Comparison of message-passing and shared memory implementations of the GMRES method on MIMD computers

Joanna Płażek[a], Krzysztof Banaś[a] and
Jacek Kitowski[b,c,*]
[a]*Section of Applied Mathematics FAPCM, Cracow University of Technology, Warszawska 24, 31-155 Cracow, Poland*
[b]*Institute of Computer Science, University of Mining and Metallurgy, al. Mickiewicza 30, 30-059, Kraków, Poland*
[c]*Academic Computer Center CYFRONET, ul. Nawojki 11, 30-950 Kraków, Poland*
Tel.: +48 12 617 3964; Fax: +48 12 338 054; E-mail: kito@uci.agh.edu.pl

In this paper we compare different parallel implementations of the same algorithm for solving nonlinear simulation problems on unstructured meshes.

In the first implementation, making use of the message-passing programming model and the PVM system, the domain decomposition of unstructured mesh is implemented, while the second implementation takes advantage of the inherent parallelism of the algorithm by adopting the shared-memory programming model. Both implementations are applied to the preconditioned GMRES method that solves iteratively the system of linear equations. A combined approach, the hybrid programming model suitable for multicomputers with SMP nodes, is introduced.

For performance measurements we use compressible fluid flow simulation in which sequences of finite element solutions form time approximations to the Euler equations. The tests are performed on HP SPP1600, HP S2000 and SGI Origin2000 multiprocessors and report wall-clock execution time and speedup for different number of processing nodes and for different meshes. Experimentally, the explicit programming model proves to be more efficient than the implicit model by 20–70%, that depends on the mesh and the machine.

Keywords: Parallel programming, finite element method, iterative solvers, compressible fluid flow, GMRES method, shared memory and message-passing programming, domain decomposition, PVM, SMP

1. Introduction

Architecture details of parallel computers make it possible to define diversity of machine taxonomies (see for example [10,28,51]), however one of the most important factors is organization of the address space. Between the two extremes, i.e., the shared address space organization and the distributed memory architecture, there is a wide class of machines with virtually shared, physically distributed memory organization (often called Distributed Shared Memory, DSM, machines). DSM can be software or hardware implemented; the latter one offers better characteristics and several typical classes, like cc-NUMA (cache coherent non-uniform memory access), COMA (coherent only memory architecture) and RMS (reflective memory systems). At present cc-NUMA implementations are commercially the most popular. According to Flynn's taxonomy [17] they belong to Multiple Instruction/Multiple Data (MIMD) computers. Examples come from HP (Exemplar and SuperDome with two interconnection layers) and from SGI (Origin2000/3000 with fat hypercube topology). The similar approach is implemented in IBM RS/6000 SP computers with SMP nodes.

Parallel computing adopts two classical programming models – the message passing and the shared address space. The former (explicit) is often treated as an assembler for parallel programming, while the latter (implicit) simplifies the programming process and is applied recently for the network of workstations [29]. For the experienced user it is easier to obtain better parallel efficiency with the message passing model. Since the advanced multiprocessors and clusters are constructed with SMP nodes the choice between the programming models is not obvious and some inte-

*Corresponding author.

Scientific Programming 9 (2001) 195–209
ISSN 1058-9244 / $8.00 © 2001, IOS Press. All rights reserved

gration of multiprocessing and multithreading would be profitable in the future. Due to faster communication within the SMP nodes than between the nodes, it would be probably not the best choice to implement DSM in the entire program (since they are specially designed for the implicit programming). The better way is to adopt this kind of programming within the SMP nodes, while using the explicit programming between the nodes. Such an approach is consistent with present trends in high performance computing, in which the dominant architecture will be clusters of SMPs in its many variants [46].

Most programming problems can have several parallel solutions. A popular design methodology is to consider distinct stages, like machine-independent stages including partitioning and communications and also machine-specific stages of design that deal with agglomeration and mapping. This kind of design is often called the PCAM methodology [18].

For the Finite Element (FE) Method applied to numerical approximation of nonlinear partial differential equations, both implicit and explicit approaches could be adopted, combined with iterative solvers for systems of linear equations and domain decomposition preconditioners. The solvers are usually implemented with imperative programming, which offers high performance. Newly appearing approaches make use of object-oriented paradigm implemented in the C++ language and/or network paradigm represented by Java (e.g. [22,31,42,52]).

Overlapping domain decomposition methods are iterative in nature, thus solutions on the individual subdomains can be combined together in order to formulate the overall solution. Iterative solvers can also be applied on any individual subdomain, which constitutes the room for further development of hierarchical or multilevel algorithms, some of them published to date, e.g. [32,34,47].

In this paper we compare different parallel implementations of the same algorithm for solving nonlinear simulation problems on unstructured meshes using the adaptive finite element approach. For this purpose we developed a parallel adaptive finite element algorithm in the version designed to solve the compressible Euler equations of inviscid fluid flow. The general purpose part consists of finite element data structure routines, including mesh modification procedures and solvers. For solving systems of linear equations, which in the case of fluid flow problems are nonsymmetric, the GMRES method combined with overlapping domain decomposition preconditioner is applied [37].

To get full advantage of parallelization, efficient implementations of domain decomposition (including load balancing and minimizing the communication cost) and of the solver should be taken into account. In this paper, however, we are interested in different parallel implementations of the same algorithm rather than in sophisticated methods concerning the domain decomposition (which are kept simple for this study), thus dealing with machine-specific issues of the PCAM methodology.

We introduce two levels of parallelization to the algorithm. Using one-level parallelism only explicit or implicit programming can be applied. With two-level parallelism a hybrid model is considered to effectively use SMP nodes. The paper describes the essential extensions to the algorithm which has been partially already presented [34–36].

The goal of the paper is to compare performance of different parallel models, that implement the same mathematical algorithm.

2. Finite element algorithm

As a test bed for parallel implementations of GMRES an h-adaptive finite element code designed to solve compressible Euler and Navier-Stokes equations has been used. The code implements standard stabilized finite element algorithms (Streamline Diffusion [1,20, 21,44] with pseudo time stepping for steady state problems or modified Taylor-Galerkin [12] for transient problems). The approximation to the Euler equations is split into a sequence of solutions to implicit finite element problems with a system of non-symmetric linear equations solved at each time step. Adaptivity is based on refinement/derefinement technique [11] with the use of explicit residual error indicators for compressible flow problems [13]. The details of the finite element algorithm can be found in [3,4,6,7]. This rather simple algorithm, still however of practical significance, demonstrates flexibility for explicit, implicit and hybrid implementations. It contains the most important ingredients found in modern finite element solvers for CFD, like error based adaptivity, Krylov space solvers and domain decomposition preconditioning. From the point of view of parallelization, the techniques developed for this particular algorithm can be easily transferred to other solvers.

The standard finite element procedures for solving an implicit, linear problem consist in creation of element stiffness matrices and load vectors, assembling

them into a global stiffness matrix and a global load vector and then solving a resulting system of linear equations. The latter task is often performed by a separate, general purpose library procedure or specialized algorithms (e.g. [40]). The parallelization of such a solver is done independently of the finite element mesh and the problem to be solved.

In the reported case the solvers are built into the algorithm and use a particular data structure related to the finite element mesh. It is assumed that the finite element module of the code provides the solvers with element stiffness matrices and load vectors, together with information on element connectivities. The connectivity information, in the form of lists of element nodes and, for each node, elements sharing a node, is sufficient for assembling the global stiffness matrix and the load vector (also in the case of irregular meshes with hanging nodes). The code assembles and stores the global stiffness matrix in a special data structure related to the decomposition of the computational domain into small subdomains. These small subdomains, called patches of elements, consist of several elements only (with one or more internal nodes). There is a patch data structure created for each patch, consisting of the corresponding blocks of the stiffness matrix and the load vector. When each node of the finite element mesh belongs to the interior of only one small subdomain (for standard meshes this corresponds to one element overlap between subdomains) than blocks of the stiffness matrix in patch structures do not overlap. In that case the storage scheme is just a version of the block compressed row storage. Otherwise, it provides a useful extension to it for the case of overlapping subdomains.

