A Parallel, State-of-the-Art, Least-Squares Spectral Element Solver for Incompressible Flow Problems *

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

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Margreet.Nool@cwi.nl
[2] Delft University of Technology, Kluyverweg 1, Delft, The Netherlands.
m.m.j.proot@lr.tudelft.nl

Abstract. The paper deals with the efficient parallelization of least-squares spectral element methods for incompressible flows. The parallelization of this sort of problems requires two different strategies. On the one hand, the spectral element discretization benefits from an element-by-element parallelization strategy. On the other hand, an efficient strategy to solve the large sparse global systems benefits from a row-wise distribution of data. This requires two different kinds of data distributions and the conversion between them is rather complicated. In the present paper, the different strategies together with its conversion are discussed. Moreover, some results obtained on a distributed memory machine (Cray T3E) are presented.

1 Introduction

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. The accuracy of a least-squares spectral element discretization of the Stokes problem (cast in velocity-vorticity-pressure form) has been reported in [5, 6] for different boundary conditions. The present paper deals with the efficient parallelization of this solver.

Parallelization of the least-squares spectral element method (LSQSEM) requires two different kinds of distribution of data and the conversion is rather complicated. The spectral element structure enables one to calculate the local matrices corresponding to each spectral element (also called cells), simultaneously. After the (parallel) calculation of the local systems, we have to switch

* 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.

from a local numbering to a global numbering to complete the gathering procedure. When this has been completed, one obtains a global Compressed Sparse Row (CSR) formatted matrix which can be easily distributed along 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, and the data per processor will be balanced again.

Since the global system is symmetric and positive definite, the robust preconditioned conjugate gradient method (PCG) can be applied directly. In this paper, we describe results obtained with the simple, easy to parallelize, Jacobi or diagonal preconditioning. In the future, we plan to test the efficiency of other preconditioning methods: block-Jacobi, FEM-matrix and Additive Schwarz methods [3, 4, 7]. The results will be discussed in forthcoming work.

The parallelization of the complete algorithm, including the important conversion of a spectral element-by-element distribution to a row-wise CSR-format distribution, and numerical results of large-scale problems arising in scientific computing are also discussed in the present paper.

2 The least-squares spectral element formulation of the Stokes problem

In order to obtain a *bona fide* least-squares formulation, the Stokes problem is *first* transformed into a system of first order partial differential equations by introducing the vorticity as an auxiliary variable. By using the identity

$$\nabla \times \nabla \times \mathbf{u} = -\Delta \mathbf{u} + \nabla(\nabla \cdot \mathbf{u})$$

and by using the incompressibility constraint $\nabla \cdot \mathbf{u} = 0$, the governing equations subsequently read

$$\nabla p + \nu \nabla \times \omega = \mathbf{f} \quad \text{in } \Omega, \tag{1}$$

$$\omega - \nabla \times \mathbf{u} = \mathbf{0} \quad \text{in } \Omega, \tag{2}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega, \tag{3}$$

where, in the particular case of the two-dimensional problem, $\mathbf{u}^{\mathrm{T}} = [u_1, u_2]^{\mathrm{T}}$ represents the velocity vector, p the pressure, ω the vorticity $\mathbf{f}^{\mathrm{T}} = [f_1, f_2]$ the forcing term (if applicable) and ν the kinematic viscosity. For simplicity it is further assumed that the density equals $\rho = 1$. In two dimensions, system (1)-(3), consists of four equations and four unknowns and is uniformly elliptic of order four. The velocity boundary condition ($\mathbf{u}$ given) is used to supplement the governing equations (1)-(3). The linear Stokes operator and its right-hand-side read:

$$\mathcal{L}(U) = F \Longleftrightarrow \begin{bmatrix} 0 & 0 & \nu\frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_1} \\ 0 & 0 & -\nu\frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} \\ \frac{\partial}{\partial x_2} & -\frac{\partial}{\partial x_1} & 1 & 0 \\ \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \omega \\ p \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ 0 \\ 0 \end{bmatrix} \quad \text{in } \Omega. \tag{4}$$

