Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT•

ACADEMIC PRESS

J. Parallel Distrib. Comput. 63 (2003) 827–837

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

An efficient 3D grid based scheduling for heterogeneous systems

Anthony T. Chronopoulos,[a,*] Daniel Grosu,[a] Andrew M. Wissink,[b] Manuel Benche,[a] and Jingyu Liu[a]

[a] Department of Computer Science, University of Texas at San Antonio, 6900 N. Loop 1604 West, San Antonio, TX 78249, USA
[b] Center for Applied Scientific Computing, Lawrence Livermore National Lab, P.O. Box 808, L-661, Livermore, CA 94551, USA

Received 23 October 2001

Abstract

The cost/performance ratio of networks of workstations has been constantly improving. This trend is expected to continue in the near future. The aggregate peak rate of such systems often matches or exceeds the peak rate offered by the fastest parallel computers. This has motivated research toward using a network of computers, interconnected via a fast network (cluster system) or a simple Local Area Network (LAN) (distributed system), for high performance concurrent computations. Some of the important research issues arise such as (i) Problem partitioning and virtual interconnection topology mapping; (ii) Execution scheduling and load balancing.

Past results exist for grid partitioning (into subdomains) and mapping to parallel and distributed systems. In our work we consider the problem of grid partitioning of a 3D domain arising in aircraft CFD simulations in order to schedule tasks for load balanced execution on a heterogeneous distributed system. This problem has additional restrictions on how to partition the grid. Past work for this problem were on parallel systems with only few processor configurations. We derive heuristic algorithms for: (1) homogeneous systems with any number of processors; (2) heterogeneous systems taking into account the processor speed and memory capacity. We implement our algorithms on a dedicated network of workstations (using MPI) and test them with a CFD simulation code (TURNS—Transonic Unsteady Rotor Navier Stokes).

Keywords: Distributed systems; CFD simulation; Load balancing

1. Introduction

Distributed computation offers the potential for cheaper and high-performance computations. Some companies are beginning to utilize parallel processing in the form of clusters of workstations, that are idle during off-hours, to attain supercomputer performance [19]. Portable parallel software exist for implementing codes on distributed computer environments (e.g. Message Passing Interface (MPI) [7]). Several researchers have studied the problems of scheduling and load balancing computations for concurrent execution in distributed environments. For example see [1,2,5,17,22] and the references therein.

The benefits of this research are immense for research agencies (e.g. NASA) and the broader scientific community. Design companies and government agencies have clusters of fast microcomputers, which are underutilized. Thus design or testing engineers could run these simulation codes on these clusters essentially free of cost.

1.1. Past results

There exist a large number of approaches to grid partitioning that have been considered in the past. These approaches are trying to find a mapping of the computational grid onto processors such that the total execution time is minimized. Such a mapping can be obtained by solving a graph partitioning problem which is known to be NP-complete [15]. We can categorize these approaches into three classes: (1) spectral partitioning methods [14], (2) geometric partitioning methods [12], and (3) multilevel partitioning methods (see [9,10,15] and references therein). Also, some recent

*Corresponding author. Fax: +210-458-4437.
E-mail addresses: atc@cs.utsa.edu (A.T. Chronopoulos), dgrosu@cs.utsa.edu (D. Grosu), awissink@llnl.gov (A.M. Wissink).

0743-7315/$ - see front matter © 2003 Published by Elsevier Inc.
doi:10.1016/S0743-7315(03)00112-6

results on dynamic graph partitioning for scientific simulations can be found in [16].

We consider an application of geometric partitioning which has been used to parallelize the computations for each integration step of a CFD computation in a homogeneous system [20]. The domain (see Fig. 2) has some special restrictions in how to be partitioned. The proposed parallelization algorithm is hard wired inside the CFD code and it assigns equal subdomains to each processor and it only works for special selections of processor configurations (see the appendix).

The CFD code simulates helicopter aerodynamics and it is written in FORTRAN using MPI [18,20,21]. The baseline numerical method is the structured-grid Euler/Navier–Stokes solver TURNS (Transonic Unsteady Rotor Navier Stokes) (see [18] and references therein). The implicit operator used in TURNS for time-stepping in both steady and unsteady calculations is the Lower–Upper symmetric Gauss–Seidel (LU-SGS) operator of Yoon and Jameson [23]. The parallel TURNS version uses the Hybrid LU-SGS and is a parallel modification to LU-SGS [20]. Once the computational space has been divided into subdomains, the original LU-SGS algorithm is applied simultaneously to each processor subdomain. Then, border data between the subdomains is communicated using the relaxation-type approach of DP-LUR [20]. Wissink implemented the CG-type iterative methods GMRES and OSOmin [3,21] in approximating the solution of the nonlinear system arising at each time step in the CFD integration.

Storage is a major consideration for the solution of three-dimensional problems. The Jacobian matrix is not computed because CG-type methods use only Jacobian times vector products which are approximated by Taylor expansion. The CG-type methods require a modest amount of extra storage compared to the storage required by the LU-SGS method [20]. The LU-SGS method is used as a preconditioner in these iterative methods to speed up their convergence rate. For more details about CG-type methods and preconditioning see [6]. Two CG-type methods (GMRES and OSOmin) which were tested with TURNS (with LU-SGS preconditioning) with grid size $135 \times 50 \times 35$ on 4, 8, 19, 57 and 114 processors of IBM SP2 gave similar convergence and performance results [20].

1.2. *Our results*

We undertake the task of obtaining scheduling algorithms for the CFD application in [20] for any configuration of homogeneous or heterogeneous processors. We propose geometric partitioning type algorithms which apply to a 3D grid domain problem which must be partitioned with some special restrictions. The aim is to schedule the CFD computations of each subdomain on a dedicated system with an arbitrary number of: (I) homogeneous processors, and (II) heterogeneous processors. Parallel solution of such CFD problems have been based on equal grid partitioning [20]. Each processor was assigned a grid subdomain with equal number of grid points. The three-dimensional flowfield spatial domain is divided in the wraparound and spanwise directions to form a two-dimensional array of processor subdomains, as shown in Fig. 2. Here, we consider the implementation on a (Ethernet) network of heterogeneous workstations (Fig. 1). The programming paradigm used is SPMD (Single Program Multiple Data) where each processor executes an identical copy of code on a fraction of the computational grid. We obtained an improved algorithm for homogeneous systems which works for every processor configuration. We then deal with the heterogeneity (in processor speed and memory) of the processors by subdividing the space domain into subdomains with unequal number of grid points.

