Parallelization of Spectral Element Methods

S. Airiau[1], M. Azaïez[2], F. Ben Belgacem[3], and R. Guivarch[1]

[1] LIMA-IRIT, UMR CNRS 5505,
2 rue Charles Camichel, 31071 Toulouse Cedex 7, France
Ronan.Guivarch@enseeiht.fr
http://www.enseeiht.fr/lima
[2] IMFT, UMR CNRS 5502,
118, route de Narbonne, 31062 Toulouse Cedex 4, France
[3] MIPS, UMR CNRS 5640,
118, route de Narbonne, 31062 Toulouse Cedex 4, France

Abstract. Spectral element methods allow for effective implementation of numerical techniques for partial differential equations on parallel architectures.
We present two implementations of the parallel algorithm where the communications are performed using MPI. In the first implementation, each processor deals with one element. It leads to a natural parallelization. In the second implementation certain number of spectral elements are allocated to each processor.
In this article, we describe how communications are implemented and present results and performance of the code on two architectures: a PC-Cluster and an IBM-SP3.

1 Introduction

Spectral element methods [2] allow for effective implementation of numerical techniques for partial differential equations on parallel architectures.

The method is based on the assumption that the given computational domain, say Ω, is partitioned into non overlap subdomains Ω_i, $i \in \{1, \ldots, N\}$. Next, the original problem can be reformulated upon each subdomain Ω_i yielding a family of subproblems of reduced size which are coupled each others through the values of the unknown solution at subdomain interfaces. Our approach is to interpret the domain decomposition as iterative procedures for an interface equation which is associated with the given differential problem, here the Laplacian one. This interface problem is handled by the Steklov-Poincaré operator (see [1] and [3]).

Two steps are considered for our algorithm: in the first one a family of local and independent problems is solved and the second one is to complete the solution by solving the interface problem. Sections 2 and 3 present the method and the algorithm to solve the two-dimensional Poisson problem.

We describe in section 4 two implementations of the parallel algorithm where the communications are performed using MPI. In the first implementation, each processor deals with one element. It leads to a maximal parallelism. The main disadvantage of this implementation is the ratio between the computations and the communications. In a second implementation certain number of spectral elements are allocated to each processor. This implementation permits flexibility in the ratio computation/communication ; various granularities and elements repartitions can be explored.

Section 5 presents results and performance of the code on two architectures: an IBM-SP3[4] and a PC Cluster[5].

2 Presentation of spectral element methods

This study is limited to Ω, a square domain of the plan. The boundary of Ω is designed by $\partial\Omega$. This domain is splitted in n square elements. Each element is discretized by $N_p * N_p$ points.

We want to solve the Poisson problem on Ω:

$$\begin{cases} -\Delta u = f \text{ for } x \in \Omega \\ \quad u = 0 \text{ for } x \in \partial\Omega \end{cases} \tag{1}$$

Decomposition of Ω into n non overlap elements Ω_i leads to interior egdes between elements and we call γ this interface.

We split the initial problem in two subproblems. The first one is the solution of local Poisson problem on each element; the second one concerns the interface to join the solution between elements. This second problem exibits a new unknown function $\lambda : \gamma \mapsto \mathbb{R}$. This function is the solution of the Poisson problem on the interface.

This process leads to decompose the unknown function u in two functions: for each element i, u_i the restriction of u on i is splitted in $\overline{u_i}$, which only depends on the element i, and in $\widetilde{u_i}(\lambda)$, which depends on λ, and so on the neighboring elements. So we have two sets of subproblems (2) and (3) to solve.

The n subproblems on each element:

$$\begin{cases} -\Delta\overline{u_i} = f_i \text{ for } x \in \Omega_i \\ \quad \overline{u_i} = 0 \text{ for } x \in \partial\Omega_i \end{cases} \tag{2}$$

Each subproblem i of (2) doesn't depend on others subproblems, so all subproblems can be solved in parallel.

