Parallel Implementation of a Least-Squares Spectral Element Solver for Incompressible Flow Problems

Margreet Nool[1], Michael M. J. Proot[2]

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail:Margreet.Nool@cwi.nl ***
[2] Delft University of Technology, Aerospace Engineering, Section Aerodynamics, Kluyverweg 1, Delft, The Netherlands.
E-mail: m.m.j.proot@lr.tudelft.nl

Abstract. Least-squares spectral element methods are based on two important and successful numerical methods: spectral/*hp* element methods and least-squares finite element methods. Least-squares methods lead to symmetric and positive definite algebraic systems which circumvent the Ladyzhenskaya-Babuška-Brezzi stability condition and consequently allow the use of equal order interpolation polynomials for all variables. In this paper, we present results obtained with a parallel implementation of the least-squares spectral element solver on a distributed memory machine (Cray T3E) and on a virtual shared memory machine (SGI Origin 3800).

1 Introduction

For many engineering flow problems, the least-squares principles offer several theoretical and computational advantages in the algorithmic design and implementation [1, 2, 3, 4] of the corresponding finite element methods, advantages that are not present in standard Galerkin based discretization. In particular, the least-squares formulations lead to symmetric and positive definite algebraic systems [5] which circumvent the Ladyzhenskaya-Babuˇska-Brezzi stability condition irrespective of the underlying partial differential equations. Due to these advantages, least-squares finite element methods are becoming increasingly popular to solve the Stokes [6, 7, 8] and Navier-Stokes equations [9, 10, 5].

Least-squares spectral element methods (LSQSEM) seem very promising since these methods combine the generality of finite element methods with the accuracy of the spectral methods and also the theoretical and computational advantages in the algorithmic design and implementation of the least-squares methods. In [11, 12], the accuracy of a least-squares spectral discretization of the Stokes problem (cast in velocity-vorticity-pressure form) has been reported for

*** Funding for this work was provided by the National Computing Facilities Foundation (NCF), under project numbers NRG-2000.07 and MP-068. Computing time was also provided by HPαC, Centre for High Performance Applied Computing at the Delft University of Technology.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 900–909, 2002.

different boundary conditions. The interested reader is referred to these papers for a sound discussion regarding the least-squares spectral element formulation of the Stokes problem, the gathering procedure and the effect of the boundary conditions on the formulation. The present paper deals with efficient parallel solution strategies to solve the algebraic systems resulting from the least-squares spectral element formulation of the Stokes problem.

Parallelization of the least-squares finite element methods seems to be straightforward by using element-by-element techniques [1, 4]. However, this is not the case with least-squares spectral element methods since two different kinds of distribution of data are required and the conversion is rather complicated. The spectral element structure enables to calculate the local matrices corresponding to each spectral element, simultaneously. Obviously, if the number of available processors is much larger than the number of spectral elements, many processors become idle unless the data of a single spectral element will be computed along several processors. In the present paper, we consider a spectral element, also called a cell, as the smallest computational unit. The parallel solution of the algebraic problem, a large, global sparse system, requires a completely different data distribution.

The present paper is organized in the following way. In Sect. 2, some implementation aspects of least-squares spectral element methods are treated. The program structure and parallel implementation are discussed in Sect. 3. The results of the numerical simulations are discussed in Sect. 4. Conclusions are given in Sect. 5.

2 Implementation aspects of least-squares spectral element methods

The domain is discretized with a mesh of k non-overlapping conforming quadrilateral spectral elements of the same order. As discussed in [11, 12], each quadrilateral spectral element is first mapped on the parent spectral element and then the local systems

$$A_i z_i = f_i, \qquad \text{with } i = 1, \cdots, k \tag{1}$$

are calculated. The matrix A_i represents the least-squares spectral element discretization of the governing equations of spectral element i and the vectors z_i and f_i represent the corresponding local variables and the right-hand function, respectively.