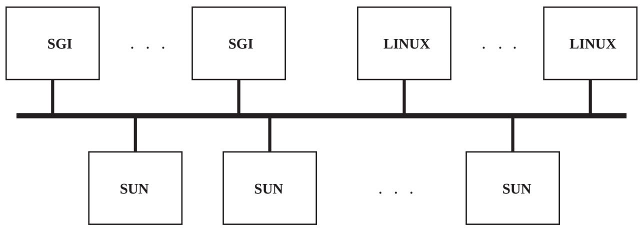


Fig. 1. The heterogeneous network of workstations.

The problem is how to obtain and map these partitions on a virtual mesh of heterogeneous processors. Due to the model implementation in TURNS, only two dimensions (J and K) of the three-dimensional grid($J \times K \times L$) are partitioned. The J dimension is partitioned into equal subpartitions and the K dimension is partitioned according to the speed of each row of processors. The speed of a processors column is defined as the speed of the slowest processor on that column. Thus, every processor is expected to complete execution in time proportional to the ratio of its load (i.e. the grid points mapped to the processor) over processing speed. Our algorithm checks every possible mesh configuration and proposes a configuration that minimizes the execution time. Also our algorithm takes into consideration the memory size of each machine in making the allocation decision.

Using our scheduling algorithm (II) we were able to obtain an improvement in the speedup up to 100% for LU-SGS (see the appendix), and up to 80% for OSOmin (see the appendix), compared with the equal subdomain allocation algorithm (I).

1.3. *Organization*

The remainder of this paper is organized as follows. In Section 2 we present a scheduling algorithm for homogeneous systems and a scheduling algorithm for heterogeneous systems. In Section 3, we present the implementation and we discuss experimental results. Section 4 contains our conclusions and future work.

2. Scheduling algorithms

We now present the algorithms for homogeneous and heterogeneous systems.

2.1. *Algorithm for homogeneous systems*

Assumptions. (1) We parallelize only the problem space domain for one integration time-step.

(2) Processors (PEs) of the parallel system are of the same design and speed.

(3) We assume a 2-D logical PE rectangular mesh with $p = R \times C$ PEs, where R is the number of PE rows and C is the number of PE columns (Fig. 2). PEs will be referred as $P_{rc},\ r = 0, 1, \ldots, R-1,\ c = 0, 1, \ldots, C-1$.

Since the processors have the same speed the goal is to assign an equal load to each PE, where the *load* is defined as a 3-D box of grid points in the space domain.

In mapping the space domain of $J \times K \times L$ grid points to a logical two-dimensional mesh of PEs the following restrictions apply:

(i) Because of the symmetric boundary condition applied at the airfoil surface in the J direction (data at $(j, *, 1)$ must equal data at $(J-j, *, 1)$), the same number of grid points are assigned to processors $P_{r,*}$ and $P_{R-r-1,*}$. This means that if J is odd then R must be odd, because J points are distributed among R PEs.

(ii) The L dimension is not divided at all. We only partition the $J \times K$ mesh and assign one partition to each PE.

(iii) Each PE has only four adjacent PEs (implied by (ii)).

(iv) No PE can have fewer than 5 grid points assigned in each direction (J or K). The reason is that each PE has two shared boundary grid points with each of its four adjacent PEs (Fig. 3).

Let a (or $a+1$)and b (or $b+1$) be the number of grid points allocated on J and respectively K direction, where $a = \lfloor (J-2)/R \rfloor + 2$ and $b = \lfloor (K-2)/C \rfloor + 2$. In a mapping any number of the four types of processor loads: $a \times b$, $(a+1) \times b$, $a \times (b+1)$ and $(a+1) \times (b+1)$ may occur (see Fig. 4).

We use as an estimate of the total execution time (T_{est}), the load corresponding to the largest value of the loads that occurred in the mapping divided by the PE speed. For simplicity we omit the division by speed because all PEs are assumed to have the same speed. For some configurations all four types of loads may not occur and then we will consider the maximum among these loads. The goal is to minimize T_{est}, by finding an optimal configuration $p = R \times C$ of processors under our assumptions and restrictions. Fig. 4 is such an example with $p = R \times C = 5 \times 3$.

Remark. Given a number of available PEs and a space grid it is possible that a smaller number of PEs may produce a lower T_{est}. For example, for $p = 59$ and a grid with $J = 135$, $K = 50$ and $L = 35$, the best factorization ($R = 59$, $C = 1$) gives $a = 4$ and $b = 50$, and $T_{\text{est}} = (a+1) \times b = 250$. On the other hand, for $p = 58$ and the same grid, the best factorization (R = 29, C = 2) gives $a = 6$ and $b = 26$, thus $T_{\text{est}} = (a+1) \times b = 182$. The best configuration in this case is the second one because it gives a lower T_{est}.

Our algorithm checks every possible factorization $p = R \times C$ of p PEs and proposes a configuration with minimum T_{est} over all configurations under our assumptions. Note that it is possible that a configuration with a smaller number of processors can produce a smaller T_{est}. The algorithm starts from the number q of available PEs and returns p PEs (where $p = R \times C$ and

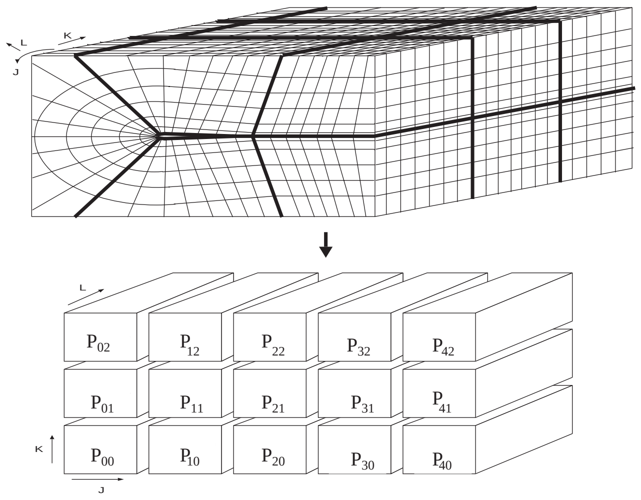


Fig. 2. Mapping the three-dimensional domain on a two-dimensional array of processors.