The second subproblems are harmonic problems:

$$\begin{cases} -\Delta\widetilde{u_i}(\lambda) = 0 \text{ for } x \in \Omega_i \\ \quad \widetilde{u_i}(\lambda) = \lambda \text{ for } x \in \gamma \\ \quad \widetilde{u_i}(\lambda) = 0 \text{ for } x \in \partial\Omega_i/\gamma \end{cases} \tag{3}$$

[4] We would like to thank IDRIS(Institut du Développement et de Ressources en Informatique Scientifique) for computation hours allocated to our project
[5] and INPT for the credits which allowed the cluster acquisition

To solve these subproblems, we have to know the function $\lambda : \gamma \to \mathbb{R}$, continuous on γ which verifies:

$$\begin{cases} -\Delta u_i(\lambda) = f \text{ for } x \in \Omega_i \\ u_i(\lambda) = \lambda \text{ for } x \in \gamma \\ u_i(\lambda) = 0 \text{ for } x \in \partial\Omega_i/\gamma \end{cases} \tag{4}$$

and on each edge between Ω_i and Ω_j,

$$\frac{\partial u_i(\lambda)}{\partial n} = \frac{\partial u_j(\lambda)}{\partial n} \tag{5}$$

After a function decomposition, we obtain:

$$\frac{\partial \widetilde{u_i}(\lambda)}{\partial n_i} - \frac{\partial \widetilde{u_j}(\lambda)}{\partial n_j} = \frac{\partial \overline{u_j}}{\partial n_j} - \frac{\partial \overline{u_i}}{\partial n_i} \tag{6}$$

A discretization with spectral elements, leads to a linear system

$$S\lambda = b \tag{7}$$

where S is the Schur complement matrix. Once λ is computed, we have to solve a Poisson problem on each element. For the same reason than previously, these problems can be solved in parallel.

3 Algorithm for the solution of problem on the interface

The Schur complement matrix is a symetric matrix and we use Conjugate Gradient method. The parallel algorithm to solve problem (7) is presented in Fig. 1.

$\beta = 0$; $\Lambda = 0$; $\rho_0 = (r_0, r_0)$;	Initializations
While the residual is greater than epsilon do	
Solve $\begin{cases} -\Delta\tilde{u}_i(\Lambda) & = & 0 & \text{for } x \in \Omega_i \\ \tilde{u}_i(\Lambda) & = & \Lambda & \text{for } x \in \gamma \\ \tilde{u}_i(\Lambda) & = & 0 & \text{for } x \in \partial\Omega_i/\gamma \end{cases}$	Local solutions
$v = S\Lambda$;	Steklov-Poincaré operator (we need contributions of each element)
$ps_1 = (\Lambda, v)$; $ps_2 = (r_0, r_0)$;	Global computation
$\alpha = \dfrac{ps_2}{ps_1}$; $\lambda = \lambda + \alpha\Lambda$; $r_1 = r_0 - \alpha v$;	Local computation
$\rho_1 = (r_1, r_1)$	Global computation
$\beta = \dfrac{\rho_1}{\rho_0}$; $r_0 = r_1$; $\Lambda = r_0 + \beta\Lambda$	Local computation
End do	

Fig. 1. Parallel algorithm

4 Parallelization and Implementations

4.1 First implementation: one spectral element by processor

In the spectral element method, we solve a local problem on each element. In the partitioning phase, a natural way to distribute the work the solver is to give an element to each processor. As a consequence, all local solutions will be performed in a single processor. An inherent disadvantage of this approach is that we have to duplicate values of the interior edges.

Because we place an element on each processor, we use indifferently the terms element and processor in this subsection.