In Fig. 1 an example is given of a domain discretized with a mesh of four spectral elements. Each spectral element contains nine local nodes, numbered from 1 to 9 (small-size digits). In the same figure, also a global numbering (normal-size digits) is shown. First, the internal nodes or variables are numbered $(1, \cdots, 9)$, then the *knowns* $(10, \cdots, 25)$ given by the boundary conditions. Since each local variable corresponds to a global variable, one can establish the local-global mapping operator $gm_{\mathcal{I}}$ for each spectral element. For the given ex-

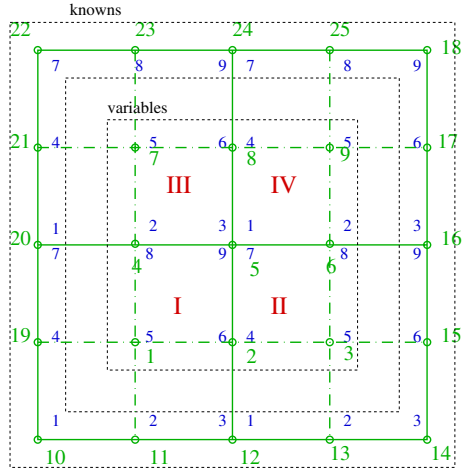

Fig. 1. Example of local and global numbering. The domain has been divided into four cells: I, II,III, IV. Each cell contains 9 nodes, denoted by a ∘.

ample, we have

$$\begin{aligned} gm_{\mathtt{I}} &= [\,10, 11, 12, 19, 1, \;\; 2, 20, \;\; 4, \;\; 5\,], \\ gm_{\mathtt{II}} &= [\,12, 13, 14, \;\; 2, 3, 15, \;\; 5, \;\; 6, 16\,], \\ gm_{\mathtt{III}} &= [\,20, \;\; 4, \;\; 5, 21, 7, \;\; 8, 22, 23, 24\,], \\ gm_{\mathtt{IV}} &= [\;\; 5, \;\; 6, 16, \;\; 8, 9, 17, 24, 25, 18\,]. \end{aligned} \tag{2}$$

The local-global mapping operator $gm_{\mathcal{I}}$ can also be expressed by the sparse *gathering matrix* $\mathcal{G}_i$ which has nonzero entries according to $\mathcal{G}_i(i, gm_{\mathcal{I}}(i)) = 1, \mathcal{I} = \mathtt{I}, \cdots, \mathtt{IV}$. The global assembly of the k local systems (1) can now readily be obtained with:

$$KU = F \Leftrightarrow \left[\sum_{i=1}^{k} \mathcal{G}_i^T A_i \mathcal{G}_i\right] U = \sum_{i=1}^{k} \mathcal{G}_i^T f_i. \tag{3}$$

where the matrix K represents the symmetrical globally gathered matrix of full bandwidth and the vectors U and F represent the global nodes (e.g., variables and knowns) and the global right-hand side function, respectively.

Since the known nodes are numbered last, one can subdivide the vector U into an unknown component U_1 and a known component U_2. Consequently, the matrix K can be factored into submatrices $K_{1,1}$, $K_{1,2}$, $K_{1,2}^T$ and $K_{2,2}$. Also the the right-hand side vector F can be factored into the submatrices F_1 and F_2. Hence, system (3) has the following matrix structure

$$\begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{1,2}^T & K_{2,2} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}, \tag{4}$$

which readily allows "*static condensation*" of the knowns, leading to the following sparse symmetric and positive definite system

$$K_{1,1}U_1 = F_1 - K_{1,2}U_2 \ . \tag{5}$$

System (5) will be solved in parallel with the conjugate gradient method.

3 Program structure and parallel implementation

3.1 Redistribution of data due to renumbering

After we have built up the grid completely and after the calculation of the local systems (1), we have to switch from local numbering to global numbering as discussed in Sect. 2. As a result, we obtain a global CSR-matrix which can easily be distributed long an arbitrary number of processors. Each processor has to send data from one cell to a *few* other processors or possibly to itself, a very unbalanced task due to the chosen numbering. However, if this task is completed, each processor contains a part of the global assembled matrix (3), and the data per processor will be balanced again.

