levels required in the binary tree communication

$$n = 1 + \lceil \log_2(P-1) \rceil \tag{23}$$

The overlap update time can be approximated as

$$t_{update} \approx 2S(t_{data} + t_{latency}) \tag{24}$$

where S is the maximum degree of sub-domain interconnection, t_{data} is the average time required to communicate an overlap data message. This can be approximated from the surface area of the sub-domain and the communication bandwidth B.

$$t_{data} \approx \frac{(\pi(\frac{6N}{P})^2)^{\frac{1}{3}}}{SB} \tag{25}$$

The achieved bandwidth (excluding latency) is a function of message length which can be crudely approximated as

$$B \approx B_{MAX}\left(1 - \left(\frac{0.8}{\log(\frac{\pi(\frac{6N}{P})^2)^{\frac{1}{3}}}{S})}\right)\right) \tag{26}$$

where B_{MAX} is the maximum bandwidth. The maximum bandwidth for parallel machines varies over the range of 10^5 – 10^8 words per second. Communication start-up latency $t_{latency}$ varies over the range of 5×10^{-6} to 10^{-3} seconds. Again these can only be estimated as practical considerations such as communication patterns and message contention will have a noticeable effect.

As mentioned in Section 33 if the modified CGM is to return improved performance then the time to calculate an inner product t_{dot} must be less than the time required to perform a global summation.

$$t_{dot} < t_{global} \tag{27}$$

As each inner product requires N/P floating point product and sum calculations the inequality may be expressed as

$$\frac{N}{P}(t_{fp-product} + t_{fp-add}) < 2(1 + \log_2(P-1))t_{latency} \tag{28}$$

In practice it is not actually helpful to decompose the dot product as t_{dot} will be dependent on the cache and superscalar pipeline performance of the processor which are functions of the vector length N/P and vector storage. Run-time evaluation of t_{dot} for the given problem size together with a run-time measurement of $t_{latency}$ provides an accurate determination of which solver to use for best performance. For a current generation platform t_{dot} may be around 10^{-8} seconds and $t_{latency}$ around 10^{-5} seconds. With a small number of processors, 8 for example, then the modification will be beneficial for N less than 61,000 ($\frac{N}{P} \approx 7,600$). For large numbers of processors, 256 for example, this figure rises to around 4.6 million ($\frac{N}{P} \approx 18,000$). The graph in Figure 1 shows that the expected loss of performance at low P is significantly less than the potential gain in performance which becomes most apparent when efficiency falls to around 50%. This figure is largely a function of the calculation to communication ratio of the platform. The development of communication technology (memory bandwidth) continues to be out-paced by the improvements in processor throughput [KKS97] and so this figure is expected to increase.

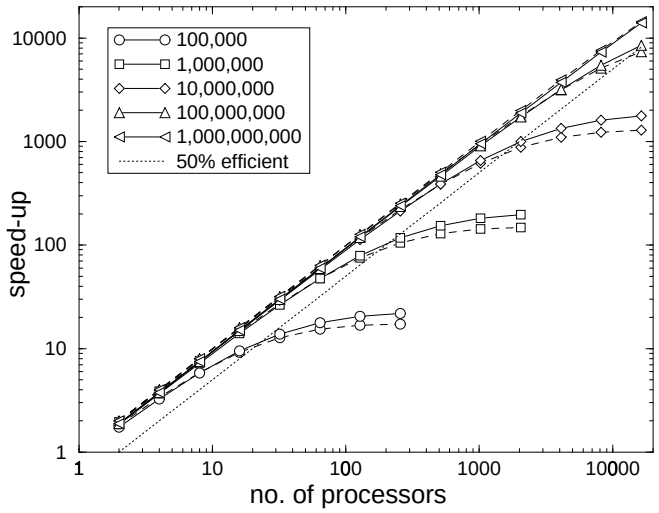

Figure 1 Predicted Speed-up for the Original and the Modified CGM.

