

A Systolic Algorithm for the Factorisation of Matrices Arising in the Field of Hydrodynamics

S.-G. Seo¹, M. J. Downie¹, G. E. Hearn¹ and C. Phillips²

¹Department of Marine Technology, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, UK

²Department of Computing Science, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, UK

Abstract. Systolic algorithms often present an attractive parallel programming paradigm. However, the unavailability of specialised hardware for efficient implementation means that such algorithms are often dismissed as being of theoretical interest only. In this paper we report on experience with implementing a systolic algorithm for matrix factorisation and present a modified version that is expected to lead to acceptable performance on a distributed memory multicomputer. The origin of the problem that generates the full complex matrix in the first place is in the field of hydrodynamics.

1. Forming the Linear System of Equations

The efficient, safe and economic design of large floating offshore structures and vessels requires a knowledge of how they respond in an often hostile wave environment [1]. Prediction of the hydrodynamic forces experienced by them and their resulting responses, which occur with six rigid degrees of freedom, involves the use of complex mathematical models leading to the implementation of computationally demanding software. The solution to such problems can be formulated in terms of a velocity potential involving an integral expression that can be thought of as representing a distribution of sources over the wetted surface of the body. In most applications there is no closed solution to the problem and it has to be solved numerically using a discretisation procedure in which the surface of the body is represented by a number of panels, or facets. The accuracy of the solution depends on a number of factors, one of which is the resolution of the discretisation. The solution converges as resolution becomes finer and complicated geometries can require very large numbers of facets to attain an acceptable solution.

In the simplest approach a source is associated with each panel and the interaction of the sources is modelled by a Green function which automatically satisfies relevant 'wave boundary conditions'. The strength of the sources is determined by satisfying a velocity continuity condition over the mean wetted surface of the body. This is

achieved by setting up an influence matrix, A , for the sources based on the Green functions and solving a linear set of algebraic equations in which the unknowns, x , are either the source strengths or the velocity potential values and the right-hand side, b , are related to the appropriate normal velocities at a representative point on each facet. By construction, the diagonal elements of A represent the influence of the sources on themselves. Since the off-diagonal elements are inversely proportional to the distance between any two sources, they tend to be of increasingly smaller magnitude as we move away from the diagonal. Hence if geometric symmetry is not used and the full matrix A is considered, it is guaranteed to be diagonally dominant. For a given wave frequency, each facet has a separate source contributing to the wave potential representing the incoming and diffracted waves, ϕ_0 and ϕ_7 , and one radiation velocity potential for each degree of freedom of the motion, $\phi_i : i=1,2,\dots,6$. When the velocity potentials have been determined, once the source strengths are known, the pressure can be computed at every facet and the resultant forces and moments on the body computed by integrating them over the wetted surface. The forces can then be introduced into the equations of motion and the response of the vessel at the given wave frequency calculated.

The complexity of the mathematical form of the Green functions and the requirement to refine the discretisation of the wetted surfaces within practical offshore analyses, significantly increases the memory and computational load associated with the formulation of the required fluid-structure interactions. Similarly the solution of the very large dense square matrix equations formulated in terms of complex variables requires considerable effort to provide the complex variable solution. In some problems 5,000 panels might be required using constant plane elements or 5,000 nodes using higher order boundary elements, leading to a matrix with 25,000,000 locations. Since double precision arithmetic is required, and the numbers are complex, this will require memory of the order of 4 gigabytes. The number of operations for direct inversion or solution by iteration is large and of the order of n^3 , e.g. 5,000 elements requires $125,000 \times 10^6$ operations. Furthermore, the sea-state for a particular wave environment has a continuous frequency spectrum which can be modelled as the sum of a number of regular waves of different frequencies with random phase angles and amplitudes determined by the nature of the spectrum. Determination of vessel responses in a realistic sea-state requires the solution of the boundary integral problem described above over a range of discrete frequencies sufficiently large to encompass the region in which the wave energy of the irregular seas is concentrated. In view of the size and complexity of such problems, and the importance of being able to treat them, it is essential to develop methods to speed up their formulation and solution times. One possible means of achieving this is through the use of parallel computers [2] [3].