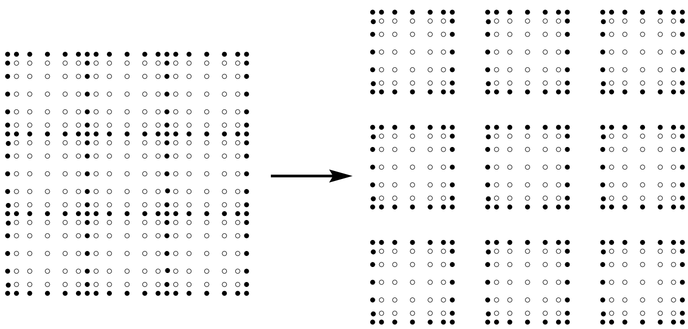

Fig. 2. *9-element domain on 9 processors*

Looking at the algorithm 1, two different tasks need communications between processors: the computation of the image of a vector by Steklov-Poincaré operator and the computation of three dot products.

In the first task, the computation for a point requires values hold on the neighbor points. For interior points, neighbor points belong to the same element. For points on the edges, we need the contribution of several elements. This operation, called gathering, can be separated in two cases: for points of vertices, four elements are concerned, while for the others points, only two elements interfere.

For the second task, the computation of the dot product needs all the elements; this operation can be carried using a global operation.

The gathering operation For each edge of the interface, we have to make the sum of the contribution of each element. A difficulty arises for interior vertices; for these points four elements are concerned. We must be careful with the order of communications between elements: a deadlock could easily occur or we could not take account of all contributions.

In order to minimize the number of communications, we do not choose to separate vertices from the others points of the edges. The solution is to decompose the gathering in two steps: the first one treats vertical egdes and the second horizontal ones.

For example, in the first step, each processor sends the contribution of its right edge and receives the contibution for its left edge, then send data of its left edge and receives data for its right edge.

Using this process, a copy of the intermediate sum in proper vectors between the two steps, and, at the end, a summation of the contribution of all elements, the vertices are treated like others points.

To avoid deadlocks, we choose to asynchronous communications offered by MPI: MPI_ISEND, MPI_IRECEIVE, MPI_TEST and MPI_WAIT. At the beginning of a step, an element sends contributions on its vertical (or horizontal) edges and asks for the receipt of contributions of its neighbors.

The dot product operation For the dot product on the complete domain, communications are different because they involve all the elements. We use the reduction operations offered by MPI which make this operation very easy to implement.

4.2 Second implementation: several spectral elements by processor

It seemed natural to associate an element to each processor. This choice, although it permits a maximal parallelism, has some disadvantages: when computing with a great number of elements, we may not have all the processors we wanted. Furthermore, if the size of an element is too small, communication will take more time than computation.

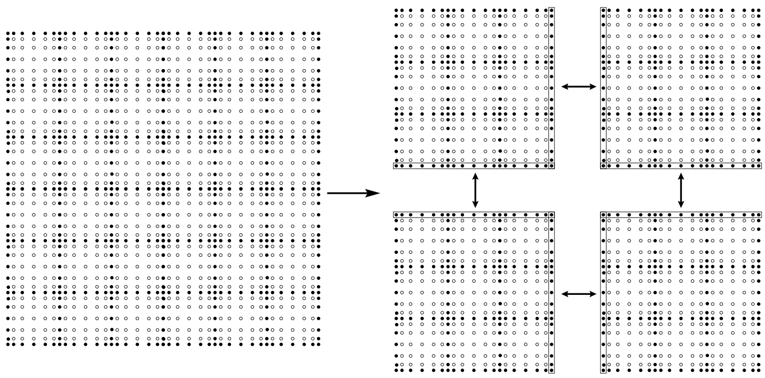

Fig. 3. *36 elements on 4 processeurs*

In this second implementation, we choose to associate several elements to a processor (see Fig. 3). Increasing the granularity permits to improve the computation communication ratio. The code is more flexible: we can perform some tests with a great number of elements and for a fixed number of elements we can choose different repartitions (see Table 1).