3 General implementation aspects

3.1 Discretization

The domain is discretized with a mesh of $\mathcal{K}$ non-overlapping conforming quadrilateral spectral elements of the same order N. Each quadrilateral spectral element is mapped on the parent spectral element Ω^e by using an iso-parametric mapping to the bi-unitsquare $[-1,1] \times [-1,1]$ with local coordinates ξ_1 and ξ_2. In the parent element all variables, located at the Gauss-Legendre-Lobatto collocation (GLL) points, can be approximated by the same Lagrangian interpolant, since the least-squares formulation is not constrained by the Ladyzhenskaya-Babuška-Brezzi stability condition. Most spectral element methods are based on the GLL numerical integration for reason of accuracy. For the two-dimensional Stokes problem, the discrete spectral element approximation yields

$$\mathbf{U}^h = \sum_{j=0}^{N_2} \sum_{i=0}^{N_1} h_i\left(\xi_1\right) h_j\left(\xi_2\right) \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{\omega} \\ \hat{p} \end{bmatrix}_{i,j}, \tag{5}$$

where $h_i\left(\xi_1\right)$ with $0 \leq i \leq N_1$ and $h_j\left(\xi_2\right)$ with $0 \leq j \leq N_2$ represent the Lagrange interpolants in the ξ_1 and ξ_2-direction through the GLL points, respectively. The vector $[\hat{u}_1, \hat{u}_2, \hat{\omega}, \hat{p}]^{\mathrm{T}}$ is the vector of unknown coefficients, evaluated at the GLL collocation point. Hence, each spectral element gives rise to a local system of the form:

$$A_i \; z_i = f_i, \qquad \text{with } i = 1, \cdots, \mathcal{K} \tag{6}$$

where the matrices A_i and right-hand side vectors f_i are given by

$$A_i = \int_{\Omega_e} \left[\mathcal{L}\left(\psi_{0,0}\right), \cdots, \mathcal{L}\left(\psi_{N,N}\right)\right]^T \left[\mathcal{L}\left(\psi_{0,0}\right), \cdots, \mathcal{L}\left(\psi_{N,N}\right)\right] \; d\Omega, \tag{7}$$

and

$$f_i = \int_{\Omega_e} \left[\mathcal{L}\left(\psi_{0,0}\right), \cdots, \mathcal{L}\left(\psi_{N,N}\right)\right]^T \; \mathcal{F} \; d\Omega, \tag{8}$$

respectively and where $\psi_{i,j} = h_i\left(\xi_1\right) h_j\left(\xi_2\right)$.

3.2 Local numbering versus global numbering

Due to the continuity requirements between the C^0-spectral elements, some of the variables z_i corresponding to internal boundary will belong to more than one local system which necessitates the introduction of a global numbering.

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

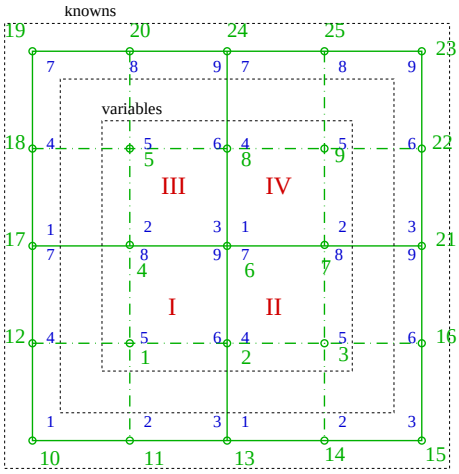

Fig. 1. Example of local and global numbering. The domain has been divided into four cells: I, II,III, IV. Each cell contains 9 nodes, denoted by a ∘.

(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 for each spectral element. For the given example, we have

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

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 $\mathcal{K}$ local systems (6) can now readily be obtained with:

$$KU = F \Leftrightarrow \left[\sum_{i=1}^{\mathcal{K}} \mathcal{G}_i^T A_i \mathcal{G}_i\right] U = \sum_{i=1}^{\mathcal{K}} \mathcal{G}_i^T f_i, \tag{10}$$

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 (10) has the following matrix structure

$$\begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{1,2}^T & K_{2,2} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}, \tag{11}$$

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

$$K_{1,1}U_1 = F_1 - K_{1,2}U_2 \ , \tag{12}$$

which can be solved efficiently in parallel with the Jacobi preconditioned conjugate gradient method [2].

The results in [2] revealed that it is necessary to construct the global system (12) in parallel to obtain a good scalable solver. To this end, the matrices A_i, the vectors f_i, the local-global mapping operator $gm_{\mathcal{I}}$ and the gathering procedure, must be performed in parallel.