The patch data structure is directly used in the solver algorithm that is built around block iterative methods. Patch stiffness matrices form stiffness matrices for local, subdomain problems, forming building blocks of the algorithm.

The GMRES algorithm [39,41,48,50] is one of the most successful and widely used iterative methods for solving nonsymmetric systems of linear equations. It is especially suited for nonlinear or time dependent problems where the whole simulation is split into a sequence of separate step problems. The solution from the previous step problem forms then a perfect candidate for the initial guess in GMRES iterations.

The performance of the GMRES depends on the conditioning of a system of equations so usually it is used with left or right preconditioning. In the case of left preconditioning, instead of solving the original system of equations $\mathbf{Ax} = \mathbf{b}$ (with $\mathbf{A}$ the global stiffness matrix, $\mathbf{x}$ the vector of unknown degrees of freedom and $\mathbf{b}$ the global load vector), the preconditioned system $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$ is solved (see e.g. [6]). The preconditioning matrix $\mathbf{M}$ should be easily invertible and designed in such a way that the preconditioned system has better convergence properties than the original one (the condition number of the product $\mathbf{M}^{-1}\mathbf{A}$ should be close to one). The preconditioned GMRES algorithm schematically is presented in Fig. 1.

Since the convergence of the GMRES was not the subject of investigations, the number of Krylov space vectors N_{ksv} was set to 10, a small arbitrary number, often found in large scale simulations. The initial guess $\mathbf{x}^0$ was formed by the solution from the previous time step.

Preconditioning of the GMRES algorithm can be efficiently achieved by using any of basic iterative methods such as the standard Jacobi, Gauss-Seidel or block Jacobi, block Gauss-Seidel methods [19].

We have implemented the preconditioning using block Jacobi and block Gauss-Seidel algorithms where all operations in the GMRES involving the global stiffness matrix $\mathbf{A}$ are performed by means of loops over local (block) problems.

In the default setting of the solver (used in all computational examples reported) the local problems have minimal dimension with patches having only one internal node. This option corresponds to blocks in standard iterative methods related to unknowns at each finite element node and has proven to be the most efficient for the Taylor-Galerkin method [3].

3. Parallel implementation of GMRES

3.1. Parallelization with explicit programming model

The GMRES algorithm preconditioned by standard iterative methods can be interpreted as domain decomposition algorithm [30,47]. It reflects the general framework of Schwarz methods: divide the computational domain into subdomains, solve the problem for each subdomain separately and combine the solutions. The role of subdomains is played by patches of elements and single iterations of iterative methods accelerated by the GMRES are just inexact solutions of subdomain problems. Depending on the preconditioner we have either an additive Schwarz method (block Jacobi preconditioner – contributions from different subdomains are summed up) or a multiplicative Schwarz method (block Gauss-Seidel preconditioner –

```
compute the initial residual of the system: r_0 := M^{-1}(b - Ax^0)
normalize the residual: r̄_0 := r_0/||r_0||
for i = 1,2,...,N_ksv
  compute preconditioned matrix-vector product: r_i := M^{-1} A r̄_{i-1}
  orthonormalize r_i wrt all previous r̄_j, j = 1,..,i-1 by the modified
    Gram-Schmidt procedure obtaining r̄_i
  solve the GMRES minimization problem and check convergence
  if convergence achieved form the approximate solution and leave GMRES
endfor
form the approximate solution and substitute it for the initial guess x^0
start from the beginning
```

Fig. 1. The restarted GMRES algorithm.

contributions are taken into account while looping over patches).

Since the number of patches is usually much larger than the number of processors, the second level of domain decomposition is introduced, equivalent for finite elements with suitable mesh partitioning. The number of subdomains at this level (submeshes) is equal to the number of computing nodes of a parallel machine. The subdomains possess one element overlap so each node in the finite element mesh belongs to the interior of only one subdomain. Using the message passing programming model, data corresponding to a particular subdomain are sent and stored in the memory of one computing node. The computing node corresponding to a given subdomain (storing its data) is responsible for all computations related to the subdomain (computing element stiffness matrices, assembling them into patch matrices, solving local problems).

Having distributed the data structure, computations of the GMRES algorithm begin. Some vector operations, such as scalar product or normalization, require communication between computing nodes and are done by standard message passing procedures using a selected computing node to gather the data. Other operations, like subtraction or multiplication by a scalar, are done entirely in parallel. The GMRES minimization problem is solved on a chosen computing node.

Iterations of the block Jacobi method are done in two steps. First the loop over all internal FE nodes of corresponding subdomains is executed by each computing node separately. Patches of elements are created and for each patch (i.e. finite element node) a local problem is solved by a direct solver (e.g. Gauss elimination).

Then all subdomains exchange data on interface nodes. Due to one element overlap between subdomains, each node is updated during computations only by one computing node (corresponding to the subdomain owning the node). Hence the exchange of data after iteration loops is simplified – the computing node owning a given interface node just broadcasts its data to neighboring subdomains.

3.1.1. Mesh partition algorithm

The optimal partition of the mesh should keep minimal the execution time of the whole problem solved by the finite element method. Equal load of the processors is usually imposed on the domain decomposition algorithm. The partition determines two important factors: the convergence properties of an iterative method used to solve the system of linear equations [38,47] and the amount of data exchanged between computing nodes during the solution procedure. The former factor favorites subdomains with regular shape and bigger overlap. The influence of the latter depends additionally on the architecture of the parallel system, especially the bandwidth of data transfer between processing nodes.

Many partitioning algorithms of the unstructured meshes have already been proposed, like coordinate bisection, spectral bisection and graph partitioning, see for instance [9,15,24,25,33,45,53]. The finite element tearing and interconnecting methods (FETI), e.g. [16] have been applied for elliptic partial differential equations. Several libraries, like Chaco [23], Metis and ParMetis [26,27] or TOP/DOMTEC [14] exist to solve graph partitioning or similar problems. Space filling curve methods represent another approach [2,8]. In the case of remeshing predictive load balancing techniques can be used, e.g. [49]. An overview of graph partitioning is presented in [43].

As it has been noticed in the introduction, in this paper we are interested in parallel implementation of the solver rather than in the domain decomposition methods. For this purpose we introduce a simple heuristic partitioning algorithm that, nevertheless, has several advantages. It enables creation of subdomains with arbitrary number of nodes and handles efficiently h-adaptive meshes with hanging nodes. In the code it is used for creating both types of subdomains: small subdomains for block iterative preconditioning and large subdomains for parallel execution. The slightly simplified version of the algorithm, with all technical complications caused by the presence of constrained (hanging) nodes omitted, is presented in Fig. 2.

```
set an initial content of the front F
for all subdomains S
  while(N_n^S < N_max^S)
    pick node N from F (according to the prescribed weight)
    if(N ∉ S) add N to S
    for all elements E such that N is a vertex of E
      if(E ∉ S)
        add E to S
        for all vertices N_v of E, N_v ≠ N
          if(N_v ∉ F and N_v not internal to S) add N_v to F
          if(N_v ∉ S) add N to S
        endfor
      endif
    endfor
    update F (delete all nodes internal to S)
  endwhile
```

Fig. 2. The mesh partition algorithm (N_n^S – the curent number of nodes in a subdoamin, N_{max}^S – the maximal allowable number of nodes in a subdomain).