Let us return to the example grid of Fig. 1. If we consider only the internal nodes and investigate the case of four processors, then before redistribution processor p_0 corresponds to cell I, p_1 to cell II and so on. After the rearrangement of the data, p_0 contains the first three rows of matrix K of (3), p_1, p_2 and p_3 each two rows. The distribution is as follows:

$$\begin{array}{lllll} p_0(\text{ cell } \texttt{I}) & \Rightarrow p_0(N_1,N_2), & p_1(N_4,N_5), & & \\ p_1(\text{ cell } \texttt{II}) & \Rightarrow p_0(N_2,N_3), & p_1(N_5), & p_2(N_6), & \\ p_2(\text{ cell } \texttt{III}) & \Rightarrow & p_1(N_4,N_5), & p_2(N_7), & p_3(N_8), \\ p_3(\text{ cell } \texttt{IV}) & \Rightarrow & p_1(N_5), & p_2(N_6), & p_3(N_8,N_9). \end{array} \tag{6}$$

3.2 Parallel Conjugated Gradient Performance

Since system (5) is symmetric and positive definite, the conjugate gradient (CG) method can be applied directly. The performance of this iterative solution strategy for least-squares finite element approximation of flow problems on distributed parallel computers is clearly of relevance to computational fluid dynamics. In this report, we describe results with the simple, easy to parallelize, Jacobi or diagonal preconditioning. At this moment, we test the efficiency of other preconditioning schemes for the incompressible Navier-Stokes problem: block-Jacobi, SSOR, FEM-matrix and Additive Schwarz and their parallel possibilities. The latter seems to be a good candidate.

Having assembled the system locally in parallel, solution by CG iteration involves repeated matrix-vector products, dot products and `DAXPY` operations. More specifically, each iteration involves one matrix-vector product, two dot products, two `DAXPY` and one `DAYPX` operations ($y = y + \alpha x, y = x + \alpha y$). Local dot products are computed in parallel on the processors and the scalar results

are accumulated across the processors using global summation followed by a broadcast. The communication of the dot product will increase logarithmically with increasing number of processors. The matrix-vector products, which clearly require the greatest fraction of the computation, are computed in parallel.

Consider the matrix-vector product

$$Y = \alpha\ A\ X + \beta\ Y, \tag{7}$$

where A is stored in Compressed Row Storage mode. A fast method to parallelize this operation is to divide matrix A and vector Y into equal parts for the sake of a good load balancing. For the matrix A this means that each processor gets the same number of rows m_p, following the next distribution:

$$m_p = m/p, \tag{8}$$

if m, the number of rows of A, is a multiple of the number of processors p. If not, which will be true in most cases, some adjustment of this approximate proces partitioning will be needed and the number of grid points per processor may vary slightly. We assume that each part has a comparable number of nonzero elements. The complete vector X must be available on each processor.

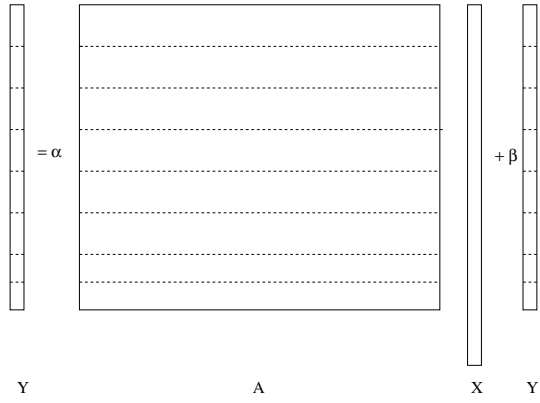

Fig. 2. Parallel distribution of matrix A and vector Y, along 8 processors

The CSR-matrix A of Fig. 2 is defined as

```
TYPE,PUBLIC :: matvec_csr
   REAL(DOUBLE), DIMENSION(:), POINTER   :: FKE
   INTEGER, DIMENSION(:), POINTER        :: JFKE
   INTEGER, DIMENSION(:), POINTER        :: IFKE
   INTEGER                               :: no_rows
END TYPE matvec_csr
```

Then on each processor the matrix A can be declared as:

```
TYPE(matvec_csr)  :: A,
```

where `A%FKE` contains the nonzero values at the processor involved, where the `INTEGER` array `A%JFKE` contains their column numbers and where `A%IFKE(i+1)`-`A%IFKE(i)` denotes the number of nonzeros of row i on that particular processor.