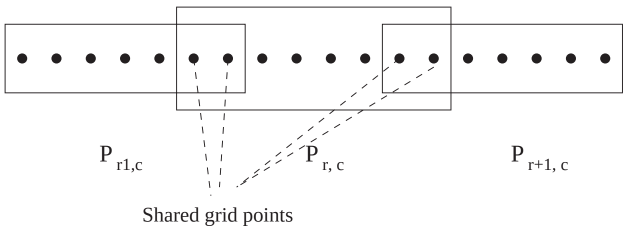


Fig. 3. Shared boundary grid points.

$p \leqslant q$) such that $(R, C) = \text{argmin}\, T_{\text{est}}$. Our heuristic approach checks these configurations by decrementing q by one at each stage.

Given p PEs our algorithm checks every possible factorization of p and forms the logical PE mesh with the lowest T_{est}.

Algorithm I.

```
Phase 1: Find $(R, C)$ that gives minimum $T_{est}$
    for $p = q$ downto 1 do
        for all $(R, C)$ where $p = R \times C$ do
                    $\min_{(R,C)} T_{est}(R, C)$;
    return $p = R \times C = \text{argmin}\ T_{est}$;

    Function $T_{est}(R, C)$
        $a = \lfloor (J-2)/R \rfloor + 2$;
        $b = \lfloor (K-2)/C \rfloor + 2$;
        $a_{rem} = (J-2) \bmod R$;
        $b_{rem} = (K-2) \bmod C$;
        if (($J$ is odd) and ($R$ is even))
                return error("symmetry restriction violated");
        else
                {Compute $T_{est}$}
                if ($a_{rem} = 0$)
                        $T_{est} = a$;
                else
                        $T_{est} = a + 1$;
                if ($b_{rem} = 0$)
                        $T_{est} = b\, T_{est}$;
                else
                        $T_{est} = (b + 1) T_{est}$;
    return $T_{est}$
Phase 2: Map grid points to processors
    for $c = 0, 1, \ldots, C-1$ do
        {Map the $J$ direction to the $R$ PEs of the $c$-th PE column}
        if ($a_{rem} \neq 0$ and $a_{rem}$ is odd)
                Map $a + 1$ points to the $a_{rem}$ PEs:
                        $P_{0,c}, \ldots, P_{(\lfloor \frac{a_{rem}}{2} \rfloor - 1),c}, P_{R/2,c}, P_{(R - \lfloor \frac{a_{rem}}{2} \rfloor),c}, \ldots, P_{R-1,c}$;
                Map $a$ points to the remaining $(R - a_{rem})$ PEs;
        else {$a_{rem}$ is even}
                Map $a + 1$ points to the $a_{rem}$ PEs:
                        $P_{0,c}, \ldots, P_{(\frac{a_{rem}}{2} - 1),c}, P_{(R - \frac{a_{rem}}{2}),c}, \ldots, P_{R-1,c}$;
                Map $a$ points to the remaining $(R - a_{rem})$ PEs;
    for $r = 0, 1, \ldots, R-1$ do
        {Map the $K$ direction to the $C$ PEs of the $r$-th PE row}
        if ($b_{rem} > 0$)
                Map $b + 1$ points to the $b_{rem}$ PEs: $P_{r,0}, \ldots, P_{r,b_{rem}-1}$;
        Map $b$ points to the remaining $(C - b_{rem})$ PEs;
```

The algorithm has two phases. In Phase 1 the processor mesh configuration (R, C) that gives the minimum T_{est} is determined. In Phase 2 the processor mesh configuration determined in Phase 1 is used to map the grid points to processors. The values of $a, a_{\text{rem}}, b, b_{\text{rem}}$ used in Phase 2 are the values corresponding to the configuration (R, C) determined in Phase 1.

2.2. Algorithm for heterogeneous systems

We use the Assumptions 1 and 3 of Algorithm I, and we consider a system in which the PEs may have different speeds.

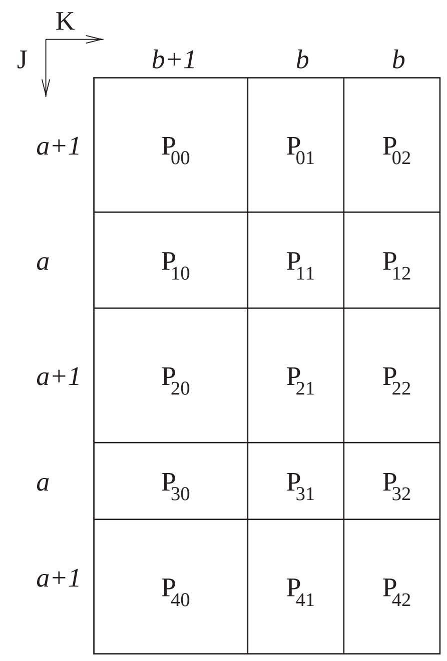


Fig. 4. The loads for 5×3 processors configuration.

Given a number of processors $p = R \times C$, we divide the $J \times K$ sized space grid in p rectangular partitions, with the J dimension divided into R subpartitions and the K dimension divided into C subpartitions. These subpartitions are in general of unequal size.

We form a linear array of PEs $(P_0, P_1, \ldots, P_{p-1})$ by sorting the PEs in non-increasing order according to their processing speed: $speed(P_0) \geqslant speed(P_1) \geqslant \ldots \geqslant speed(P_{p-1})$ and map the PE subpartition (r, c) of the 2D mesh to processor P_{cR+r}.

Because of the mapping of the PE linear array onto the 2D mesh and the ordering of PE according to the speed, the processors of a PE column are expected to have approximately the same computing power. Thus, we divide the J dimension into equal subpartitions. The K direction is partitioned and mapped according to the speed of each PE column, where the speed of a PE column is defined as the speed of the slowest PE on that column. This mapping may result in slightly underutilizing a fast processor, but it avoids the more important issue of overloading a slow one.

Every PE is expected to complete execution in time proportional to the ratio of its load over its processing speed. We take $\hat{T}_{\text{est}}$ to be the maximum of these ratios. The goal is to find a configuration (R, C) of PEs that minimizes $\hat{T}_{\text{est}}$.

Our algorithm checks every possible factorization $p = R \times C$ of p PEs and proposes an optimal configuration with minimum $\hat{T}_{\text{est}}$ over all configurations under our assumptions. This is a suboptimal solution to the general grid partitioning problem see [4,13] and references therein.