The standard for greedy type algorithms principle of adding the nearest neighbor to the created subdomain is combined with an intermediate step of creating a front (group) of nodes being the candidates for adding to the subdomain. The presence of heuristic weights at nodal points, allows for using different mesh partition strategies. These strategies are based on some geometrical principles. The first node in a given subdomain becomes a reference point and then we create subdomains trying to minimize the distance of the subdomain boundary from this point. Choosing different definitions for the distance in 2D space we get different strategies for mesh partition by obtaining different shapes of subdomains and as a consequence different sizes of inter-subdomain boundaries. The choice of the strategy, which results in minimal number of elements shared by the processors, follows from the qualitative knowledge concerning the solution and local density of mesh points. In Fig. 3(b)–(g) several partitions of the mesh from the Fig. 3(a) are presented with corresponding distance definitions. The expanse of white between the partitions represents the shared elements. We implemented also the greedy Farhat criterion [14, 15] of adding the node which belongs to the minimum number of elements. In all examples the total number of 8441 nodes is almost uniformly distributed among subdomains, each of which possesses more than one thousand internal nodes. The initial front for all cases is the same and consists of the lower left corner node.

We developed a simple strategy for load balancing [7] which is used together with the heuristic strategy for definition of the distance in 2D space. The load balancing strategy exploits the fact that in the mesh partition algorithm we can specify the number of nodes for each subdomain. In the presented GMRES implementation, the workload for a given computing node is proportional to the number of FE nodes. Then the load balance in the corresponding subdomain is achieved by ascribing to each computing node a subdomain with the number of FE nodes proportional to power of the computing node. As an estimate for computing node performance the inverse of the average time for computing local stiffness matrices is taken. The information on the performance is sent as an input to the procedure which specifies the numbers of nodes for particular subdomains. These numbers form the input for the mesh partition algorithm. Combining the strategy for load balancing with the strategy of distance definition one is able to get a decomposition which is useful for solver testing.

To retain the efficiency of the algorithm for adaptive meshes, for which the number of nodes and elements grows considerably, the loops in Fig. 2 can be performed for initial mesh elements and the corresponding nodes only. At that moment adding an element to a subdomain consist of adding the element and all its ancestors with all their nodes. Additional techniques are introduced to maintain the one element overlap between subdomains. As a result, the speed of the algorithm depends only on the parameters of the initial mesh.

3.1.2. Implementation

The code is written in the C language and uses typical C features like structures and dynamic memory allocation. The structures are used for nodes, elements and patches data representation. Access to data is realized by an array of pointers to the structures.

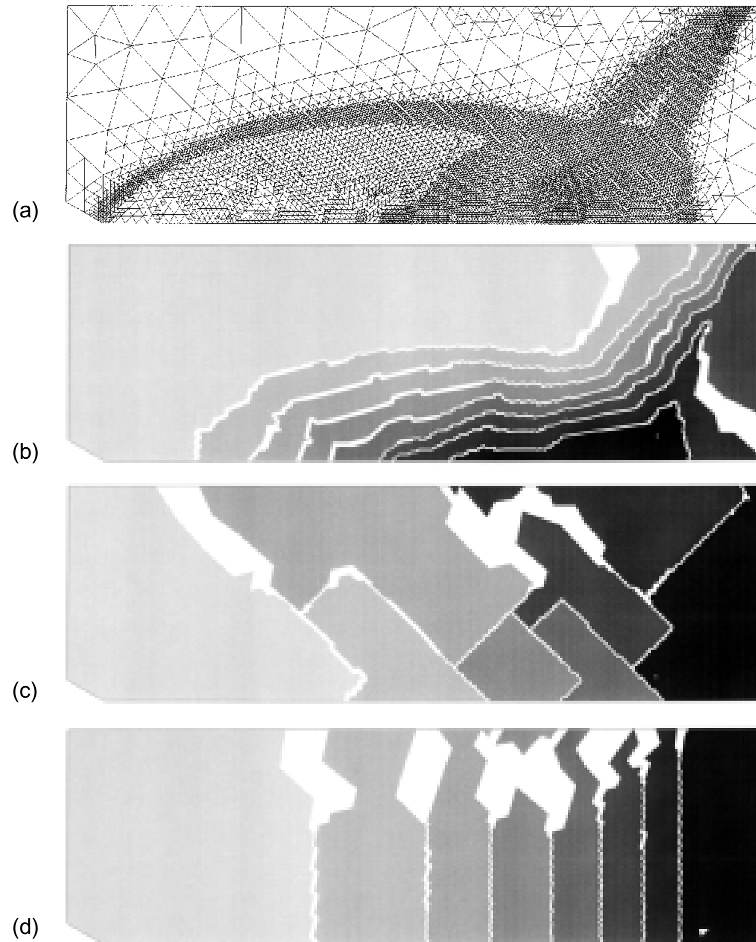


Fig. 3. The mesh (a) and partitions using different strategies in the domain decomposition algorithm. (b) no weights associated with front nodes, (c) weights based on the distance: $\rho(x,y) = |x_1 - y_1| + |x_2 - y_2|$, (d) weights based on the distance: $\rho(x,y) = |x_1 - y_1|$, (e) weights based on the distance: $\rho(x,y) = |x_2 - y_2|$, (f) weights based on the distance: $\rho(x,y) = \max(|x_1 - y_1|, |x_2 - y_2|)$, (g) weights based on the distance: $\rho(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

Characteristic feature of the present algorithm is the existence of patch structures, each storing all data (including lists of nodes and elements, assembled stiffness matrices and load vectors) for a corresponding patch.

The exchange of data between subdomains is based on arrays storing, for each subdomains, lists of nodes for which data is sent to or received from the neighboring subdomains.

The practical realization of the whole computations in the explicit model is achieved using master – slave paradigm. There exists one master process that controls the solution procedure and slave processes performing in parallel the most of calculations. The flow diagram of the whole simulation is presented in Fig. 4. Since the implementation of the explicit programming model makes use of the PVM system for message passing between different processing nodes, it is denoted by PVM in the following sections.

Adaptation of the mesh can be performed between any two time steps of the algorithm. Usually for simulations of transient problems, adaptations of the mesh are frequent while for steady state problems only several adaptations are required in the whole solution procedure. Partitioning of the mesh is performed after mesh adaptation. This may mean that different mesh partition strategies are optimal for these two types of problems.

In the current setting of mesh adaptation all the domain information must fit the master processor. To overcome this effect fast refinement-derefinement technique is used for mesh modification, rendering the time for adaptation to the range of several percents of the to-

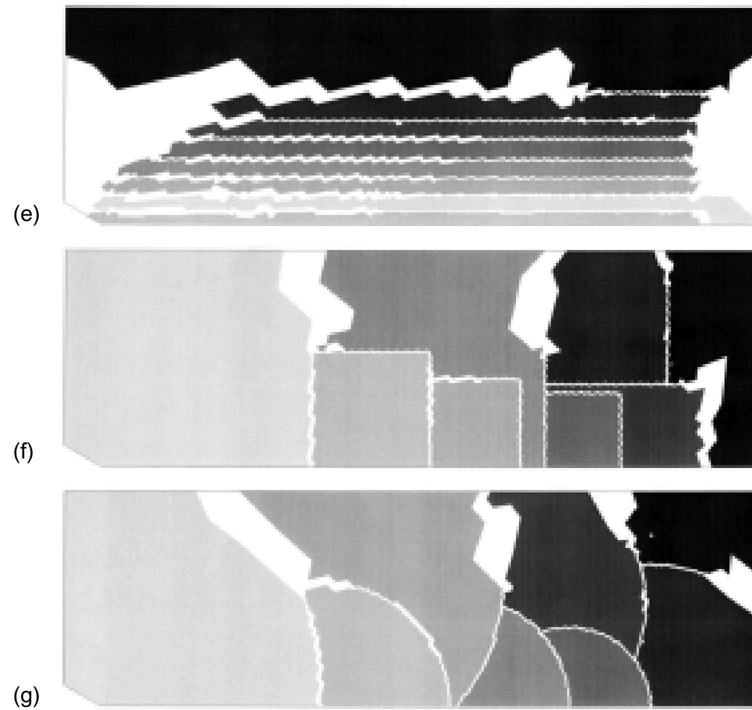


Fig. 3. continued.

tal computing time. Nevertheless, despite the fact that expensive error estimation is done perfectly in parallel, the algorithm is expected to scale only up to the range of several dozens of processors. Then special parallel adaptation strategies has to be designed.