Table 1. Repartitions of 64 × 64 elements

Number of processors	Number of elements on a processor in X-direction	Number of elements on a processor in Y-direction
1	64	64
2	64	32
4	32	32
4	64	16
8	32	16
8	64	8
16	16	16
16	32	8
16	64	4
32	16	8
32	32	4
32	62	2

Concerning the gathering operation, we do not have to communicate data of edges inside the domain treated by a processor. Communications only concern the edges of the boundary. Instead treating communications at the level of an element, we can gather in an interface, all edges to be sent by a processor to one of its neighbors (see Fig. 4). We then reduce the number of messages (messages are bigger) and thus reduce communication time.

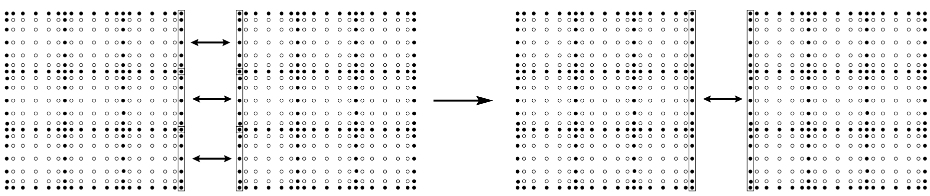

Fig. 4. *Communications* between two processors: a message by edge against a message by interface

Nothing changes for the dot product operation; we still use reduction operations.

There is another advantage of this repartition regarding asynchronous communication. A processor can start its communications and overlay them with the computation of the sum on the interior edges. To sum contributions on the interface, it waits the end of communications and hopes that computations have recovered communications. This advantage will be shown in next section with numerical experiments.

5 Numerical Experiments

Numerical experiments were carried on two architectures: an IBM-SP3 and a cluster of 8 biprocessors PC. We only show performance of the second implementation which permits several elements on each processor.

5.1 Specifications of the two architectures

IBM-SP3 Brodie of IDRIS Each Node of the SP3 has 16 Power3 processors at 375 MHz with 2 Gbytes of shared memory. Inter-node communications are handled with the IP protocol.

PC-Cluster Tarasque of INPT–ENSEEIHT Each node of the cluster has 2 Pentium III processors at 800 MHz with 265 Mbyte of shared memory. The network is a Commute Ethernet. The operating system is Linux (SuSE 7.2 distribution) and we use a free implementation of MPI (LAM 6.5.1). The software has been compiled with GNU compilers.

5.2 Description of the numerical tests

The problem to solve is a Poisson problem. The difficulty depends on the nature of the second member. We choose four kinds of functions:

- sinus product: $f : (x, y) \mapsto 4\pi^2 \sin(\pi x) sin(\pi y)$
- polynomial: $f : (x, y) \mapsto 4 - 2x^2 - 2y^2$
- rationnal: $f : (x, y) \mapsto \dfrac{2}{(4 + x + y)^2}$
- constant: $f : (x, y) \mapsto \dfrac{1}{2}$

The termination criterion for Conjugate Gradient method is a precision, epsilon, of 10^{-12} on the residual.

The speed-up is defined by $S_p = \frac{T_1}{T_p}$ where T_p is the observed elapsed time on p processor(s) and the efficiency is defined by $E_p = \frac{S_p}{p}$.

5.3 Effect of the second member

We report the efficiency for the four considered second members on the two architectures. The parameters are set to:

- 64×64 elements,

Table 2. Effect of the second member

Second member	Number of iterations	Efficiency on IBM-SP3	Efficiency on Cluster
$f : (x, y) \mapsto 4\pi^2 \sin(\pi x) sin(\pi y)$	15	1.14	0.96
$f : (x, y) \mapsto 4 - 2x^2 - 2y^2$	516	1.16	0.96
$f : (x, y) \mapsto \dfrac{2}{(4 + x + y)^2}$	921	1.07	0.96
$f : (x, y) \mapsto \dfrac{1}{2}$	835	1.05	0.96

- 15×15-point discretization on an element,
- 16 processors
- 16×16-element repartition on each processor,

Table 2 shows that the efficiency does not depend on the nature of the second member. Whatever the number of iterations is (and so the computation amount), the efficiency remains the same for both architectures.