3.3 Investigation of the grid

In order to construct the matrices A_i and vectors f_i, one only needs the coordinates of the corners of the cells and its GLL orders in ξ_1- and ξ_2 direction, begin N_1 and N_2, respectively. For the calculation of the local-global mapping operator $gm_{\mathcal{I}}$, detailed grid information regarding the neighbouring cells is required to obtain the global numbering in parallel. Hence, the coordinates of the corners, the neighbouring cells and the GLL order together with a unique global number (see Section 3.4) are the minimal information needed to construct the global system (12) in parallel. These data will be collected on one processor when the grid is investigated and broadcasted from the root processor to $N_c - 1$ processors.

This information can be collected the following way. The order of the spectral elements $N = N_1 = N_2$ is an input variable. When the grid file is read, the nodes, the nodal coordinates and the corners of the spectral element mesh become available. The neighbors of the spectral elements can be obtained by investigating the mesh. This can be done the following way. The corners of a spectral element are given in fixed order, see Fig. 2. In case two elements share the same node, their position with respect to each other is known. E.g., if node 0 of element S_5 equals node 3 of element S_2 then S_2 must be the *South*-neighbor of S_5. If node 2 of S_5 equals node 3 of S_6 then S_6 will be the *East*-neighbor of S_5, and so on. If a spectral element has less than four neighbors it must be a boundary element.

For the parallel global numbering of the nodes, it is necessary to assign the global GLL collocation points to one particular cell. For the inner nodes this is trivial, but the situation is more complicated for the nodes located at the boundaries. Indeed, internal boundaries are shared by at least two spectral elements and internal nodes by at most four spectral elements. Therefore, we must define to which spectral element the corners and GLL collocation points belong. There are several possibilities, with the same complexity, and we choose one of them:

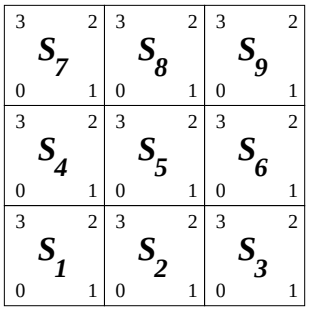

Fig. 2. Ordering of corners, The corners are numbered in fixed order, such that it is easy to recognize which spectral elements are neighbors

- a spectral element consists of all its internal GLL-points, the *lower-left* corner, the collocation points at the *South* and *West* boundary, without the corner nodes.
- if a spectral element has an external boundary, the set is extended with the boundary points and possibly by corners that do not belong to other spectral elements.

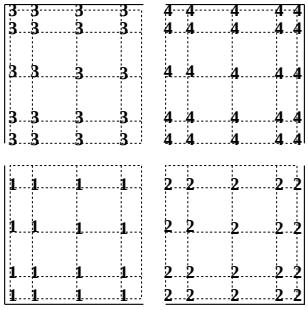

Fig. 3. An example of a grid discretized into four spectral elements. The nodes numbered 1 belong to spectral element 1, those numbered 2 belong to spectral element 2, and so on. The solid line corresponds to an external boundary

Fig. 3 displays a mesh of four spectral elements and the numbers 1 to 4 elucidate to which spectral element the nodes belong.

Communication is required to obtain the missing values on the internal (*North* and *East* interfaces: first boundary data are sent from *North* to *South*. Subsequently, boundary data are sent from *East* to *West*. After these steps, also the right-upper corner has got its correct component values.

3.4 Unique global numbering

To obtain a unique global numbering, first the internal (boundary) points are numbered which belong to a single spectral element: this local numbering starts with 1. Then we know exactly how much *variables* N_{var}^k are present per element and transfer this information along the processors, such that we can compute M_i

$$M_i = \sum_{k=1}^{i-1} N_{var}^k, \tag{13}$$

being the sum of the number of variables on preceding spectral elements $k, k < i$. The value M_i will be added to the local number. The *knowns* are numbered analogously. The numbering starts with $M_{N_c} + N_{\mathcal{K}} + 1$. Hence, one obtains the global node numbering discussed in Section 3.2.