The predicted speed-up for the CGM iteration in Figure 1 is an extrapolation the performance of current generation hardware to examine the potential scalability of

parallel applications. Without cache effects and constant N t_{dot} scales linearly with P but t_{global} scales as $P\log_2(P)$ and so it was hoped that the curves in Figure 1 would diverge accordingly. However, some of the performance gain is offset with very large P by the overlap update becoming dominant as the bandwidth drops for very short message lengths.

Asynchronous Communication

Asynchronous or non-blocking communication calls to allow calculation to overlap communication. Communication subroutines can initialise a communication and return from the subroutine call before completion of the communication. The communication can then be tested for completion (synchronised) at some future point in the code execution. Calculations that are independent of the communicated data can be performed immediately subsequent to an asynchronous communication call and synchronisation effected prior to the point at which the communicated data is accessed [KKS97]. Unstructured mesh parallel applications with local numbering schemes lend themselves to such communication hiding as it is a simple matter to identify the parts of each sub-domain that are required by other processors and renumber the distributed unstructured mesh so that the dependent data is numbered before the independent data with the overlaps being numbered last. Some of the loops can be split so that part of the loop is executed concurrently with the overlap update communication.

$$\mathbf{u^{(k)}} = \mathbf{A}\mathbf{p^{(k-1)}} \qquad \texttt{DEPT..TOTAL} \qquad (29)$$

synchronise

$$\mathbf{u^{(k)}} = \mathbf{A}\mathbf{p^{(k-1)}} \qquad \texttt{1..DEPT} \qquad (30)$$

$$\alpha^{(k)} = \frac{\rho^{(k-1)}}{\mathbf{p^{(k-1)T}}\mathbf{u^{(k)}}} \qquad (31)$$

$$\rho^{(k)} = \rho^{(k-1)} + \alpha^{(k)}(\alpha^{(k)}\mathbf{u^{(k)T}}\mathbf{u^{(k)}} + \mathbf{2r^{(k-1)T}}\mathbf{u^{(k)}}) \qquad (32)$$

$$\mathbf{x^{(k)}} = \mathbf{x^{(k-1)}} + \alpha^{(k)}\mathbf{p^{(k-1)}} \qquad (33)$$

$$\mathbf{r^{(k)}} = \mathbf{r^{(k-1)}} - \alpha^{(k)}\mathbf{u^{(k)}} \qquad (34)$$

$$\beta^{(k)} = \frac{\rho^{(k)}}{\rho^{(k-1)}} \qquad (35)$$

$$\mathbf{p^{(k)}} = \mathbf{r^{(k)}} + \beta^{(k)}\mathbf{p^{(k-1)}} \qquad \texttt{1..DEPT} \qquad (36)$$

asynchronous exchange

$$\mathbf{p^{(k)}} = \mathbf{r^{(k)}} + \beta^{(k)}\mathbf{p^{(k-1)}} \qquad \texttt{DEPT..TOTAL} \qquad (37)$$

The dependent data `1..DEPT` forms a layer on the inside surface of a sub-domain corresponding to the overlaps of adjacent sub-domains. Equation (36) calculates $\mathbf{p^{(k)}}$ over this inner surface layer and these values are sent to the adjacent sub-domain(s). At this point a receive is also posted to obtain the values of $\mathbf{p^{(k)}}$ required in the overlap. Calculation of $\mathbf{p^{(k)}}$ in Equation (37) and $\mathbf{u^{(k)}}$ in Equation (29) overlaps the communication time. The communication is synchronised before calculating the values of $\mathbf{u^{(k)}}$ that require the communicated data.

The success or otherwise of this strategy depends upon the implementation of the parallel system. If message passing is implemented using dedicated hardware then the potential for improvement is significant. If however the communication is implemented

entirely in software then there is little to be gained as the processor is required to perform both calculation and communication. Each asynchronous overlap update also requires an additional synchronisation message which can lead to performance degradation in comparison with synchronous communication.