Numerical techniques for solving linear systems of equations neatly divide into direct methods that involve some transformation of A , and iterative methods that do not change A but typically involve a matrix-vector multiplication involving A at each iteration. (Iterative techniques, such as those based on the conjugate gradient method, often employ preconditioning, which requires the formulation of a pseudo-transformation of A in order to improve the convergence characteristics.) Iterative techniques are normally restricted to the case that A is sparse, when the reduction in

operations count and space requirements caused by fill-in merit their use. For our application A is a full, square matrix, and the use of a direct method of solution based on elimination techniques is therefore the most attractive proposition. The method of LU -decomposition has been chosen because in this scheme only one factorisation is required for multiple unknown right-hand side vectors \mathbf{b} . We recall that A is diagonally dominant and hence pivoting, which often limits the performance of parallel algorithms, is not an issue here. It is well known that this factorisation is computationally intensive, involving order n^3 arithmetic operations (multiplications and additions). In contrast the forward- and backward-substitution required to solve the original system once the factorisation has been performed involves an order of magnitude less computation, namely order n^2 , which becomes insignificant as the matrix size increases. Consequently, we limit consideration to the factorisation process only.

Formally, we have that the elements u_{ij} of U are given by

$$u_{rj} = a_{rj} - \sum_{k=1}^{r-1} l_{rk} u_{kj} \quad j = r, \dots, n \quad (1)$$

and l_{ij} of L are given by

$$l_{ir} = \left(a_{ir} - \sum_{k=1}^{r-1} l_{ik} u_{kr} \right) / u_{rr} \quad i = r+1, \dots, n \quad (2)$$

(Doolittle factorisation) leading to an upper-triangular U and a unit lower-triangular L .

2. A Naive Systolic Algorithm Solution

A systolic system can be envisaged as an array of synchronised processing elements (PEs), or cells, which process data in parallel by passing them from cell to cell in a regular rhythmic pattern. Systolic arrays have gained popularity because of their ability to exploit massive parallelism and pipelining to produce high performance computing [4] [5]. Although systolic algorithms support a high degree of concurrency, they are often regarded as being appropriate only for those machines specially built for the particular algorithm in mind. This is because of the inherent high communication/computation ratio.

In a soft-systolic algorithm, the emphasis is on retaining systolic computation as a design principle and mapping the algorithm onto an available (non-systolic) parallel architecture, with inevitable trade-offs in speed and efficiency due to communication and processor overheads incurred in simulating the systolic array structure.

Initially a systolic matrix factorisation routine was written in Encore Parallel Fortran (epf) and tested on an Encore Multimax 520. This machine has seven dual processor cards (a maximum of ten can be accommodated), each of which contains two independent 10 MHz National Semiconductor 32532 32-bit PEs with LSI

