Static Load Balancing for CFD Simulations on a Network of Workstations *

Anthony T. Chronopoulos, Daniel Grosu
Div. of Computer Science, Univ. of Texas at San Antonio,
6900 N Loop 1604 W, San Antonio, TX 78249
{atc, dgrosu}@cs.utsa.edu

Andrew M. Wissink
Center for Applied Scientific Computing,
Lawrence Livermore National Lab,
P.O. Box 808, L-661, Livermore, CA 94551

Manuel Benche
Div. of Computer Science,
Univ. of Texas at San Antonio,
6900 N Loop 1604 W, San Antonio, TX 78249

Abstract

In distributed simulations, the delivered performance of networks of heterogeneous computers degrades severely if the computations are not load balanced. In this work we consider the distributed simulation of TURNS (Transonic Unsteady Rotor Navier Stokes), a 3-D space CFD code. We propose a load balancing heuristic for simulations on networks of heterogeneous workstations. Our algorithm takes into account the CPU speed and memory capacity of the workstations. Test run comparisons with the equal task allocation algorithm demonstrated significant efficiency gains.

1 Introduction

Accurate numerical simulation of the aerodynamics and aeroacoustics 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 [5] the 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 [7]. The baseline numerical method is the structured-grid Euler/Navier-Stokes solver TURNS (Transonic Unsteady Rotor Navier Stokes) [5] developed in conjunction with the U.S. 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 [8].

We now review the parallel implementation of TURNS for a parallel system with homogeneous processors [6]. 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 Figure 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 in [7], or (b) the Inexact Newton Krylov methods. There are essentially four main steps of the inexact Newton algorithm (OSGCR) [6]; (1) explicit (flux) function evaluation to form the right-hand-side vector, (2) preconditioning using hybrid LU-SGS [7], (3) implicit solution by the Krylov subspace solver, and (4) ex-

*This research was supported, in part, by research grants from (1) NASA NAG 2-1383 (1999-2000), (2) Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999 (3) Air Force grant F49620-96-1-0472 (4) NSF cooperative agreement ACI-9619020 through computing resources provided by NPACI at the Univ. of California San Diego. The third author was supported by a NASA Graduate Student Fellowship while the majority of this work was performed.

0-7695-1432-4/01 $10.00 © 2001 IEEE

plicit application of boundary conditions. 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 OSGCR was performed on the IBM SP. Each processor was assigned a grid subdomain with equal number of grid points [6]. To deal with the heterogeneity of the processors we now consider subdividing the space domain into subdomains with unequal number of grid points.

We propose a load balancing algorithm that finds an optimal configuration of workstations minimizing the execution time. It also takes into consideration the memory size of each workstation in making the allocation decision.

Using our load balancing algorithm we were able to obtain an improvement in the speedup between 40% and 68% for LUSGS and 13% and 81% for OSGCR, compared with the equal allocation method.

The remainder of this paper is organized as follows. In Section 2 we present a load balancing algorithm for heterogeneous systems. In section 3 we present the implementation and we discuss experimental results.

2 The Load Balancing Algorithm

Assumptions:

1. We parallelize only the problem space domain and leave the time dimension for serial execution.
2. We assume a 2-D logical PE mesh with $p = r \times c$ PEs (r/c = number of PEs per row/column of the PE mesh), (Figure 1).
3. A *load* is measured as a 3-D box of grid points in the space domain.
4. We assume that the PEs of the parallel system are of different designs and speeds.
5. Our goal is to assign loads to different PEs such that the execution time is minimized.

In mapping the domain of $J \times K \times L$ grid points to a logical 2-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 $P_{*,j}$ and $P_{*,J-j}$. For example this is only possible when J is odd.

(ii) The L dimension is not divided at all. We only partition the $J \times K$ mesh and assign $\frac{JK}{p}L$ grid points to each PE. Each PE has only four adjacent PEs.

(iii) 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.

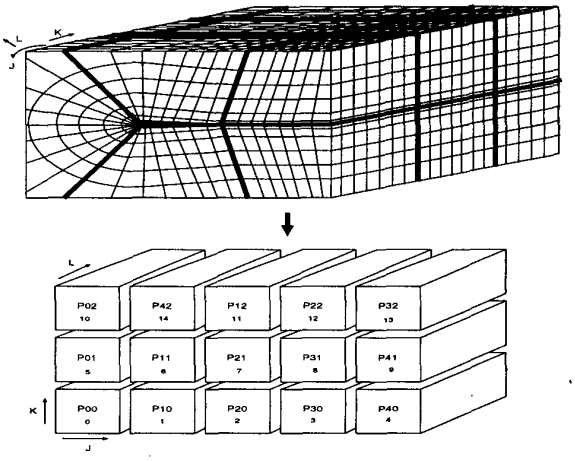

Figure 1. Partitioning the three-dimensional domain on a two-dimensional array of processors.

In a mapping we may have four types of processor loads: $J_{load} \times K_{load} \times L$, $(J_{load}+1) \times K_{load} \times L$, $J_{load} \times (K_{load}+1) \times L$ and $(J_{load}+1) \times (K_{load}+1) \times L$ grid points (see Figure 2), where: $J_{load} = \lfloor (J-2)/c+2 \rfloor$, $K_{load} = \lfloor (K-2)/r+2 \rfloor$, $L_{load} = L$.

We use as an estimate of the total execution time (T_{est}), the load corresponding to the largest value of these four loads. The goal is to minimize this value, by finding an optimal configuration $p = r \times c$ of processors. Figure 2 is an example with $p = r \times c = 3 \times 5$.