Note that it is possible that a configuration with a smaller number of processors can produce a smaller $\hat{T}_{\text{est}}$. An example similar to the Remark in Section 2.1 can be easily presented. The algorithm starts from the number q of available PEs and returns p PEs (where $p = R \times C$ and $p \leqslant q$) such that $(R, C) = \text{argmin}\ \hat{T}_{\text{est}}$. Our heuristic approach checks these configurations by decrementing q by one at each stage.

Algorithm II.

```
Phase 1: Find $(R, C)$ that gives minimum $\hat{T}_{est}$
    sort PEs $(P_0, P_1, \ldots, P_{q-1})$ in non-increasing order so that
        $speed(P_0) \geq speed(P_1) \geq \ldots \geq speed(P_{q-1})$;
    for $p = q$ downto 1 do
        for all $(R, C)$ where $p = R \times C$ do
                $\min_{(R,C)} \hat{T}_{est}(R, C)$;
    return $p = R \times C = \text{argmin}\ \hat{T}_{est}$;

    Function $\hat{T}_{est}(R, C)$
        $a = \lfloor (J-2)/R \rfloor + 2$;
        $a_{rem} = (J-2) \bmod R$;
        if (($J$ is odd) and ($R$ is even))
                return error("symmetry violated");
        else
                {Obtain the PE with the minimum speed within each PE column}
                for $c = 0, 1, \ldots, C-1$ {column $c$ of PEs}
                        $column\_speed(c) = \min_{r=0,1,\ldots R-1} speed(P_{r,c})$;
                {Compute the sum of all these minimum speeds}
                $total\_speed = \sum_{c=0}^{C-1} column\_speed(c)$;
                for $c = 0, 1, \ldots, C-1$
                        $l(c) = K \times \frac{column\_speed(c)}{total\_speed}$;
                $b_{rem} = K - \sum_{c=0}^{C-1} \lfloor l(c) \rfloor$;
                for $c = 0, 1, \ldots, C-1$
                        if ($c = 0$ and $c = C-1$)
                                $b(c) = \lfloor l(c) \rfloor + 1$;
                        else
                                $b(c) = \lfloor l(c) \rfloor + 2$;
                {Compute $\hat{T}_{est}$}
                if ($a_{rem} = 0$)
                        $\hat{T}_{est} = a$;
                else
                        $\hat{T}_{est} = a + 1$;
                if ($b_{rem} = 0$)
                        $\hat{T}_{est} = \hat{T}_{est} \max_{c=0,1,\ldots,C-1} (b(c)/column\_speed(c))$;
                else
                        $\hat{T}_{est} = \hat{T}_{est} \max_{c=0,1,\ldots,b_{rem}-1} ((b(c)+1)/column\_speed(c))$;
    return $\hat{T}_{est}$;
Phase 2: Map grid points to processors
    for $c = 0, 1, \ldots, C-1$ do
        {Map the $J$ dimension to the $R$ PEs of the $c$-th PE column}
        if ($a_{rem} \neq 0$ and $a_{rem}$ is odd)
                Map $a+1$ points to the $a_{rem}$ PEs:
                        $P_{0,c}, \ldots, P_{(\lfloor \frac{a_{rem}}{2} \rfloor - 1),c}, P_{R/2,c}, P_{(R - \lfloor \frac{a_{rem}}{2} \rfloor),c}, \ldots, P_{R-1,c}$;
                Map $a$ points to the remaining $(R - a_{rem})$ PEs;
        else {$a_{rem}$ is even}
                Map $a+1$ points to the $a_{rem}$ PEs:
                        $P_{0,c}, \ldots, P_{(\frac{a_{rem}}{2} - 1),c}, P_{(R - \frac{a_{rem}}{2}),c}, \ldots, P_{R-1,c}$;
                Map $a$ points to the remaining $(R - a_{rem})$ PEs;
    for $r = 0, 1, \ldots, R-1$ do
        {Map the $K$ dimension to the $C$ PEs of the $r$-th PE column}
        if ($b_{rem} > 0$)
                Map $b(c)+1$ points to the $b_{rem}$ PEs: $P_{r,c}$, $c = 0, 1, \ldots, b_{rem} - 1$;
        Map $b(c)$ points to the remaining $(C - b_{rem})$ PEs: $P_{r,c}$, $c = b_{rem}, \ldots, C-1$;
```

Remark. (i) Both algorithms only check rectangular meshes corresponding to integer factorizations of p. (ii) The algorithms only deal with 2D PE meshes because of the special parallel implementation of TURNS (see Introduction and [21]).

We next show an example to illustrate Algorithm II.

Example 1. We assume a distributed system of 15 heterogeneous PEs arranged in a 2D mesh presented in Fig. 5. In this figure the speed of each processor is given in parenthesis under the processor label. We consider a space grid with: $J = 135$ and $K = 50$ and we apply Algorithm II. In this example we are not interested in finding the best PE mesh configuration, we only show how the algorithms works for a given configuration. We consider the number p of PEs fixed and we skip the first for loop in Phase 1. Then the following values are computed:

$$a = \lfloor (135-2)/5 \rfloor + 2 = 28, \qquad a_{\text{rem}} = 3,$$

$$column\_speed(0) = 3, \qquad column\_speed(1) = 2,$$
$$column\_speed(2) = 1,$$

$$total\_speed = 6,$$

$$l(0) = 25, \qquad l(1) = 16.67, \qquad l(2) = 8.33,$$

$$b_{\text{rem}} = 1,$$

$$b(0) = 26, \qquad b(1) = 18, \qquad b(2) = 9.$$

The mapping obtained in Phase II is presented in Fig. 5. For example P_{00} will get $a + 1 = 29$ points on J direction and $b(0) + 1 = 27$ points on K direction.

We next use an example to show that Algorithm II is more efficient than Algorithm I, in heterogeneous systems.