3.5 The construction of matrix K

The parallel global assembly of the $\mathcal{K}$ local systems can now be performed resulting in the sparse symmetric positive definite global matrix K. Since the solution of equation (12) is by far the most time consuming part, it must be optimally performed. Therefore, K will be distributed along the processors into balanced parts of rows. The sparsity of K desires a CSR-formatted storage approach. The symmetry is not exploited: all nonzero elements of K are stored. However, the CG method can only be applied successfully if K is symmetric and positive definite.

The conversion of the local systems A_i into the global system K is complicated and requires communication. Beforehand, it is difficult to say which processors will communicate with each other and how long the messages will be. The global number determines to which processor the data will be sent. Therefore, a type `gambit` has been defined which couples the global number with the processor number and the row number on that particular processor:

```
TYPE :: gambit
   INTEGER  :: globalnr
   INTEGER  :: procnr
   INTEGER  :: rownr
END TYPE
```

Let N_{var} denote the number of variables. Because only the first N_{var} rows of matrix K are desired, the value `procnr` is not important when the global number will be larger than N_{var}: to avoid confusion a negative value is added to `procnr`. For each nonzero element of A_i it is then easy to determine whether it contributes to $K_{1,1}$, to $K_{1,2}$, or whether it can be neglected (row number $> N_{var}$). In the worst case, a single cell contains data which must be sent to all available processors.

All nonzero elements of A_i are considered and on account of its global number it is known to which processor it has to be sent. Before such an element

and corresponding information about column number in K is sent, all data are gathered and put into one message intended for a particular processor. Three different cases can occur:

1. The message must be sent to the processor on which it is already present. In that case, a send instruction is superfluous and the data can be processed immediately, or after the other messages have been sent.
2. The message is empty: no communication will take place, except that a message will be sent to the receiving processor that no contribution of the send processor can be expected.
3. The message is not empty and the send and receive processors are unequal. In that case communication is necessary.

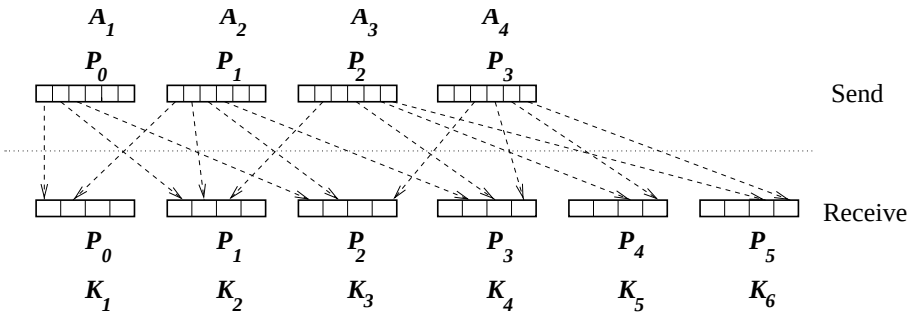

Fig. 4. Example of mapping of local matrices A_i into the global matrix K, partitioned into matrix parts K_j of consecutive rows

The order in which the messages will be received is not known, and also not important. If nonzero elements are already present (see case 1 above) the first *very sparse* matrix L_i can be built up. This matrix L_j contains the nonzero values of an A_i descendent of a spectral element stored in CSR-format. Contributions of other processors will be added to L_j - i.e. each step the sum of two sparse matrices, stored in CSR-format, is calculated with a SPARSKIT [9] subroutine. In case no data are present, e.g., the rank of the processor is larger than the number of cells, the first data received will be used to build up the basis matrix L_j. In the worst case, to create K_j, data from N_c spectral elements are needed. In practice, this number will be less. We do not investigate to find for all (N_c, p) combinations a favorite global numbering.

We observe that two steps are required to construct the matrix K of (11) completely: one to construct $K_{1,1}$ and one to construct $K_{1,2}$ of equation (12). The distribution of $K_{1,1}$ and $K_{1,2}$ along the processors is similar.

4 Parallel platform and implementation tools

The calculations have been performed on

- Cray T3E system Vermeer (named after the Dutch painter) at HPαC with 128 user processing elements (PE's) interconnected by the fast 3D torus network with a peak performance of 76.8 Gigaflop/s. The T3E uses the DEC Alpha 21164 for its computational tasks. Each node in the system contains one PE which in turn contains a CPU, memory, and a communication engine that takes care of communication between PE's. The bandwidth between nodes is quite high: 325 MB/s, bi-directional. Each PE is configured with 128 Mbytes of local memory, providing more than 16 Gbytes of globally addressable distributed memory.