3.2. Parallelization with implicit programming model

Compilers for parallel computers parallelize codes automatically, verifying loop dependencies. Such an approach results in sufficient efficiency for simple loops only. For example, a subroutine call inside a loop prevents it from possibility of automatic parallelization. To overcome those problems compiler directives/pragmas are additionally used to increase degree of parallelization and to enable manual control of many aspects of execution.

In the implicit programming model, the GMRES algorithm sketched in Fig. 1 (written in C language in a similar way to the explicit one) is parallelized using compiler directives whenever a loop over patches, nodes, or individual degrees of freedom is encountered. In particular the parallelization is applied for:

- loops over blocks of unknowns at the step of blocks construction,
- computation of element stiffness matrices and assembly into patch stiffness matrices,
- iterations of the block method.

Since the implicit programming model makes use of distributed shared memory of the multiprocessors, its practical implementation is denoted by DSM in the following sections.

3.3. Parallelization with hybrid programming model

A hybrid programming model is a combination of explicit and implicit models. In this model, suitable for a machine with SMP nodes (or for a cluster of multiprocessor workstations), we introduce two levels of parallelization:

- first-level parallelization with pure explicit programming model making use of geometrical data decomposition into subdomains; a number of subdomains is equal to the number of SMP nodes coordinated by a message passing library.
- second-level parallelization with the implicit programming model for each SMP node, that makes use of the shared memory for performing iterations for the block method.

```
start slave processes on different processors of Parallel Machine
divide the whole mesh into submeshes and send
  subdomain data to corresponding processors (slave processes)
receive subdomain data
for each time step
  create patches of elements corresponding to blocks of unknowns
          in the Jacobi preconditioner
  compute element stiffness matrices, assemble them into matrices
          of block problems, invert block problems' matrices
  repeat
  perform subsequent steps of the GMRES algorithm:
    the steps involving matrix-vector product consist of loops over all internal
        nodes of the submesh followed by the exchange of the data concerning
        the nodes on the boundary of the submesh
    computations of vector norms and scalar products require selection
        of one process that gather results from submeshes and then broadcast
        the final result to other processes
    other vector operations (axpy products) are performed perfectly in parallel
  until converge
  compute the error in time step and give back control to the master process
  either go directly to the next step or adapt the mesh
  if the mesh has been adapted
    divide the whole mesh into submeshes and send
      subdomain data to corresponding processors (slave processes)
    receive subdomain data
endfor
```

Fig. 4. Diagram of the whole simulation in the explicit programming model (indicating **master** and *slave* processes).

4. Results

We tested our parallel versions of GMRES (preconditioned with the block Jacobi method) with an example of flow simulations – a well known transient benchmark problem – the ramp problem [54]. A Mach 10 shock traveling along a channel and perpendicular to its walls meets at time $t = 0$ s a ramp, having an angle of 60 degrees with the channel walls. A complicated flow pattern develops with double Mach reflection and a jet of denser fluid along the ramp behind the shock. The mesh after initial adaptation is shown in Fig. 3(a), that presents qualitatively the solution.

The problem is solved with the Taylor-Galerkin method. Adaptations of the mesh are done every five steps. Since the flow is inviscid, a simplified version of the element error indicator is used:

$$e_K^2 = \left\| \frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} + \mathbf{f}^E_{k,U}\mathbf{U}_{,k} \right\|_{H_{,UU}}$$

where h is a linear element size, $\mathbf{U}$ is the vector of conservation variables at time t^n and t^{n+1} ($\Delta t = t^{n+1} - t^n$) and $\mathbf{f}^E_k$ are the vectors of Eulerian fluxes. The Euclidean vector norm is weighted by the Hessian of the entropy function H with respect to $\mathbf{U}$ [4].

The time step length is controlled by the constant CFL number equal to 2 [5].

In the explicit parallel programming model (denoted by PVM) we used a version of domain decomposition (mesh partition) algorithm that ensures vertical alignment of subdomain interfaces.

The results refer to the wall-clock execution time, T, for one time step chosen as a representation for the whole simulation, with different meshes (having $N = 4474, 16858, 18241$ and 73589 nodes and denoted in the following figures by s4t, s16t, s18t and s73t respectively). To verify performance of the algorithm for different meshes and different programming models we fixed the number of restarted GMRES iterations, l, that keeps the convergence index, ϵ, at the acceptable level. We used $\epsilon = \| \mathbf{r}_l \|$, where $\mathbf{r}_l = \mathbf{x}^{l-1} - \mathbf{x}^l$ and $\mathbf{x}^{l-1}$, $\mathbf{x}^l$ are solution approximations in $l-1$ and l iterations respectively.

In Fig. 5 dependence of convergence index on the number of restarted GMRES iterations, l, for three meshes is presented. Since $\epsilon \leqslant 10^{-10}$ for $l \geqslant 5$, $l = 5$ was adopted.

To get statistically more reliable results the measurements have been collected three times from 5 subsequent time steps, since fluctuations in T of several percents were observed.

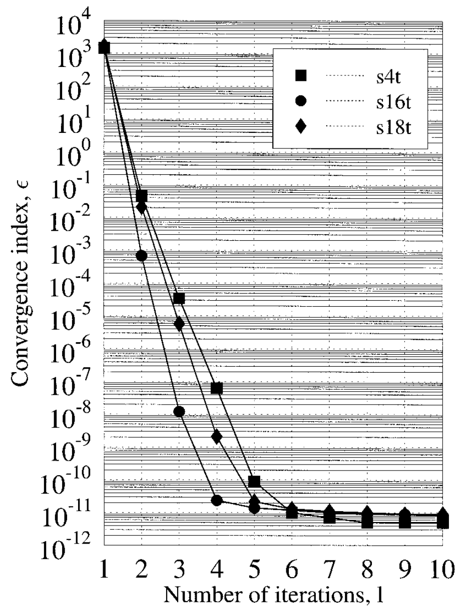


Fig. 5. Dependence of convergence index on number of restarted GMRES iterations.

Table 1
Number of interface nodes for different kinds of the mesh partition strategies

#Domains	Number of the interface nodes		
	Strategy from Fig. 3(b)	Strategy from Fig. 3(d)	With Greedy strategy
2	350	114	368
3	772	179	855
4	1115	324	1319
6	1790	465	1241
8	2542	800	1453
10	4881	1356	1917

4.1. Performance of the mesh partition algorithm

The final result of partition produced by the algorithm from Fig. 2 depends, for a given computational domain, on the initial front and the way of selecting nodes from the front. On the basis of knowledge concerning the solution and local density of mesh nodes, from many possibilities presented in Section 3 we implemented three for explicit programming to compare their efficiency.