Example 2. We assume that we have three columns on the PE mesh ($C = 3$). Let the speed of column c be $s(c)$ so that $s(1) > s(2) > s(3)$. We assume that there are K

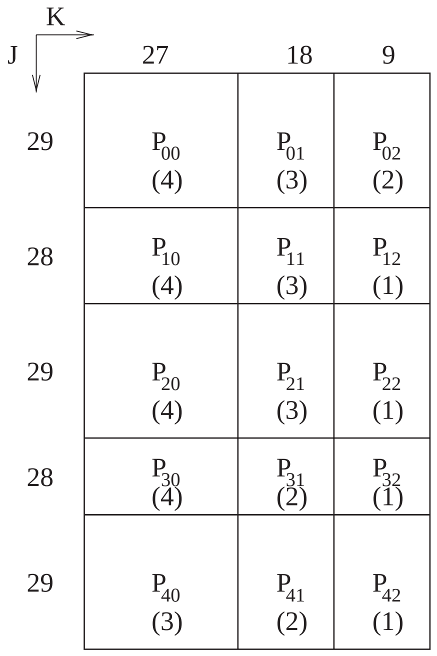


Fig. 5. The allocation for Example 1.

grid points on the K direction. First we use the equal allocation and each column of PEs will get $K/3$ points. In this case $T_{\text{est}} = K/(3\,s(3))$ because the slowest processor will finish last. Using the balanced allocation according to the Algorithm II we have the following allocations:

$$K\frac{s(1)}{s(1)+s(2)+s(3)}, \qquad K\frac{s(2)}{s(1)+s(2)+s(3)},$$
$$K\frac{s(3)}{s(1)+s(2)+s(3)}$$

for the three columns of PEs. In this case we have $\hat{T}_{\text{est}} = K/(s(1)+s(2)+s(3))$. Since $3\,s(3) < s(1)+s(2)+s(3)$ we conclude that $T_{\text{est}} > \hat{T}_{\text{est}}$.

3. Implementation and results

3.1. Distributed environment

In our experiments, we use a heterogeneous network of workstations which includes two groups of computers: (i) 45 SUN Ultra-10 (440 MHz, 128 MB), three Pentium PC (450 MHz, 128 MB), one SGI-O2 (270 MHz, 128 MB) and two SGI-O2 (20 MHz, 64 MB) and (ii) 12 SUN Ultra-1 (166 MHz, 64 MB). The workstations in group (i) are connected to each other via a 100 Mb/s switched Ethernet. The workstations in group (ii) are connected to each other via a 10 Mb/s switched Ethernet. As a message-passing library we use the MPICH 1.2.0 [7]. For compiling the TURNS code on SUN workstations we use the Sun WorkShop Compiler FORTRAN 90 SPARC Version 2.0, for SGI workstations we use the MIPSPro FORTRAN 90 compiler and for LINUX PCs we use the Lahey/Fujitsu LF95 compiler. We ran our programs in a dedicated environment.

3.2. Implementation details

We implemented (in C++) a load balancer based on the two algorithms described in the previous section. The first one considers the environment as a homogeneous network of workstations. The user has to provide the dimensions of the grid and the number of processors. The output is a parameter file used for compiling the TURNS code. The second program takes into consideration the heterogeneity of the computers. The user has to provide a list of machines with their speed and amount of memory in addition to the dimensions of the grid. The output is a parameter file used for compiling the TURNS code.

3.3. Experimental results

3.3.1. Performance analysis

In order to analyze the performance of our algorithms we quantify the processing power of the heterogeneous distributed environment as a number of virtual processors. The fastest processor in the system is considered to be one *virtual processor*. Let S be the set of processors used in a distributed system configuration, s_i be the clock speed of processor i, $i \in S$, and $s_{\max}$ the clock speed of the fastest processor. Slower processors are considered to be fractions of one virtual processor. We define the number of virtual processors as

$$Vp = \frac{\sum_{i\in S} s_i}{s_{\max}}. \tag{1}$$

For example, if we have three Pentium PCs (450 MHz, 128 MB), three SUN Ultra-10 (440 MHz, 128 MB) workstations, one SGI-O2 (270 MHz, 128 MB) and two SGI-O2 (200 MHz, 64 MB) workstations the number of virtual processors is $Vp = \frac{3\times450+3\times440+1\times270+2\times200}{450} = 7.42$.

We use the following notations:

- p—number of workstations;
- Vp—number of virtual processors;
- T_{comp}—computation time per integration step;
- T_{comm}—communication time per integration step;
- T_p—execution time per integration step on p workstations, $T_p = T_{\text{comp}} + T_{\text{comm}}$;

In order to compare the efficiency of our load balancer we used as a base line the execution time obtained using an equal allocation.

We computed the speedup according to the following equation [8]:

$$S_p = \frac{\min\{T_{P1}, T_{P2}, \ldots, T_{Pp}\}}{T_p}, \tag{2}$$

where T_{Pi} is the execution time per integration step on workstation Pi.

3.3.2. Tests and results

In this section we present the experimental results for the proposed scheduling algorithms. We ran the code for two commonly used methods: (1) LU-SGS and (2) OSOmin (see the appendix), for several different configurations of workstations. The dimensions of the problem grid are: $J = 135$, $K = 50$ and $L = 35$. The results of these runs are presented in Table 1.

We present in Tables 2 and 3 the execution times for two configurations resulting from the equal and balanced allocations. In both cases Algorithm II chooses a configuration with a smaller number of PEs. For example in Table 2, 15 slow PEs are excluded which reduces the execution times for LU-SGS and OSOmin.

Fig. 6 shows the speedup in LU-SGS execution time for equal and balanced load allocation (using Algorithm II). In general, using the balanced allocation we obtained a much better speedup than using the equal allocation. For example, using balanced allocation the speedup is 5.58, 11.16 and 16.74 and using the equal

Table 1
Execution and communication time per integration step for LU-SGS and OSOmin using equal and load balanced allocation

p (Vp)	PE mesh	Allocation method	LU-SGS	OSOmin
			T_{comm}/T_p(s)	T_{comm}/T_p(s)
1 (1)	1×1	—	—/16.74	—/52.02
9	3×3	Equal	3.2/5.2	11.9/17.2
(7.5)		Balanced	1.5/3.1	5.09/9.83
15	3×5	Equal	2.5/3.5	8.2/11.6
(11.2)		Balanced	1.5/2.3	3.5/7.4
28	7×4	Equal	1.5/2.1	6.1/7.8
(21.7)		Balanced	0.9/1.5	3.2/4.3
33	11×3	Equal	1.7/2.1	4.5/5.9
(23.6)		Balanced	1.2/1.5	3.23/4.09
35	7×5	Equal	0.7/1.1	1.6/3.0
(35.0)		Balanced	0.7/1.1	1.6/3.0
60	15×4	Equal	1.2/2.0	3.6/4.4
(52.9)		Balanced	0.8/0.9	2.13/2.7

allocation the speedup is 3.22, 7.97 and 8.37, when the number of virtual processors is 7.5, 21.7 and 52.9. We obtained the same speedup for both allocations at $Vp = 35.0$. This is because all 35 workstations included in this configuration are of the same speed (i.e. homogeneous system). For all configurations the balanced allocation achieved a speedup from 5.58 to 16.74. The gain in speedup for LU-SGS using balanced allocation is from 40% to 100% and this is expected to increase with the number of virtual processors.