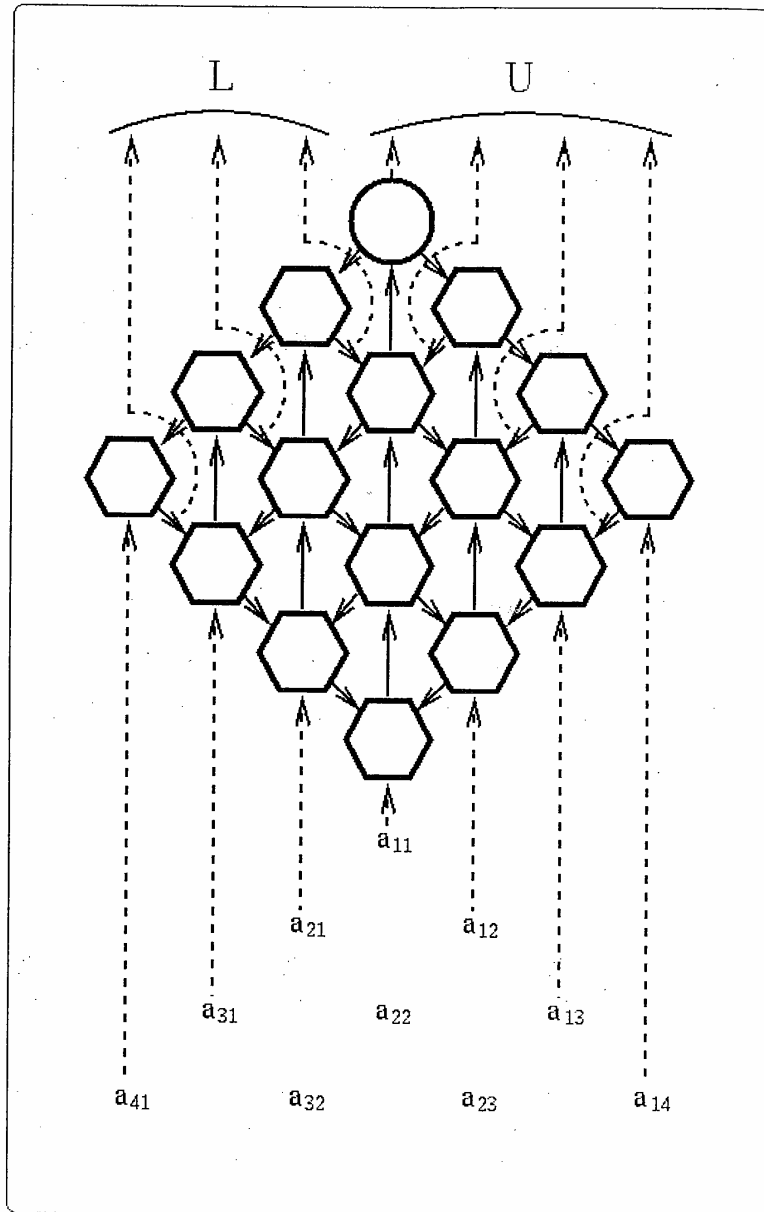


Fig. 1. Data flow for LU factorisation

memory management units and floating-point co-processors. This work was undertaken on a well-established machine platform that was able to provide a mechanism for validating the model of computation, and the software developed from that model, with a view to refining the model for subsequent development on a state-of-the-art distributed memory parallel machine. epf's parallel program paradigm is much simpler to work with than that for message passing on distributed memory architectures and produces a convenient means of providing validation data for subsequent developments. As long as the underlying algorithm is retained this implementation can be used to check the correctness of the equations developed and the validity of the algorithm with a view to an eventual port.

The task of systolic array design may be defined more precisely by investigating the cell types required, the way they are to be connected, and how data moves through the array in order to achieve the desired computational effect. The hexagonal shaped cells employed here are due to Kung and Leiserson [6].

The manner in which the elements of L and U are computed in the array and passed on to the cells that require them is demonstrated in Fig. 1. An example of a 4x4 cell array is shown for the purposes of illustrating the paradigm employed. Of the PEs on the upper-right boundary, the top-most has three roles to perform:

1. Produce the diagonal components of L (which, for a Doolittle-based factorisation, are all unit).
2. Produce the diagonal components of U using elements of A which have filtered through the cells along the diagonal (and been modified, as appropriate).
3. Pass the reciprocal of the diagonal components of U down and left.

The cells on the upper left boundary are responsible for computing the multipliers (the elements of L), having received the appropriate reciprocals of the diagonal elements of U .

The flow of data through the systolic array is shown in Fig. 1. Elements of A flow in an upward direction; elements of L (computed from (1)) flow in a right-and-down direction; and elements of U (computed from (2)) flow in a left-and-down direction. Each cell computes a value every 3 clock ticks, although they start at different times. Note that at each time step each cell responsible for forming an element of L or U calculates one multiplication only in forming a partial summation. Data flowing out correspond to the required elements of the L and U factors of A . Formally, the elements of A are fed into the cells as indicated, although for efficiency the cells are directly assigned the appropriate elements of A .

Table 1 shows the execution times (T_p) of the parallel code with a fixed-size (100*100 double complex) matrix and various numbers of PEs (p). The gradual algorithmic speed-up (S_p), defined as the ratio of the time to execute the program on p processors to the time to execute the same parallel program on a single processor, is clearly seen all the way up to twelve PEs. The (generally) decreasing efficiency (E_p), defined as the ratio of speed-up to the number of PEs times 100, is a consequence of the von Neumann bottleneck. The results show some minor anomalies, but this is not atypical when attempting to obtain accurate timings on a shared resource, with other

comparison to the DMT routine in terms of elapsed time. Nevertheless, the systolic algorithm shows better speedup characteristics than the DMT algorithm, as illustrated by Fig. 2.