The initial front for all cases is the same and consists of the lower left corner node. Table 1 shows the number of the interface nodes (representing the communication cost) for strategies presented in Figs 3(b) and (d) supplemented with the greedy Farhat's strategy [14, 15]. These values were obtained for the ramp problem with the number of nodes equal to 18241.

It can be noticed, that for the particular domain considered, the heuristic strategy presented in Fig. 3(d) results in the smallest number of the interface nodes minimizing the communication requirements. This result remains valid also if the mesh is refined, because remeshing is done mainly in the region of high mesh density which does not influence much mesh density profiles.

4.2. Performance of the parallel GMRES

Three parallel machines have been used: HP SPP1600 (32 PA 7200/120MHz processors organized in 4 SMP hypernodes, with SPP-UX 4.2, Convex C 6.5 and Convex PVM 3.3.10), HP S2000 (16 PA8000/180MHz processors in one hypernode using SPP-UX 5.2, HP C 2.1 and PVM 3.3.11) and SGI Origin2000 (32 R10000/250MHz processors, IRIX 6.5, SGI C 7.3 and SGI PVM 3.1.1). We run the code on a SPP1600 isolated subcomplex consisting of 16 processors from four SMP nodes. During the measurements on S2000 and on Origin2000 (subsequently called S2K and O2K respectively) no other users were allowed to use the machines.

In Figs 6(a) and (b) CXpa profiles for the wall-clock execution time, T, for one simulation step and for the implicit parallel programming (denoted by DSM) are reported. CXpa is a performance analysis tool for CONVEX and HP parallel computers [55] for monitoring program performance at user-selectable source code regions, such as routines, loops, and compiler-generated parallel loops. Collected data include wall-clock/cpu time, execution counts, cache miss counts and latency for memory access. In our case the profiles represent the time spent by the program in the most time consuming program modules. Seven execution threads were declared of a rather small problem with 4474 FE nodes. For execution with one thread, $T = 48$ s (see Fig. 6(a)), compared to $T = 19$ s obtained for seven threads (cf. Fig. 6(b)). Since the results concern program threads with procedures including children (e.g. the `main()` procedure includes time spent in all other routines), it can be noticed that sufficient parallel performance is obtained. The most involved modules are a `main()` procedure, `dumpin()` for reading input data, `adapt()` for performing mesh adaptation and `gmressol()` for solving a finite element problem using preconditioned GMRES method. The last mentioned procedure calls other modules, like `assebj()` to compute element stiffness matrices and assembly them into patch stiffness ma-

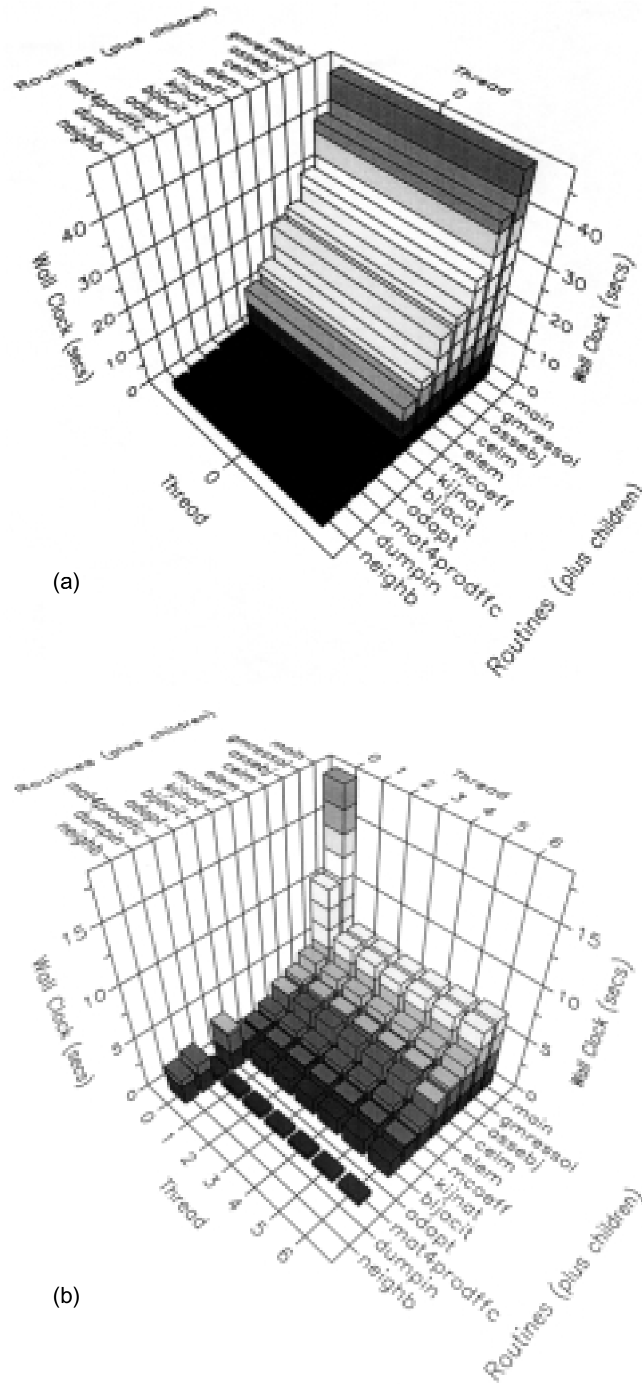


Fig. 6. CXpa profiles (SPP 1600) for wall-clock execution time for one simulation step and for the implicit parallel programming model for: (a) one and (b) seven threads with children, $N = 4474$.

trices using `celm()`, `elem()`, `mcoeff()`, `kijnat()`, `mat4prodffc()` modules and `bijacit()` to perform one block Jacobi iteration. Due to rather high CXpa overhead the results can be interpreted qualitatively only.

In Fig. 7(a) the wall-clock execution time, T, for different number of processors, K, for different machines and programming models is presented for number of