4 Numerical results

In (least-squares) spectral element applications, t wo different kinds of refinement strategies are commonly used: h-refinement and p-refinement. The purpose of the numerical simulations is to check the parallel performance for both refinement strategies. To this end, the least-squares spectral element formulation of the velocity-vorticity-pressure formulation of the Stokes problem is demonstrated by means of the smooth model problem of Gerritsma-Phillips [13] with $v = 1$. This model problem involves an exact periodic solution of the Stokes problem defined on the unit-square ($[0,1] \times [0,1]$). The velocity boundary condition is used for all the numerical simulations. The pressure constant is set at the point $(0,0)$. The $h-$ and $p-$grids used in the present paper correspond to the grids in [11, 12].

4.1 The h- and p-refinement approach and its parallel performance

Six different grids are used to check the parallel performance of the h-refinement. As can be observed in Table 1, the polynomial order of all the spectral elements equals 4, which means that each direction has four Gauss-Legendre-Lobatto(GLL) collocation points, and the number of spectral elements is varied from 4 to 144. For the moment, we consider a cell as the smallest computational unit. Obviously, an increase of the number of cells allows to use more processors, and the parallel efficiency will grow. In case the number of processors is less than the number of cells, one or more processors will compute data of more than one cell.

In the middle column of Tables 1 and 2 the order of the large sparse global system is given together with the number of iterations required to solve this system using CG. The parallel solution of the systems may give a slightly different number of iteration steps. The right column in the Tables lists the L_2 norm of the different components, like the velocity (L_2 norm of $x-$ and $y-$components agree), the vorticity and pressure. Only four different grids have been used to check the parallel performance in case of the p-refinement (see Table 2). Each grid contains four spectral elements. The order of the approximating polynomial varies from 4 to 10 and is the same in all the variables. A growth of the polynomial order in the p-refinement case will increase the number of nodes per cell and so does the amount of computational effort per cell. However, the highest parallel efficiency will be achieved in case the number of cells equals the number

Table 1. The different grids used for the investigation of the $h-$refinements.

Spectral elements	GLL-order	size of global system	# iterations	L_2 norm Velocity	Vorticity	Pressure
2×2	4	259	132	$9.2\ 10^{-4}$	$4.8\ 10^{-2}$	$1.8\ 10^{-2}$
4×4	4	1027	232	$5.0\ 10^{-5}$	$1.6\ 10^{-3}$	$7.1\ 10^{-4}$
6×6	4	2307	326	$5.2\ 10^{-6}$	$2.8\ 10^{-4}$	$6.9\ 10^{-5}$
8×8	4	4099	431	$1.1\ 10^{-6}$	$8.7\ 10^{-5}$	$1.3\ 10^{-5}$
10×10	4	6403	569	$3.2\ 10^{-7}$	$3.5\ 10^{-5}$	$3.6\ 10^{-6}$
12×12	4	9219	707	$1.2\ 10^{-7}$	$1.7\ 10^{-5}$	$1.3\ 10^{-6}$

Table 2. The different grids used for the investigation of the $p-$refinements.

Spectral elements	GLL-order	size of global system	# iterations	L_2 norm Velocity	Vorticity	Pressure
2×2	4	259	132	$9.2\ 10^{-4}$	$4.8\ 10^{-2}$	$1.8\ 10^{-2}$
2×2	6	579	224	$8.7\ 10^{-6}$	$7.5\ 10^{-4}$	$1.9\ 10^{-3}$
2×2	8	1027	305	$6.5\ 10^{-8}$	$7.1\ 10^{-6}$	$1.6\ 10^{-6}$
2×2	10	1603	388	$4.4\ 10^{-10}$	$4.5\ 10^{-8}$	$7.6\ 10^{-9}$

of processors. If the number of processors is larger than the number of cells, processors will become idle and for parallel performance and scalability this result is dramatic.

We remark that four spectral elements and a GLL-order of 8 gives a higher accuracy compared to the grid with 12×12 spectral elements and a GLL-order of 4. Moreover, the systems to solve are much smaller whereas the number of iterations is halved.