If A is an n by n dense matrix then the systolic algorithm implemented on n^2 PEs can compute L and U in $4n$ clock ticks, giving a cell efficiency of 33%. Assuming a hardware system in which the number of PEs is much less than the number of cells, and using an appropriate mapping of cells to PEs, we can improve this position considerably, and we now address this issue.

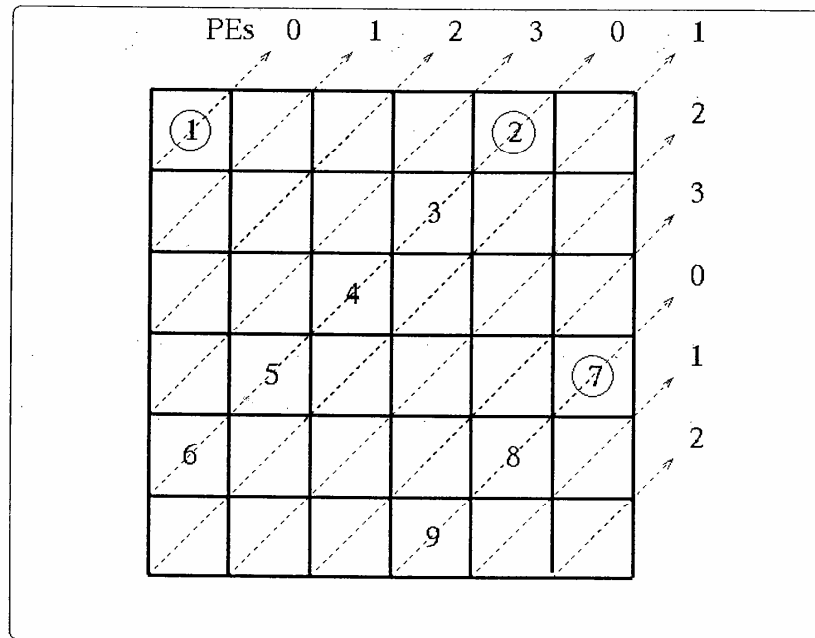


Fig. 3. Allocation of pseudo cells to PEs for a 6x6 matrix.

3. An Improved Systolic Algorithm

As already indicated, the ultimate goal is to produce an efficient systolic matrix factorisation routine for general-purpose distributed parallel systems including clusters of workstations. This is to be achieved by increasing the granularity of the computation within the algorithm and thus reducing the communication/ computation ratio, while balancing the load on each PE to minimise the adverse effect due to enforced synchronisation. Nevertheless, the characteristics peculiar to systolic algorithms should be retained. Thus we aim at

- massive, distributed parallelism
- local communication only
- a synchronous mode of operation

Each PE will need to perform far more complex operations than in the original systolic systems used as the original basis for implementation. There is an inevitable increase in complexity from the organisation of the data handling required at each synchronisation (message-passing) point.

The systolic algorithm can be regarded as a wave front passing through the cells in an upwards direction in Fig. 1. This means that all pseudo cells in a horizontal line, corresponding to a reverse diagonal of A , become active at once. It follows that we allocate the whole of a reverse diagonal to a PE, and distribute the reverse diagonals from the top left to the bottom right in a cyclic manner so as to maintain an even load balance (for a suitably large matrix size). Fig. 2 shows such a distribution for a 6×6 matrix distributed on 4 PEs.

The computation starts at PEO with pseudocell 1. As time increases, so the computation domain over the matrix domain increases, and later shrinks. The shape of the computation domain is initially triangular, to include the first few reverse diagonals. On passing the main reverse diagonal, the computation domain becomes pentagonal, and remains so until the bottom right-hand corner is reached, when it becomes a quadrilateral. Once the computation domain has covered the whole of the domain of pseudo cells it shrinks back to the top left, whilst retaining its quadrilateral shape. The whole process is completed in $3n-2$ timesteps.