Fig. 7 shows the speedup for OSOmin for both equal and balanced allocations. As in the case of LU-SGS, we obtained similar speedup curves showing that the balanced allocation achieves a much higher speedup than the equal allocation. For example, using balanced allocation the speedup is 5.29, 12.09 and 18.40 and using the equal allocation the speedup is 3.02, 6.69 and 11.80, when the number of virtual processors is 7.5, 21.7 and 52.9. The speedup is the same for both types of allocations at $Vp = 35$ because the workstations included in this configuration are of the same speed and the balanced allocation is the same as the equal allocation. The achieved speedup is from 5.29 to 18.40. It can be observed that for the same size grid the speedup using balanced allocation increases with the number of virtual processors providing good scalability. The gain in speedup for OSOmin using balanced allocation is from 44% to 80% and this is expected to increase with the number of virtual processors. However, this is not true for equal allocation as it can be seen in Figs. 6 and 7.

In Table 1 we also show the communication time for all configurations and allocation methods. A high percentage of the execution time is due to the communication, since we keep the problem size fixed. The performance gains due to our algorithm will be more significant in heterogeneous clusters with fast communication.

Table 2
Execution times for $Vp = 35.0$ and 41.0

p (Vp)	PE mesh	Allocation method	LU-SGS	OSOmin
			T_{comm}/T_p(s)	T_{comm}/T_p(s)
35	7×5	Equal	0.7/1.1	1.6/3.0
(35.0)		Balanced	0.7/1.1	1.6/3.0
50	5×10	Equal	1.6/2.0	4.4/5.3
(41.0)		Balanced	1.1/1.3	2.8/4.2

Table 3
Execution times for $Vp = 52.9$ and 54.0

p (Vp)	PE mesh	Allocation method	LU-SGS	OSOmin
			T_{comm}/T_p(s)	T_{comm}/T_p(s)
60	15×4	Equal	1.2/2.0	3.6/4.4
(52.9)		Balanced	0.8/0.9	2.13/2.7
63	21×3	Equal	1.25/2.1	3.2/3.9
(54.0)		Balanced	0.74/0.9	2.49/2.89

3.3.3. *Load balancing considering memory capacity*

Our load balancer incorporates a technique that takes into consideration the memory capacity of each machine in making the allocation decision. This technique is based on the following assumptions: (i) The allocated grid points to some PEs in the system exceed their local available memory. (ii) The final configuration (with reduced number of PEs) given by the algorithm will accommodate all grid points in the local memory. We consider systems for which the PEs with a higher speed have also a larger memory. This technique works as follows. At first the load balancer allocates the grid points according to Algorithm II and then checks each PE (in order from the slowest to the fastest) if the allocation exceeds its memory capacity. If assumptions (i) and (ii) are not true then the algorithm will execute with the old allocation. The PEs which have their memory capacity exceeded are eliminated from the distributed system and we get a lower p in Algorithm II. Then a new allocation is computed using Algorithm II. As an example we considered the $q = 63$ PEs case in which the load balancer, without taking into account the memory capacity, suggests $p = 60$ PEs as the best configuration. We ran the load balancer considering the memory capacity and in this case it excluded 18 processors. By not using 18 PEs, the number of PEs becomes $p = 45$ and the execution time for OSOmin is reduced from 2.7 to 2.5 s. For LU-SGS we obtained the same execution time but with fewer PEs. These results are shown in Table 4.

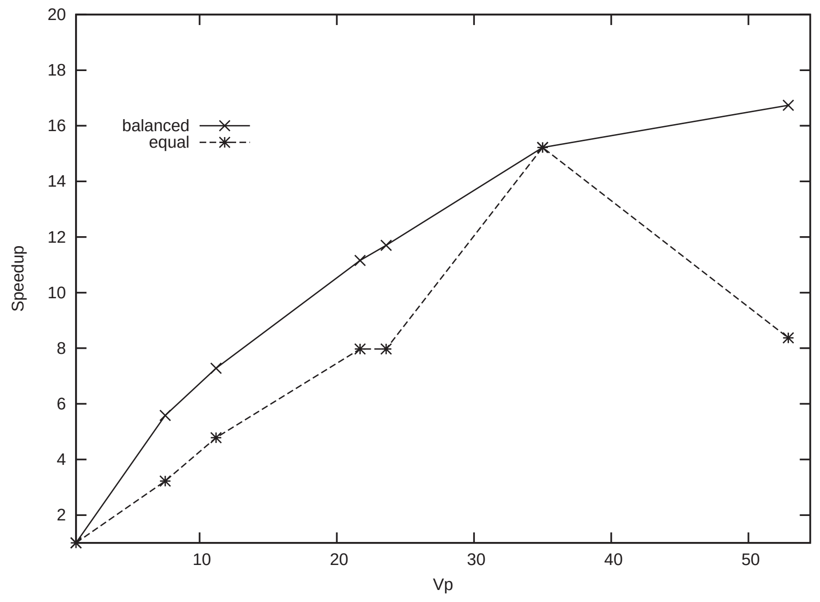


Fig. 6. Speedup vs. number of virtual processors for LU-SGS.

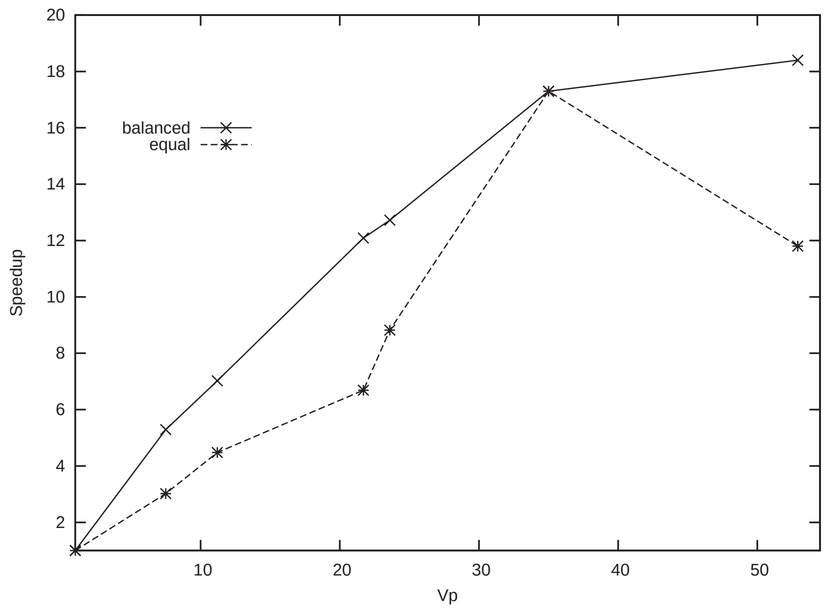


Fig. 7. Speedup vs. number of virtual processors for OSOmin.