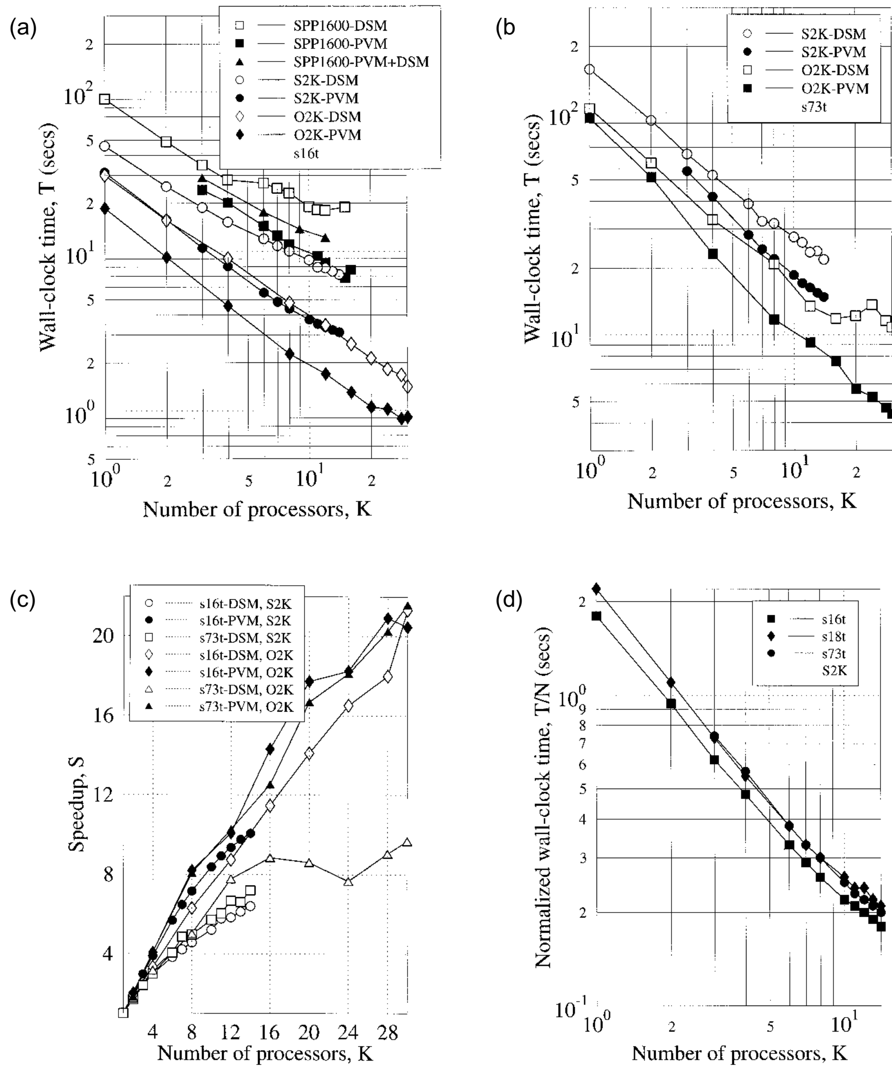


Fig. 7. Wall-clock execution time for one simulation step, different programming models and machines: (a) the mesh with $N = 16858$ nodes, (b) with $N = 73589$ nodes. (c) Speedup for different meshes ($N = 16858$ and 73589 nodes). (d) Normalized T for $N = 16858$, 18241 and 73589 nodes.

nodes $N = 16858$. The architecture of SPP1600 has influenced results of the DSM implementation, while results from the explicit model remained undistored by the lower bandwidth between the hypernodes. Better scalability is observed for the explicit model in comparison with DSM, although the latter results are obtained with relatively small programmer's effort. For 30 Origin2000 processors and the PVM implementation characteristic saturation becomes evident, while kept monotonic for DSM. Results for the hybrid implementation (denoted by PVM+DSM) are in between those obtained for PVM and DSM.

Difference in performance between explicit and implicit models for $K = 1$ results from distinct nodes numbering in the meshes and consequently different cache performance.

In Fig. 7(b) timings for a greater mesh ($N = 73589$) are shown. Again, the difference between the PVM and DSM implementations is not high for rather small number of processors ($K \leqslant 16$). The significant difference is obtained for $K > 16$, i.e., for O2K hypercube dimensionality $d > 2$. For the DSM implementation unexpected response is found, with local maximum for $K = 24$ and monotonic execution time decrease for $K > 24$. This feature results from the complicated O2K communication topology. Contrary to DSM, no

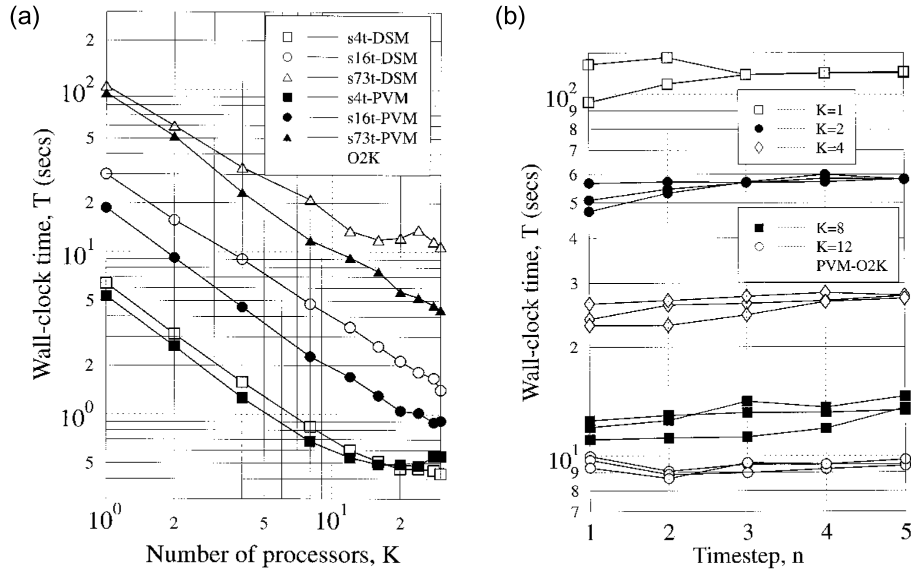


Fig. 8. (a) Timings for O2K for different mesh sizes and models, (b) Wall clock time distribution for each of the first 5 time steps and for the mesh $N = 73589$.

characteristic saturation is observed for the PVM implementation due to higher computation to communication ratio, resulting from higher data locality.

In all cases the DSM implementation shows worse performance in comparison with PVM. Although the machines are the shared memory computers, DSM hardware represents different access times between differing levels of memory (i.e. non-uniform memory access). This fact introduces difference in the latency and in the memory bandwidth, that reduce the performance.

The PVM implementation takes advantage of the PVM system adopted by the vendors. No additional knowledge of the architecture has been included. The architecture details influence the hybrid implementation (PVM+DSM), which is more flexible in choice of communication and computation granularities, according to the surface-to-volume ratio [18].

Relative speedup values, S, are depicted in Fig. 7(c). Good scalability is obtained for the message-passing model, with better characteristics for Origin2000. Despite of the interval $16 < K < 28$ (with the biggest mesh, $N = 73589$ influenced again by different levels of memory), the DSM implementation demonstrates higher speedup on Origin2000 than on S2000. In Fig. 7(d) we present the wall-clock execution time normalized to number of mesh nodes, T/N. Since the characteristics are very close each other, this confirms experimentally linear computational complexity $o(N)$ of the algorithm.

In Fig. 8(a) timings for Origin2000 are collected. For a small mesh ($N = 4474$), a minimum is observed due to small computation to communication ratio. The scalability is getting better as the ratio increases. Wall clock time distribution for each of the first 5 time steps and the PVM implementation is presented in Fig. 8(b). It reflects uncertainty in wall clock time measurements (due to operating system overhead), that confirms reasons for multiple measurements.

5. Conclusions

In the paper we have focused on parallel implementations of the algorithm for GMRES method. The DSM implementation demonstrates useful and scalable parallelization. No PCAM methodology is applied, making the development easier, for the price of suitable choice of data structures and program control flow. In particular, this implementation is adequate for rather small number of processing nodes, although this limitation depends on the architecture of the computer system and the problem size. In our case, a practical limit for processors is $K = 4$ (for SPP1600 and $N = 16858$) and $K = 16$ (for Origin2000 and $N = 73589$). No performance degradation is observed on S2000, due to only one (SMP-) hypernode available.

The PVM implementation still demonstrates higher performance and speedup characteristics closer to the ideal case for the price of more complicated code struc-

ture. In the development stage application of the PCAM methodology is useful to obtain efficient implementation. In our case, in spite of use of the simple partitioning method, no essential limitation of this implementation is encountered. The biggest production run with $N \approx 375000$, completed for the PVM implementation in the 32-processor Origin2000 multiuser environment, proved sufficient practical performance.