4.2 Parallel platforms and implementation

The calculations have been performed on

- Cray T3E system Vermeer (named after the Dutch painter) at HPαC with 128 user PEs interconnected by the fast 3D torus interconnect network with a peak performance of 76.8 Gigaflop/s. Each PE is configured with 128 Mbytes of local memory, providing more than 16 Gbytes of globally addressable distributed memory.
- The SGI Origin 3800 Teras with 1024 500 MHz RI 14000 processors, subdivided into six partitions, two (interactive) 32-CPU partitions and four batch partitions of 64, 128, 256 and 512 CPU's, respectively. The theoretical peak performance is 1 Teraflop/s. The Teras is a CC-NUMA machine, Cache-Coherent, Non Uniform Memory Access. For the user the complete memory is accessible, though as a matter of fact the memory is distributed along all

processors. The memory access is not uniform, because each processor can access its own memory much faster than the memory of other processors.

To get good portable programs which may run on distributed-memory multiprocessors, networks of workstations as well as shared-memory machines we use `MPI`, Message Passing Interface. At this moment, **standard** or blocking communication mode is used: a send call does not return until the message data have been safely stored away so that the sender is free to access and overwrite the send buffer. All routines have been implemented in `FORTRAN 90`.

4.3 Parallel performance and speedups

The grid creation and the calculations of the global systems can be performed completely in parallel and is very fast compared to the solution of the global systems. However, the conversion of the cell distribution to the parallel CSR-format distribution becomes more expensive in case more processors are involved. Table 3 shows wall-clock timings for the Teras of this conversion simulated on a single processor and we do not expect a high parallel speedup for this process that is mainly dominated by communication.

Table 3. Teras: Wall-clock timings in seconds for conversion of cell-wise distribution of grid with 2×2 spectral elements into global matrix in CSR-format, simulated on a single processor.

# Processors	GLL-order			
converted for	4	6	8	10
1	0.03	0.12	0.35	0.86
2	0.04	0.17	0.51	1.25
4	0.07	0.28	1.01	2.40
8	0.12	0.65	2.31	5.30
16	0.25	1.20	5.85	13.25
32	0.69	4.78	13.64	30.67

In Fig. 3, speedups for the solution part are given for grids with different numbers of spectral elements. The speedups, obtained at Teras and Vermeer, are achieved for 2,4,8,16 and 32 processors. The speedup S_p is defined as the quotient of the wall-clock time measured on one processor and the time measured on p processors. Obviously, the speedup on the distributed memory machine Vermeer is much higher than on the virtual shared memory Teras (cf. Fig. 3a and 3b). Since the SGI MPI-implementation on Teras takes into account that the CPUs *share* the memory, we did not expect this behaviour. The disappointing speedup may be dominated by the *slow* communication compared to its high performance. To get an indication of the performance of both machines, the solution times for grid 8×8 on 1 and 32 processors are listed in Table 4.

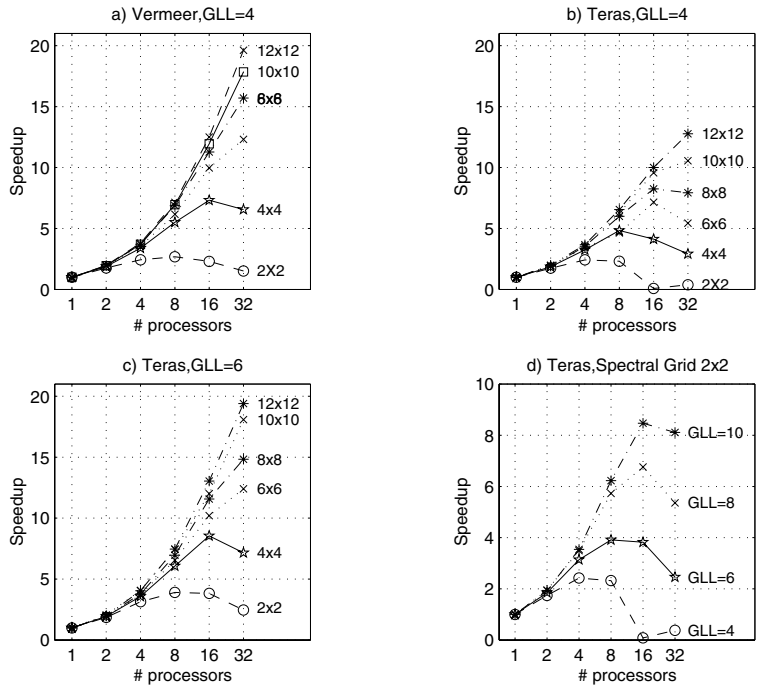

Fig. 3. Speedups achieved on both Vermeer and Teras for different kind of grids.