5.4 Effect of the number of discretization points

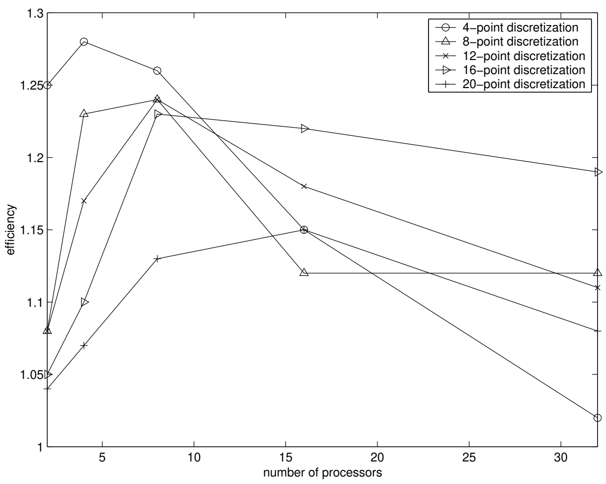

Fig. 5. *Efficiency* on the IBM-SP3 Brodie with different *numbers of discretization points* on each element

We present in Figures 5 and 6, the efficiency of our code with different numbers of discretization points on each spectral element.

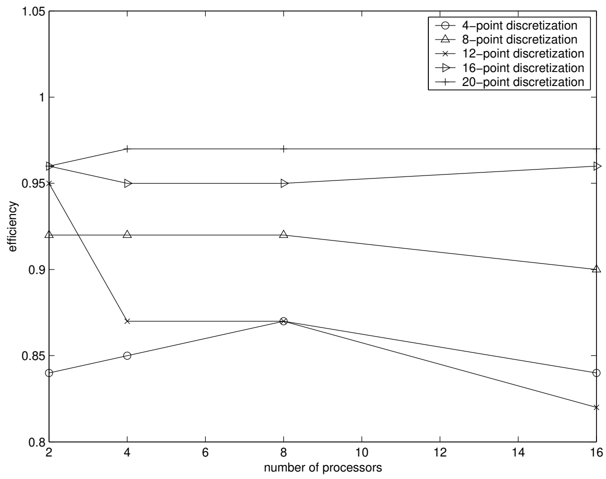

Fig. 6. *Efficiency* on the PC-Cluster Tarasque with different *numbers of discretization points* on each element

Table 3. Number of iterations with different discretizations

Number of discretization points	Number of iterations
4	239
8	372
12	473
16	555
20	631

The number of elements is 64×64, the second member is constant ($f : (x,y) \mapsto \frac{1}{2}$) and results are obtained with 2, 4, 8, 16 and 32 processors on IBM-SP3 and 2, 4, 8 and 16 processors on PC-Cluster.

These results show that the number of discretization points does not influence the efficiency very much. When we increase the number of points on each element (and so the computation amount, see Table 3), the efficiency decreases on the IBM-SP3 although it increases on the PC-Cluster. It is therefor difficult to draw conclusions.

We can note than on the IBM-SP3, the efficiency decreases significantly when we use 32 processors. This can be explained easily by the fact that when we use less than 16 processors, we are within a single node of the machine and the communications are intra-node; using 32 processors means that we use two nodes and change the nature of the communications.

5.5 Scalability

In this subsection, we present results on the two architectures in order to show the good scalability of our code. The parameters are:

- rationnal second member : $f : (x, y) \mapsto \dfrac{2}{(4 + x + y)^2}$
- 64×64 elements,
- 15×15-point discretization on an element,

We use the different repartitions of Table 1.

Tables 4 and 5 presents observed elapsed time, speed-up and efficiency for different number of processors.