4. Conclusion and future work

In distributed simulations, the delivered performance of networks of heterogeneous computers degrades severely if the computations are not load balanced. This article deals with 3D grid domain partitioning, subdomain mapping and associated computation scheduling for heterogeneous distributed systems for a CFD application in helicopter aerodynamics. We derived new algorithms for grid partitioning taking into account processor speed and memory capacity. We implemented our algorithms on a network of workstations (using MPI) and tested it with the CFD simulation code. Test run comparisons of these algorithms demonstrated significant efficiency gains. The importance of our results is that they can be applied to other CFD simulation codes. We base our system performance on the individual processor clock rates. Other more reliable measurements of performance could be used in future work.

Table 4
Execution time per integration step for LU-SGS and OSOmin using load balanced allocation

Configuration	Memory taken into account	$T_p(s)$	
		LU-SGS	OSOmin
$63->60$	No	0.9	2.7
$63->45$	Yes	0.9	2.5

Acknowledgments

This research was supported, in part, by research grants from (1) NASA NAG 2-1383 (1999–2000). (2) State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999. (3) This research was also supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the University of California San Diego. (4) The third author was supported by a NASA Graduate Student Fellowship while the majority of this work was performed and is currently employed at the University of California Lawrence Livermore National Laboratory under US Department of Energy contract number W-7405-Eng-48.

Some reviewers' comments helped enhance the quality of presentation.

Appendix. A testbed CFD simulation code

Accurate numerical simulation of the aerodynamics and aeroacoustic of rotary-wing aircraft is a complex and challenging problem. Three-dimensional unsteady Euler/Navier–Stokes computational fluid dynamics (CFD) methods are widely used (see [18] and references therein), but their application to large problems is limited by the amount of computer time they require. Such an example of a CFD application, which we will focus on, is the computation of a helicopter aerodynamics. Efficient utilization of parallel processing is one effective means of speeding up these calculations [20]. The baseline numerical method is the structured-grid Euler/Navier–Stokes solver TURNS (Transonic Unsteady Rotor Navier Stokes) (see [18] and references therein) developed in conjunction with the US Army Aeroflightdynamics Directorate at NASA Ames Research Center. It is used for calculating the flowfield of a helicopter rotor (without fuselage) in hover and forward flight conditions. The governing equations solved by the TURNS code are the three-dimensional unsteady compressible thin-layer Navier–Stokes equations, applied in conservative form in a generalized body-fitted curvilinear coordinate system. The implicit operator used in TURNS for time-stepping in both steady and unsteady calculations is the Lower–Upper symmetric Gauss–Seidel (LU-SGS) operator of Yoon and Jameson [23].

The parallel TURNS version uses the Hybrid LU-SGS which is a parallel modification to LU-SGS [14]. Once the computational space has been divided into subdomains, the original LU-SGS algorithm is applied simultaneously to each processor subdomain. Then, border data between the subdomains is communicated using the relaxation-type approach of DP-LUR [20]. The use of multiple relaxation sweeps is retained to enhance robustness of the original algorithm lost in the domain decomposition. On a single processor, hybrid LU-SGS is identical to the original LU-SGS algorithm. On many processors (in the limit as the number of processors approaches the number of gridpoints), the algorithm becomes identical to DP-LUR (see [20] and references therein). Like DP-LUR, hybrid LU-SGS can be implemented such that it is completely load balanced with only nearest-neighbor communication required between the subdomains. Hybrid LU-SGS can be implemented such that it is completely load balanced with only nearest-neighbor communication between the sub-domains. In tests with transonic and supersonic problems with up to 512 subdomains, the hybrid LU-SGS method converges with a single relaxation sweep but the convergence rate is less than that of original LU-SGS. With two relaxation sweeps, the convergence rate is essentially identical to original LU-SGS. Further details of the hybrid LU-SGS algorithm are given in [20].

Inexact Newton methods coupled with Conjugate Gradient (CG)-type iterative methods for nonsymmetric linear systems have also been used. Many authors studied the CG-type methods for CFD applications (see [1,11] and references there in). CG-type methods like GMRES were shown to be most efficient. Wissink implemented the CG-type iterative methods GMRES and OSOmin [3,21]. The two methods gave similar convergence and performance results. Storage is a major consideration for the solution of three-dimensional problems. The Jacobian matrix is not computed because CG-type methods use only Jacobian times vector products which are approximated by Taylor expansion. The CG-type methods require a modest amount of extra storage compared to the storage required by the LU-SGS method [20]. The LU-SGS method is used as a preconditioner in these iterative methods to speed up their convergence rate. For more details about CG-type methods and preconditioning see [6].

We now review the parallel implementation of TURNS for a parallel system with homogeneous processors [21]. The time stepping is serial. The three-dimensional flowfield spatial domain is divided in the wraparound and spanwise directions to form a two-dimensional array of processor subdomains, as shown in Fig. 1. Each processor executes a version of the code simultaneously for the portion of the flowfield that it holds. Coordinates are assigned to the processors to determine global values of the data each holds. Border data is communicated between processors, and a single layer of ghost-cells stores this communicated data. The Message Passing Interface (MPI) software routes communication between the processor subdomains.

TURNS approximates the solution at each time step based on two alternatives: (a) the relaxation (DP-LUR or LU-SGS) methods described above, or (b) the iterative (Inexact Newton OSOmin method). There are essentially four main steps of the inexact Newton algorithm [21]; (1) explicit (flux) function evaluation to form the right-hand side vector, (2) preconditioning using hybrid LU-SGS (explained above), (3) implicit solution by the iterative solver, and (4) explicit application of boundary conditions. The (Jacobian-free) matrix multiplications are based on function evaluations in (3). Local processor communication is required in (1)–(4). We also have global communications in the error computation at each timestep and in the dotproducts in the Krylov methods.

The parallel implementation of TURNS with hybrid LU-SGS and OSOmin was performed on the IBM SP. Each processor was assigned a grid subdomain with equal number of grid points [21]. We now consider the implementation on a network of heterogeneous workstations. To deal with the heterogeneity of the processors we consider subdividing the space domain into subdomains with unequal number of grid points.