Table 4. Wall-clock timings in seconds for the solution part obtained for the grid of 12×12 spectral elements and GLL-order=4.

Teras		Vermeer	
$p = 1$	$p = 32$	$p = 1$	$p = 32$
46.9	3.8	314.9	16.0

If we add per spectral element two more GLL-collocation points per direction, the computational efforts increase and the speedup on Teras is nearly twice as much (see Fig. 3c). Finally, Fig. 3d demonstrates that the efficiency of the CG-solution method depends on the GLL-order. Actually, the model problem discussed here appears to be too small for both machines.

5 Conclusions and future plans

The LSQSEM method results in symmetric and positive definite systems of linear equations which can be solved by CG in parallel. At the moment, a Jacobi

preconditioner is used that does not converge very fast. Since the total execution time is dominated by solving the linear systems it is necessary to concentrate on good parallelizable preconditioners for these systems. Obviously, we have to complete the parallelization of the conversion part and to reduce communication time by making use of nonblocking MPI-routines. The execution times listed in Fig. 4 indicate that the parallel implementation is very suitable for large-scale problems arising in scientific computing.

Acknowledgements

The authors wish to thank B. Koren for his careful reading of the paper.

References

1. G. F. Carey and B.-N. Jiang. Element-by-element linear and nonlinear solution schemes. *Communications in Appl. Numer. Meth.*, 2:145–153, 1986.
2. G. F. Carey and B.-N. Jiang. Least-squares finite element method and preconditioned conjugate gradient solution. *Int. J. Numer. Meth. Eng.*, 24:1283–1296, 1987.
3. B.-N. Jiang, T. L. Lin, and L. A. Povinelli. Large-scale computation of incompressible viscous flow by least-squares finite element method. *Comput. Methods Appl. Mech. Eng.*, 114:213–231, 1994.
4. G. F. Carey, Y. Shen, and R. T. Mclay. Parallel conjugate gradient performance for least-squares finite elements and transport problems. *Int. J. Num. Meth. Fluids*, 28:1421–1440, 1998.
5. P. B. Bochev and M. D. Gunzburger. Finite element methods of least–squares type. *SIAM Rev.*, 40(4):789–837, 1998.
6. J. M. Deang and M. D. Gunzburger. Issues related to least-squares finite element methods for the Stokes equations. *SIAM J. Sci. Comput.*, 20(3), 1998.
7. B.-N. Jiang. On the least-squares method. *Comput. Methods Appl. Mech. Eng.*, 152:239–257, 1998.
8. B.-N. Jiang and C. L. Chang. Least-squares finite elements for the Stokes problem. *Comput. Methods Appl. Mech. Eng.*, 78:297–311, 1990.
9. B.-N. Jiang and L. Povinelli. Least-squares finite element method for fluid dynamics. *Comput. Methods Appl. Mech. Eng.*, 81:13–37, 1990.
10. B.-N. Jiang. A least-squares finite element method for incompressible Navier-Stokes problems. *Int. J. Num. Meth. Fluids*, 14:843–859, 1992.
11. M. M. J. Proot and M. I. Gerritsma. A least-squares spectral element formulation for the Stokes problem. Accepted in *SIAM J. Sci. Comput.*, 2002.
12. M. M. J. Proot and M. I. Gerritsma. Least-squares spectral elements applied to the Stokes problem: best of all worlds? Submitted to *SIAM J. Sci. Comput.*
13. M. I. Gerritsma and T. N. Phillips. Discontinuous spectral element approximations for the velocity-pressure-stress formulation of the Stokes problem. *Int. J. Numer. Meth. Eng.*, 43:1401–1419, 1998.