Table 4. Efficiency and speed-up on the IBM-SP3 Brodie

Number of processors	Repartition	time	Speed-up	Efficiency
1	64×64	1040.11	-	-
2	64×32	512.55	2.03	1.02
4	32×32	243.80	4.26	1.07
4	64×16	249.33	4.17	1.04
8	32×16	115.42	9.01	1.13
8	64×8	115.97	8.97	1.12
16	16×16	60.73	17.13	1.07
16	32×8	60.68	17.14	1.07
16	64×4	62.36	16.68	1.04
32	16×8	35.45	29.34	0.92
32	32×4	33.36	31.18	0.97
32	64×2	33.69	30.87	0.96

Table 5. Efficiency and speed-up on the PC-Cluster Tarasque

Number of processors	Repartition	time	Speed-up	Efficiency
1	64×64	6251.44	-	-
2	64×32	3264.49	1.91	0.96
4	32×32	1628.61	3.84	0.96
4	64×16	1628.84	3.84	0.96
8	32×16	814.18	7.68	0.96
8	64×8	813.43	7.68	0.96
16	16×16	408.58	15.30	0.96
16	32×8	411.00	15.21	0.95
16	64×4	408.35	15.31	0.96

The results exhibit the good scalability of the code ; efficiency remains the same when we increase the number of processors. As we said previously, computations overlay communications.

A second remark concerns the different repartitions for a fixed number of processors. The results are quite the same whatever the repartition is (expecially on the cluster) ; repartition only concern the size of the interface between processors. When we change repartition, some interfaces of a processors are increased and some others reduced; the number of values to communicate remain the same because our elements are square.

The Figures 7 and 8 synthetize these results with the linear curve of the speed-up; for number of processors with several repartitions, we only plot the speed-up of the first repartition.

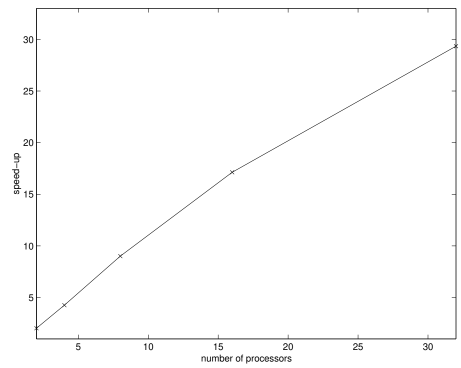

Fig. 7. *Speed-up* on the IBM-SP3 Brodie with different *numbers of processors*

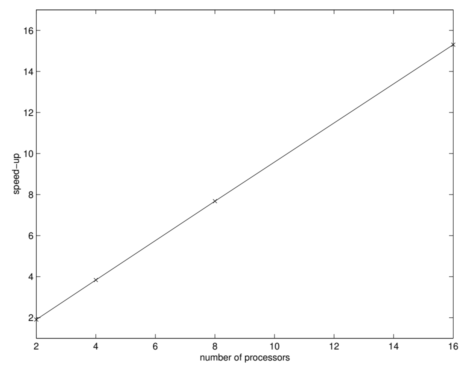

Fig. 8. *Speed-up* on the PC-Cluster Tarasque with different *numbers of processors*

6 Conclusion

In this work, we have proposed a parallel implementation of spectral element methods to solve a 2D Poisson problem. The performance of this implementation have been studied on two architectures, an IBM-SP3 and a PC-Cluster following several criteria: different second members, different numbers of discretization points, different repartitions of the elements on the processors.

It appears that our approach exhibits good parallel performance on both platforms and our code shows a good scalability.

Future works can study parallel implementation of much complicated problems: convection diffusion problem, Navier-Stokes problem in two-dimensional or three-dimensional domains.

References

1. V.I.Agoshkov, *Poincaré-Steklov's operators and domain decomposition methods in finite dimensional spaces*, First International Symposium on Domain Decomposition Method for Partial Differential Equations, R. Glwinski, G.H. Golub, G.A. Meurant and J. Périaux eds., SIAM, Philadelphia, pp. 73-112 (1988).
2. C. Bernardi and Y. Maday, *Approximations spectrales de problèmes aux limites elliptiques*, Paris, Springer Verlag (1992).
3. A.Quarteroni and A.Valli, *Domain Decomposition Methods for Partial Differential Equations*, Oxford University Press, Ofxord (1999).