References

[1] K. Ajmani, M.S. Liou, R.W. Dyson, Preconditioned implicit solvers for the Navier Stokes equations on distributed-memory machines, AIAA Paper 94-0408.

[2] C.A. Bohn, G.B. Lamont, Asymmetric load balancing on a heterogeneous cluster of PC, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 1999, pp. 2512–2522.

[3] A.T. Chronopoulos, C.D. Swanson, Parallel iterative S-step methods for unsymmetric linear systems, Parallel Comput. 22 (5) (1996) 623–641.

[4] P.E. Crandall, M.J. Quinn, Non-uniform 2-D grid partitioning for heterogeneous parallel architectures, in: Proceedings of the 9th International Parallel Processing Symposium, Santa Barbara, CA, 1995, pp. 428–435.

[5] T. Decker, R. Luling, S. Tschoke, A distributed load balancing load balancing algorithm for heterogeneous parallel computing systems, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 1998, pp. 933–940.

[6] J. Dongarra, I. Duff, D. Sorensen, H. van der Vorst, Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, 1998.

[7] W.D. Gropp, E. Lusk, User's Guide for mpich, a Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, aNL-96/6, 1996.

[8] D. Grosu, Some performance metrics for heterogeneous distributed systems, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. 5, Sunnyvale, CA, 1996, pp. 1261–1268.

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

[10] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distributed Comput. 48 (1998) 71–95.

[11] P.R. McHugh, D.A. Knoll, Comparison of standard and matrix-free implementations of several Newton-Krylov solvers, AIAA J. 32 (12) (1994) 2394–2400.

[12] G.L. Miller, S.H. Teng, S.A. Vavasis, A unified geometric approach to graph separators, in: Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 538–547.

[13] D.M. Nicol, Rectilinear partitioning of irregular data parallel computations, J. Parallel Distributed Systems 23 (1994) 119–134.

[14] A. Pothen, H.D. Simon, L. Wang, S.T. Bernard, Towards a fast implementation of spectral nested dissection, in: Proceedings of Supercomputing'92, Minneapolis, MN, 1992, pp. 42–51.

[15] K. Schloegel, G. Karypis, V. Kumar, Graph partitioning for high performance scientific simulations, in: J. Dongarra, I. Foster, G. Fox, K. Kennedy, L. Torczon, A. White (Eds.), CRPC Parallel Computing Handbook, Morgan Kaufman, San Francisco, 2001.

[16] K. Schloegel, G. Karypis, V. Kumar, Graph partitioning for dynamic, adaptive and multi-phase scientific simulations, in: Proceedings of the 3rd IEEE International Conference on Cluster Computing (CLUSTER'01), Newport Beach, CA, 2001, pp. 271–273.

[17] Q. Snell, G. Judd, M. Clement, Load balancing in a heterogeneous supercomputing environment, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 1998, pp. 951–957.

[18] G.R. Srinivasan, J. Baeder, TURNS: A Free-Wake Euler/Navier-Stokes numerical method for helicopter rotors, AIAA J. 31 (5) (1993) 959–962.

[19] T. Sterling, T. Cwik, D. Becker, J. Salmon, M. Warren, B. Nitzberg, An assessment of Beowulf-Class computing for NASA requirements: initial findings from the first NASA workshop on Beowulf-Class clustered computing, in: Proceedings of the IEEE Aerospace Conference, Aspen, CO, 1998.

[20] A.W. Wissink, A.S. Lyrintzis, R.C. Strawn, Parallelization of a three-dimensional flow solver for Euler rotorcraft aerodynamics predictions, AIAA J. 34 (11) (1996) 2276–2283.

[21] A.W. Wissink, A.S. Lyrintzis, A.T. Chronopoulos, A parallel Newton-Krylov method for rotary-wing flowfield calculations, AIAA J. 37 (10) (1999) 1213–1221.

[22] J. Xu, A.T. Chronopoulos, Distributed self-scheduling for heterogeneous workstation clusters, in: Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing Systems, Ft. Lauderdale, FL, 1999, pp. 211–217.

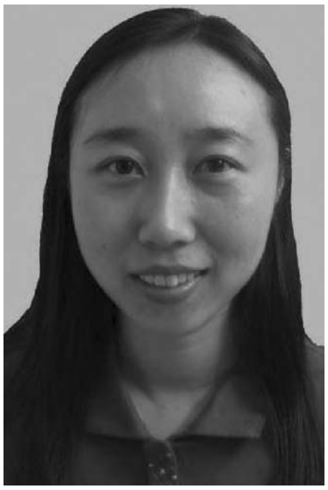
[23] S. Yoon, A. Jameson, A lower-upper symmetric Gauss Seidel method for the Euler and Navier Stokes equations, AIAA J. 26 (1988) 1025–1026.

Anthony T. Chronopoulos is an associate professor at The University of Texas at San Antonio. He received his Ph.D. at the University of Illinois in Urbana-Champaign in 1987. He is a senior member of IEEE. He has published 32 journal and 35 refereed conference proceedings publications in the areas of scientific computation, parallel and distributed computing. He has been awarded 12 federal/state government research grants. His work is cited in more than 145 non-co-authors' research articles.

Daniel Grosu is a Ph.D. candidate in the Department of Computer Science at The University of Texas at San Antonio. He received the Diploma in engineering (Automatic Control and Industrial Informatics) from the Technical University of Iasi, Romania in 1994 and the M.Sc. in computer science from The University of Texas at San Antonio in 2002. His research interests include load balancing, distributed systems and topics at the border of computer science, game theory and economics. He is a student member of the IEEE and the ACM.

Andrew M. Wissink received his Ph.D. in aerospace engineering from the University of Minnesota in 1997. He worked at NASA Ames from 1996 to 1999 on dynamic overset grid methods. In 1999, he joined Lawrence Livermore National Laboratory at the Center for Applied Scientific Computing as a senior researcher. He is currently, investigating adaptive mesh refinement algorithms on parallel computer systems.

Manuel Benche received his BS in computer science from Transylvania University of Brasov, Romania in 1998 and his M.Sc. in computer science from The University of Texas at San Antonio in 2000.



Jingyu Liu received her M.Sc. degree in applied physics at the Beijing Institute of Technology, China in 1997 and her M.Sc. in computer science from The University of Texas at San Antonio in 2002.