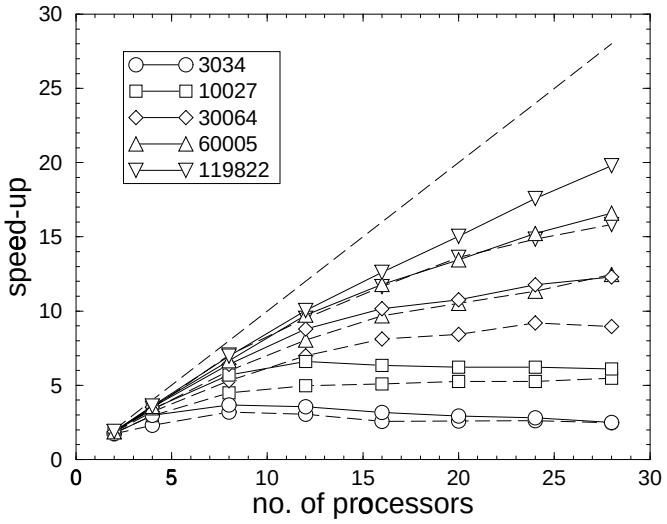

Figure 2 Speed-up using synchronous (dotted line) and asynchronous (solid line) communications in a CGM solver.

The curves in Figure 2 demonstrate a clear performance improvement using a Transtech Paramid in which each Intel i860 processor based compute node is equipped with an Inmos T800 transputer that is used to implement message passing. This system allows the i860 to continue processing while the T800 handles the communication. For performance improvement the asynchronous scheme requires that there is sufficient calculation between the message initialisation and the message synchronisation to compensate for the cost of the synchronisation. Performance gains on other platforms have been less successful as a consequence of both hardware and software limitations.

Discussion

The conjugate gradient method presented has a number of advantages that have led to it being regularly used in several codes. Parallelisation of the method does not require algorithmic modification and therefore results in parallel code for which convergence is largely independent of P. The modifications described have the potential to improve parallel performance in some cases. Modification of the algorithm in order to further reduce communication and hence improve parallel performance is possible [DER93] but this can lead to convergence and stability problems. Further improvement in the scalability of the parallel CGM will therefore require communication costs to

be lowered in relation to compute costs. But the communication performance of parallel machines continues to lag behind the developments in processor performance. In particular, inter-processor communication start-up latency is often very high in proportion to the calculation rate. However, the impact of latency and finite bandwidth can be effectively reduced if the parallel system provides truly asynchronous (buffered) message passing.

REFERENCES

[Amd67] Amdahl G. M. (1967) Validity of the single-processor approach to achieving large scale computing capabilities. In *Proc AFIPS*, pages 483–485.

[BBC+94] Barrett R., Berry M., Chan T., Demmel J., Donato J., Dongarra J., Eijkhout V., Pozo R., Romine C., and van der Vorst H. (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Netlib.

[BJG94] Bjørstad P. E., Jr. W. M. C., and Grosse E. (1994) Parallel domain decomposition applied to coupled transport equations. In Keyes D. E. and Xu J. (eds) *Domain Decomposition Methods in Scientific and Engineering Computing*, pages 369–380. American Mathematical Society. Proceedings of the 7th International Conference on Domain Decomposition, October, 1993.

[DER93] D'Azvedo E., Eijkhout V., and Romine C. (January 1993) Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors. Technical Report S-93-185, Oak Ridge National Laboratory and the University of Tenessee. Lapack Working Note 56.

[GL89] Golub G. H. and Loan C. F. V. (1989) *Matrix Computations: Second Edition*. John Hopkins University Press.

[KKS97] Keyes D. E., Kaushik D. K., and Smith B. F. (1997) Prospects for CFD on petaflops systems. Technical report, ICASE NASA Langley.

[LL94] Lai C.-H. and Liddell H. M. (1994) Preconditioned conjugate gradient methods on the DAP. In *The Mathematics of Finite Elements and Applications VI*, pages 145–156. Academic Press.

[MCJ95] McManus K., Cross M., and Johnson S. (1995) Integrated flow and stress using an unstructured mesh on distributed memory parallel systems. In *Proceedings PCFD'94*, pages 287–294.

[RL90] Robinson G. and Lonsdale R. (April 1990) Fluid dynamics in parallel using an unstructured mesh. Internal report, UKAEA.

[She94] Shewchuck J. R. (March 1994) An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Carnegie Mellon University.