For more information about the Cray T3E used, we refer to [10].

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. **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. Besides the standard communication mode, it appears to be necessary to use the buffered-mode send operation `MPI_BSEND` (see e.g., [8]). This operation can be started whether or not a matching receive has been posted. Its completion does not depend on the occurrence of a matching receive. In this way a deadlock situation, in which all processes are blocked is prevented. The execution of the send-receive process demonstrated in Fig. 4 profits of the buffered-mode send operation.

All routines have been implemented in `FORTRAN 90`, making frequently use of dynamic allocation of memory and derived types. As described in Sect. 3.5 the conversion of the cell-wise distribution into the row-wise CSR-format storage is difficult to implement and asks for a high level of flexibility. On fore hand it is not clear which processing elements will communicate and how much data will actually transferred. An upper bound for the number of processors which may need data of cell i can be N_c, the number of cells involved1s, whereas an upperbound of data from cell i to PE j can be the whole contents of that particular cell i. Apparently, multiplication of these two upperbounds leads to a storage demand per processor which grows with the number of processors used in the execution. The solution can be found in the usage of data structures with pointers to variable array sizes.

5 Numerical results

5.1 The h- and p-refinement approach and its accuracy

In (least-squares) spectral element applications, two different kinds of refinement strategies are commonly used: h-refinement and p-refinement. The purpose of the

numerical simulations is to check the accuracy 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 [1] 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 $(0,0)$.

Six different grids are used to check the accuracy 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. In the middle column of Tables 1 and 2 the order of the large sparse global

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	$4.4\ 10^{-10}$	$4.5\ 10^{-8}$	$7.6\ 10^{-9}$

system is given together with the number of iterations required to solve this system using CG. 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 accuracy 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 likewise the amount of computational effort per cell.

5.2 The parallel performance

The least-squares spectral element solver (LSQSEM) code allows for the following situations:

$N_c \geq p$: At least several processors contain more than one spectral element. It is not required that p is a true divisor of N_c, or put it another way, the number of spectral elements per PE may differ.

$N_c < p$: A less plausible possibility. As we may conclude from Tables 1 and 2 that in order to increase the accuracy of the solution it is to preferred to enlarge the GLL order rather than to add more spectral elements. The latter gives rise to much larger global systems and accordingly more execution time. Indeed, during the construction of the local systems $p - N_c$ processors are idle, but for the same accuracy, the wall-clock time can be less. We remark, that MPI gives the opportunity to define an intracommunicator, which is used for communicating within a group, for instance within the group of PE's which correspond to a spectral element.

In the code we may consider several phases in the parallelization. The phases described correspond to the parts shown in Table 3:

- The decomposition of the domain, or the investigation of the grid (cf. Sect. 3.3). In this phase, information is read from file: the number of spectral elements, the coordinates of the corners of the spectral elements, and of each spectral element its corners are listed on file. Further the position of the spectral elements with respect to its neighbors is derived. This phase will be performed on a single processor.
- the second phase called `GAMBIT`, performs the computation on the internal GLL nodes and the boundary points. Also the local-global mapping (cf. equation (10) is calculated. Communication is required for internal boundary points.
- The third phase computes the local systems and the right hand side values. This part is the most time-consuming part of the calculations on spectral elements and as can be conclude from Table 3, it is well scalable.
- The fourth and fifth phases consider the conversion of the local systems into the global systems. In `SUM` the contribution of the local systems to the construction of matrix K is gathered. `CSR` performs the actual construction of $K_{1,1}$ and $K_{1,2}$ of equation (11) by means of computing the sum of sparse matrices. Also the communication as described by Fig. 4 is included in `CSR`.
- In the last phase of the program PCG is used to solve the global system. In [2] it is described how the matrix-vector product in PCG are computed in parallel and gives pictures of speed-ups of this part of the code achieved

on the TERAS, an SGI Origin 3800 platform with 1024 processors, located at SARA, Amsterdam and the Vermeer, the platform used for the results of the present paper.