A Fortran implementation of the revised systolic algorithm is currently under development using the distributed memory Cray T3D at the Edinburgh Parallel Computer Centre (EPCC), and the Fujitsu AP1000 at Imperial College, London, and MPI [8] [9] for message passing.

In this implementation the elements in each reverse (top-right to bottom-left) diagonal of the matrix are bundled together so that they are dealt with by a single PE and all of the reverse diagonals which are active at a given time step are again grouped together to be passed around as a single message. Preliminary experience with the revised algorithm indicates that it scales well, as shown by the speed up figures for a 400×400 array computed on the Fujitsu machine and illustrated in Fig. 4.

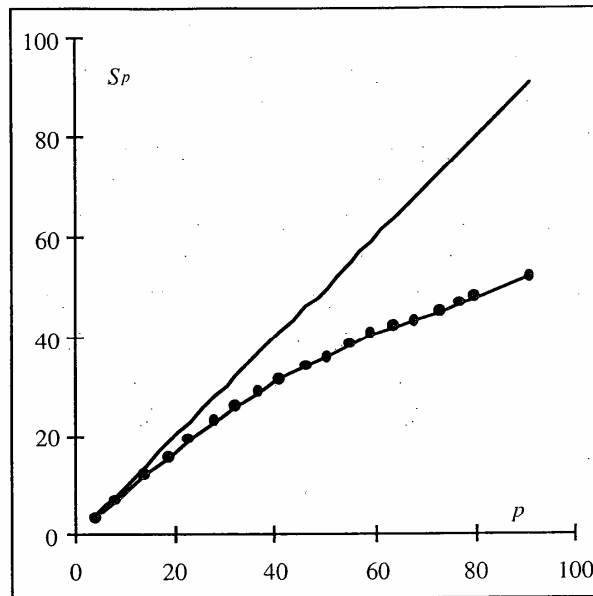


Fig. 4. Speedup on distributed memory machine for 400 by 400 array

4. Conclusions

The algorithm described initially was a precursor to a more generalized one designed with the intention of increasing the granularity of computation and with the solution of large systems of equations in mind. As expected, it performed poorly in terms of elapsed time due to the penalties imposed by an unfavourable balance between processor communication and computation. However, the speedup characteristics compared favourably with those of a more conventional approach and pointed to the potential for the generalised algorithm.

Accordingly, a generalised version of the algorithm has been developed and is currently being tested. The speedup obtained on the distributed memory machine suggest that the approach taken is valid. It remains to benchmark the algorithm against a conventional solver, such as the one available within the public domain software package, ScaLAPACK.

References

1. Hearn, G.E., Yaakob, O., Hills, W.: Seakeeping for design: identification of hydrodynamically optimal hull forms for high speed catamarans. Proc. RINA Int. Symp. High Speed Vessels for Transport and Defence, Paper 4, London (1995), pp15.
2. Hardy, N., Downie, M.J., Bettess, P., Graham, J.M.: The calculation of wave forces on offshore structures using parallel computers., *Int. J. Num. Meth. Engng.* **36** (1993) 1765-1783
3. Hardy, N., Downie, M.J., Bettess, P.: Calculation of fluid loading on offshore structures using parallel distributed memory MIMD computers. Proceedings, Parallel CFD, Paris (1993).
4. Quinton, P., Craig, I.: *Systolic Algorithms & Architectures*. Prentice Hall International (1991)
5. Megson, G. M. (ed.): *Transformational Approaches to Systolic Design*. Chapman and Hall (1994)
6. Kung, H. T. and Leiserson, C. E. *Systolic Arrays for VLSI*, Sparse Matrix Proceedings, Duff, I.S. and Stewart, G.W. (ed.). Society for Industrial and Applied Mathematics (1979) PP.256-282.
7. Applegarth, I., Barbier, C., Bettess, P.: A parallel equation solver for unsymmetric systems of linear equations. *Comput. Sys. Engng.* **4** (1993) 99-115
8. MPI: A Message-Passing Interface Standard. Message Passing Interface Forum (May 1994)
9. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: *MPI: The Complete Reference*. MIT Press (1996)