The PVM implementation is less sensitive to the communication bandwidth than DSM. Changing from one SMP node execution to multi-SMP node execution, the former is only weakly affected, while the latter suffers the performance degradation. The PVM implementation is suitable also for the clusters of computers. In this implementation the PCAM methodology should be carefully applied, considering partitioning and communication models as well as implementation issues of atomic tasks agglomeration and mapping on the architecture [18]. On machines with vast number of processors implemented for big problems solving, difficulties with load-balancing and with computation to communication ratio kept high could be encountered. On the other hand, the DSM implementation, although easy to develop and efficient for small number of processors, would suffer of low performance and excessive resources consuming while the problems got bigger. Thus neither of the specific implementations is universal.

Since most of the modern machines with vast number of processors implement different layers of memory and also because the dominant architecture for the future is expected to be clusters of SMPs [46], the hybrid model, that makes use of the both implementations, would be a choice for those kinds of machines. If necessary, in this model both domain and functional partitioning methods can be implemented simultaneously. The present study shows the possibility and feasibility of using the hybrid approach for parallelization of finite element, and hopefully, other scientific codes. To fully assess advantages of this approach against the explicit model more numerical studies for systems consisting of a large number of SMP nodes should be performed.

Future work could consider also several topics, like tuning the solvers for each type of architecture, coding in parameters, which estimate processor performance, memory access speed and the interconnection bandwidth, to be used later on for further solver optimization. Other possibilities include implementation of more sophisticated partitioning methods, load-balancing procedures and parallel adaptation strategies.

In practice, to get advantage of parallel implementations on tightly coupled architectures, more processors have to be available. Otherwise, in multiuser environments with vast number of users, performance loss is observed.

Acknowledgments

The work has been sponsored by Polish State Committee for Scientific Research (KBN) Grants 8T11C 006 15, 7T11F 014 21 and 7T11F 014 20. We thank Dr. Monika Dekster (AGH Cracow) for her assistance in preparation of the manuscript.

References

[1] S.K. Aliabadi, S.E. Ray and T.E. Tezduyar, SUPG finite element computation of viscous compressible flows based on the conservation and entropy variables formulations, *Computat. Mechanics* **11** (1993), 300–312.

[2] S. Aluru and F. Sevilgen, Parallel domain decomposition and load balancing using space-filling curves, in: *Proc. of 4th Int. Conf. on High Performance Computing, HiPC97*, V.K Prasanna etal., eds, IEEE Comp. Society, Bangalore, Dec. 18–21, 1997.

[3] K. Banaś, Convergence to steady-state solutions for stabilized finite element simulations of compressible flows, *Computers and Mathematics with Applications* **40** (2000), 625–643.

[4] K. Banaś and L. Demkowicz, Entropy controlled adaptive finite element simulations for compressible gas flow, *J. Computat. Physics* **126** (1996), 181–201.

[5] K.Banaś and L.Demkowicz, New quasi-natural artificial viscosity models for compressible fluid flow, with improved entropy production mechanism, *J. Theor. Appl. Mech.* **35** (1997), 233–248.

[6] K. Banaś and J. Płażek, Dynamic load balancing for the preconditioned GMRES solver in a parallel, adaptive finite element Euler code, in: *Proc.Third ECCOMAS Computat. Fluid Dynamics Conf.*, J.-A. Desideri et al., eds, J. Wiley & Sons Ltd., Paris, Sep. 9–13, 1996, pp. 1025–1031.

[7] K. Banaś and J. Płażek, Parallel h-adaptive simulations of inviscid flows by the finite element method, *J. Theor. Appl. Mech.* **35** (1997), 249–262.

[8] J. Behrens and J. Zimmermann, Parallelizing an unstructured grid generator with space-filling curve approach, in: *Proc. of Euro-Par2000 Conf.*, P. Bode, T. Ludwig, W. Karl and R. Wismuller, eds, Munchen, 2000, *Lecture Notes in Computer Science* **1900** (2000), 815–823, (Springer 2000).

[9] F. Chalot, G. Chevalier, Q.V. Dinh and L. Giraud, Some investigations of domain decomposition techniques in parallel CFD, in: *Proc. of Euro-Par'99 Conf.*, P. Amestoy, P. Berger, M. Dayde, I. Duff, V. Fraysse, L. Giraud and D. Ruiz, eds, Toulouse, Aug. 31–Sep. 3, 1999, pp. 595–602, (Springer 1999).

[10] D.E. Culler, J.P. Singh and A. Gupta, Parallel Computer Architecture. A Hardware/Software Approach, Morgan Kaufmann Publ., 1999.

[11] L. Demkowicz, J.T. Oden, W. Rachowicz and O. Hardy, Towards a universal h-p adaptive finite element strategy, Part.1 Constrained approximation and data structure, *Comp. Meth. Appl. Mech. Engng.* **77** (1989), 79–112.

[12] J. Donea, A Taylor-Galerkin method for convective transport problems, *Int. J. Numer. Meth. Eng.* **20** (1984), 101–119.

[13] K. Ericsson and C. Johnson, Adaptive streamline diffusion finite element methods for stationary convection diffusion problems, *Mathem. of Comput.* **60** (1993), 167–188.

[14] C. Farhat, S. Lanteri and H.D. Simon, TOP/DOMDEC – a software tool for mesh partitioning and parallel processing, *Comput. Systems Eng.* **6** (1995), 13–26.

[15] C. Farhat, N. Maman and G.W. Brown, Mesh partitioning for implicit computations via iterative domain decomposition: impact and optimization of the subdomain aspect ratio, *Int. J. Numer. Meth. Eng.* **38** (1995), 989–1000.

[16] C. Farhat, K. Pierson and M. Lesoinne, The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems, *Comp. Meth. Appl. Mech. Engng.* **184**(2–4) (2000), 333–374.

[17] M.J. Flynn, Some computer organizations and their effectiveness, *IEEE Trans. on Computers*, **C-21**(9) (1972), 948–960.

[18] I. Foster, Designing and Building Parallel Programs, Addison Wesley, 1995.

[19] G. Golub and J.M. Ortega, Scientific Computing (Academic Press, San Diego, 1993). Numerical Approximation of Partial Differential Equations (Springer, Berlin, 1994).

[20] P. Hansbo and C. Johnson, Adaptive streamline diffusion method for compressible flow using conservation variables, *Comp. Meth. Appl. Mech. Engng.* **87** (1991), 267–280.

[21] P. Hansbo, Explicit Streamline Diffusion Finite Element Methods for the Compressible Euler Equations in Conservation Variables, *J. Computat. Physics* **109** (1993), 274–288.

[22] J. Hauser, T. Ludewig, R.D. Williams, R. Winkelmann, T. Gollnick, S. Brunett and J. Muylaert, A test suite for high-performance parallel Java, *Proc. of Fifth NASA National Symp. on Large-Scale Analysis, Design and Intelligent Synthesis Environments*, Williamsburg, Oct. 12–15, 1999, *Advances in Engineering Software* (*Elsevier*) **31**(8–9) (2000), 687–696.

[23] B. Hendrickson and R. Leland, The Chaco User's Guide. Version 2.0. Technical Report, SAND95-2344, Sandia National Laboratories, Albuquerque, NM, 1995.

[24] B. Hendrickson and T. Kolda, Graph partitioning models for parallel computing, *Parallel Computing* **26**(12) (2000), 1519–1534.

[25] S.-H. Hsieh, G.H. Paulino and J.F. Abel, Recursive spectral algorithms for automatic domain partitioning in parallel finite element analysis, *Comput. Meth. Appl. Mech. Eng.* **121** (1995), 137–162.

[26] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20**(1) (1998), 359–392.

[27] G. Karypis, K. Schloegel and V. Kumar, ParMetis – Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 2.0, User's Manual, Dept. of Comp. Sci. University of Minnesota, 1998.