The last column of Table 3 gives the wall-clock time for a complete run. We emphasize that the conversion part may not be neglected, especially the man power that was needed to give the code the flexibility for allowing an arbitrary number of spectral elements with respect to the number of processors involved, but for 32 processors it takes about 1.2 % of the total wall clock time.

Table 3. Wall-clock timings in seconds for the different parts in the parallel LSQSEM code obtained for the grid of 12×12 spectral elements and GLL order=6

p	Spectral elements			Conversion		PCG	Execution
	read	`GAMBIT`	Stokes	`SUM`	`CSR`	time	time
4	0.19	0.38	118.95	3.01	21.94	584.57	729.19
8	0.20	0.30	59.44	1.56	7.41	308.55	377.65
16	0.18	0.23	29.70	0.85	2.65	176.34	210.15
32	0.19	0.21	16.54	0.59	1.11	121.67	140.77

Finally, Fig. 5 shows the overall performance of the code on $p = 4, 8, 12, 16, 24, 32$ processors. Here we do not show speedup pictures, because it is not possible to run the largest problem on a single processor. We remark, that not in all cases the number of cells is a multiple of p, e.g., the 10×10 grid has only $p = 4$ as a true divisor. The speedup obtained when eight times more processors are used, (viz. from $p = 4$ to $p = 32$) is more than 5.2 and for the 8×8 is nearly 7.

6 Conclusions and outlook

Least-squares spectral element methods result in symmetric and positive definite systems of linear equations which can be efficiently solved in parallel by PCG. The parallelization of this sort of problems requires two different strategies. Indeed, the spectral element discretization benefits from an element-by-element parallelization strategy whereas an efficient strategy to solve the large sparse global systems benefits from a row-wise distribution of data.

The numerical results, obtained with a simple model problem, confirm the good parallel properties of the element-by-element parallelization strategy. The combination of this strategy with the parallel JCG solver resulted in a good parallizable code to solve incompressible flow problems. In the future, more effective preconditioning methods for least-squares spectral element methods will be developed.

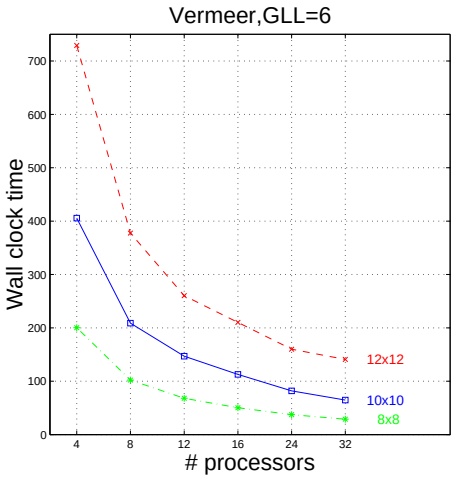

Fig. 5. Wall-clock timings achieved for a grid decomposed into 8×8, 10×10 and 12×12 spectral elements

References

1. 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.
2. M. Nool and M. M. J. Proot. Parallel implementation of a least-squares spectral element solver for incompressible flow problems. Submitted to *J. Supercomput.*, 2002.
3. L. F. Pavarino and O. B. Widlund. Iterative substructuring methods for spectral element discretizations of elliptic systems. I: compressible linear elasticity. *SIAM J. NUMER. ANAL.*, 37(2):353–374, 1999.
4. L. F. Pavarino and O. B. Widlund. Iterative substructuring methods for spectral element discretizations of elliptic systems. II: Mixed methods for linear elasticity and Stokes flow. *SIAM J. NUMER. ANAL.*, 37(2):375–402, 1999.
5. M. M. J. Proot and M. I. Gerritsma. A least-squares spectral element formulation for the Stokes problem. *J. Sci. Comp.*, 17(1-3):311–322, 2002.
6. M. M. J. Proot and M. I. Gerritsma. Least-squares spectral elements applied to the Stokes problem. *submitted*.
7. A. Quateroni and A. Valli. *Domain Decomposition Methods for Partial Differential equations*. Oxford University Press, 1999.
8. Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, Massachusetts, 1996.
9. Yousef Saad. SPARSKIT: a basic tool-kit for sparse matrix computations (Version 2) `http://www.cs.umn.edu/research/arpa/SPARSKIT/sparskit.html`
10. Aad J. van der Steen. Overview of recent supercomputers, Issue 2001 *National Computing Facilities Foundation*, Den Haag, 2001.