[28] V. Kumar, A. Grama, A. Gupta and G. Karypis, Introduction to Parallel Computing, The Benjamin/Cummings Publishing Company, Redwood City, 1994.

[29] K.D. Ryu and J.K. Hollingsworth, Exploiting fine grained idle periods in networks of workstations, *IEEE Trans. on Parallel and Distrib. Systems* **11**(7) (2000), 683–698.

[30] P. LeTallec, Domain decomposition method in computational mechanics, in: J.T. Oden, ed., Amsterdam, North Holland, 1994.

[31] P. Luksch, Parallel and distributed implementation of large industrial applications, *Future. Gen. Comp. Syst.* **16**(6) (2000), 649–663.

[32] M. Papadrakakis, S. Bitzarakis and A. Kotsopulos, An improved dual domain decomposition technique for computational structural mechanics, in: *Proc.Second ECCOMAS Conf. Numer. Methods in Engng*, J.-A. Desideri et al., eds, J. Wiley & Sons Ltd., Paris, Sep. 9–13, 1996, pp. 482–490.

[33] A. Patra and J.T. Oden, Problem decomposition for adaptive hp finite element methods, *Comput. Systems Eng.* **6** (1995), 97–109.

[34] J. Płażek, K. Banaś and J. Kitowski, Finite element message-passing/DSM simulation algorithm for parallel computers, in: *Proc.HPCN'98*, P. Sloot, M. Bubak and B. Hertzberger, eds, Amsterdam, April 21–23, 1998, *Lecture Notes in Computer Science* **1401** (1998), 878–880, (Springer, 1998).

[35] J. Płażek, K. Banaś and J. Kitowski, Implementation Issues of Computational Fluid Dynamics Algorithms on Parallel Computers, in: *Proc. of 6th European PVM/MPI Users' Group Meeting*, J. Dongarra, E. Luque and T. Margalef, eds, Barcelona, Spain, September 1999, *Lecture Notes in Computer Science* **1697** (1999), 349–355, (Springer, 1999).

[36] J. Płażek, K. Banaś and J. Kitowski, Explicit and Implicit Parallel Programming Models for Finite Element Method on Multiprocessor Systems, in: *Proc. of SGI Users Conference*, M. Bubak, J. Mościński and M. Noga, eds, Cracow, Oct. 11–14, 2000, pp. 324-333, (ACK Cyfronet AGH 2000).

[37] A. Quarteroni and A. Valli, Numerical Approximation of Partial Differential Equations, Springer, 1994.

[38] W. Rachowicz, An overlapping domain decomposition preconditioner for an anisotropic h-adaptive finite element method, *Comput. Meth. Appl. Mech. Eng.* **127** (1995), 269–292.

[39] Y. Saad and M. Schultz, GMRES: a generalizad minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comp.* **7** (1986), 856–869.

[40] Y. Saad and M. Sosonkina, Non-standard parallel solution strategies for distributed sparse linear systems, in: *Proc. of ParNum'99 Conf.*, P. Zinterhof, M. Vajtersic and A. Uhl, eds, Salzburg, Feb. 16–18, 1999, pp. 13–27, (Springer 1999).

[41] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Pub. Company, 1996.

[42] M. Sato, K. Kusano, H. Nakada, S. Sekiguchi and S. Matsuoka, Net CFD: a NinfCFD component for global computing, and its Java applet GUI, *Proc. of Fourth Int.Conf./Exhibition on HPC in the Asia-Pacific Region*, Beijing, May 14–17, 2000, *IEEE Comput. Soc. Part* **1** (2000), 501–506.

[43] K. Schlogel, G. Karypis and V. Kumar, Graph Partitioning for High-Performance Scientific Simulations, in: *CRPC Parallel Computing Handbook,* J. Dongarra et al., eds, Morgan Kaufmann, 2000.

[44] F. Shakib, T.J.R. Hughes and Z. Johan, A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations, *Comp. Meth. Appl. Mech. Engng.* **89** (1991), 141–219.

[45] H.D. Simon, Partitioning of unstructured problems for parallel processing, *Comput. Systems Eng.* **2**(2/3) (1991), 135–148.

[46] H.D. Simon, HPCwire interview with Horst Simon on his talk "The Future of Supercomputers", given at 16-th Int. Supercomputer Conf., Heidelberg, June 21–23, 2001, *HPCwire* **10**(20), May 18, 2001.

[47] B. Smith, P. Bjorstad and W. Gropp, Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differnetial Equation, Cambridge University Press, Cambridge, 1996.

[48] M. Sosonkina, D.C.S. Allison and L.T. Watson, Scalable parallel implementations of the GMRES algorithm via Householder reflections, in: *Proc. 1998 Int. Conf. Parallel Processing*, T. Lai, ed., Minneapolis, Aug. 11–14, 1998, *IEEE Comput. Soc.* (1998), 396–404.

[49] B.K. Szymanski, J.E. Flaherty, R.M. Loy, C. Ozturan, M.S. Shephard, J.D. Teresco and L.H. Ziantz, Predictive load balancing for parallel adaptive finite element computation, in: *Proc. of PDPTA'97 Conf.*, (Vol. 1), H.R. Arabnia, ed., 1997, pp. 460–469.

[50] E. de Sturler and H.A. van der Vorst, Communication cost reduction for Krylov methods on parallel computers, in: *Proc. HPCN'94 Conf.*, W. Gentsch and U. Harms, eds, April 18–20, 1994, Munich, *Lecture Notes in Computer Science* **797** (1994), 190–195, (Springer, 1994).

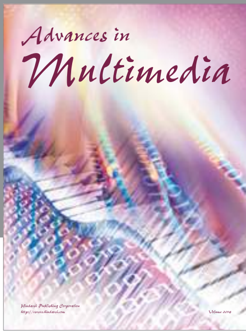
[51] A.S. Tanenbaum, *Distributed Operating Systems,* Prentice-Hall, Inc., 1995.

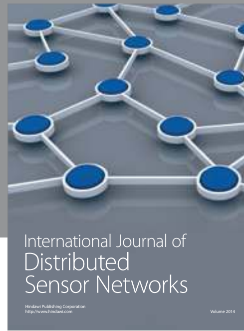
[52] H. Uehara and M. Hatakeyama, Object-oriented parallelization system for object-oriented CFD simulation system, *J. of the Japan Society for Simulation Technology* **19**(2) (2000), 128–142.

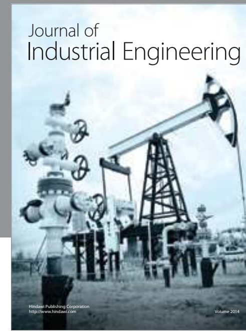
[53] R. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, *Concurrency: Practice and Experience* **3**(5) (1991), 457–481.

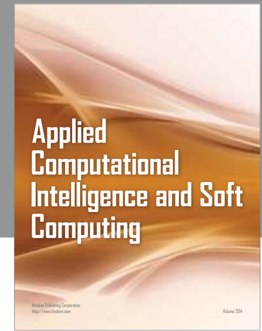
[54] P. Woodward and P. Colella, The numerical simulation of two dimensional fluid flow with strong shocks, *J. Computat. Phys.* **54** (1984), 115–173.

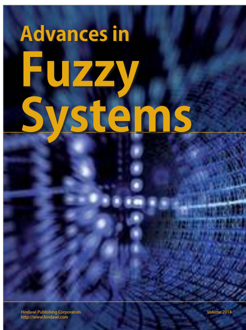
[55] —, CONVEX CXpa Reference for Exemplar Systems, First Edition, Richardson, 1995.











Hindawi

Submit your manuscripts at
http://www.hindawi.com