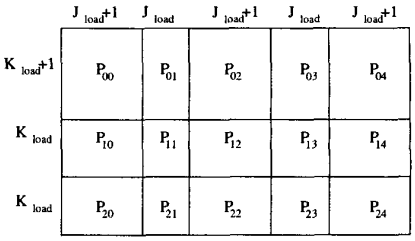

Figure 2. The loads for 3×5 processors configuration .

Given a number of processors $p = r \times c$, we divide the $J \times K$ sized grid in p rectangular partitions, with the J dimension divided in c subpartitions and the K dimension divided in r subpartitions. These subpartitions are possibly of unequal size.

We sort the PEs $(P_1, P_2, \ldots, P_p)$ in non-increasing order of their processing powers: $speed(P_0) \geq speed(P_1) \geq$

$\ldots \geq speed(P_p)$ and assign the subpartition (j,k) to processor $P_{k\times J+j}$.

We expect the processors of a PE row to have approximately the same computing power. Thus, we divide the J dimension into equal subpartitions. The K direction is divided according to the power of each PE row, where the power of a PE row is defined as the power of the slowest PE on that row. This might result in slightly under-utilizing a more powerful processor, but avoids the more important issue of overloading a weak one.

Every PE is expected to complete execution in time proportional to the ratio of its load over processing power. We use as T_{est} the maximum of these ratios. The goal is to find a configuration (r,c) of PEs that minimizes T_{est}.

Our algorithm checks every possible factorization $p = r \times c$ of p PEs and proposes an optimal configuration with minimum T_{est} over all configurations under our assumptions. This is a suboptimal solution to the general grid partitioning problem (see [1, 4] and references therein).

Note that it is possible that a configuration with a smaller number of processors can produce a smaller estimate value. The algorithm starts from the number q of available PEs and returns p PEs (where $p = r \times c$ and $p \leq q$) such that $(r,c) = \text{argmin}\ T_{est}$. Our heuristic approach checks these configurations by decrementing q by one at each stage.

Algorithm:
sort PEs $P_0 \ldots P_{p-1}$ in non-increasing order so that
  $speed(P_0) \geq speed(P_1) \geq \ldots \geq speed(P_{p-1})$
for $p = q$ downto 1 **do**
  for all (r,c) where $p = r \times c$ **do**
    $\min_{(r,c)} T_{est}(r,c)$
return $p = r \times c$ = argmin T_{est}
Function $T_{est}(r,c)$
  $a = \lfloor (J-2)/c + 2) \rfloor$
  $a_r = (J-2) \bmod c$
  if ((J is odd) **and** (c is even))
    return error("symmetry violated")
  else
    {Obtain the minimum PE speed for each k-th PE row}
    $row\_speed(k) = \min_{j=0\ldots c-1} speed(P_{k,j})$
    {Compute total of all speeds}
    $total\_speed = \sum_{k=0}^{r-1} row\_speed(k)$
    {Map the J direction to the c PEs of the k-th PE row }
    if ($a_r \neq 0$ **and** a_r is odd)
      map $J_{load} = (a+1)$ points to a_r PEs:
        $P_{k,0}, \ldots, P_{k,\frac{(a_r-1)}{2}}, P_{k,c/2}, P_{k,\frac{c-(a_r-1)}{2}}, \ldots, P_{k,c}$.
      map $J_{load} = a$ points to remaining $(c - a_r)$ PEs
    else a_r is even
      map $J_{load} = (a+1)$ points to a_r PEs:
        $P_{k,0}, \ldots, P_{k,\frac{a_r}{2}}, P_{k,\frac{c-a_r}{2}}, \ldots, P_{k,c-1}$.
      map $J_{load} = a$ points to remaining $(c - a_r)$ PEs
    {Map the K direction to the r PEs of the j-th PE column }
    $b(k) = \frac{row\_speed(k)}{total\_speed} \times K$
    $rem = K - \sum_{k=0}^{r-1} \lfloor b(k) \rfloor$
    map $K_{load} = (\lfloor b(k) \rfloor + 1)$ points to rem PEs:
      $P_{0,j}, \ldots, P_{rem-1,j}$.
    map $K_{load} = \lfloor b(k) \rfloor$ points to remaining $(r - rem)$ PEs.
    {Compute T_{est}}
    if ($a_r \neq 0$)
      $J_{load} = a + 1$
    else
      $J_{load} = a$
    if ($rem \neq 0$)
      $K_{load} = \max_{k=0,\ldots,rem-1} (\lfloor b(k) \rfloor + 1)$
    else
      $K_{load} = \max_{k=0,\ldots,r-1} (\lfloor b(k) \rfloor)$
  return $T_{est} = J_{load} \times K_{load}$

3 Implementation and results

In our experiments, we use a heterogeneous network of workstations which includes 48 SUN Ultra-10(440Mhz, 128MB), 12 SUN Ultra-1 (166Mhz, 64MB), one SGI-O2 (270Mhz, 128MB) and two SGI-O2 (200Mhz, 64MB). The SUN Ultra-10 and SGI workstations are connected to each other via a 100Mb/s switched Ethernet. All the other workstations are connected to each other via a 10Mb/s switched Ethernet. As a message-passing library we use the MPICH 1.2.0 [2]. For compiling the TURNS code on SUN workstations we use the Sun WorkShop Compiler FORTRAN 90 SPARC Version 2.0 and for SGI workstations we use the MIPS Pro FORTRAN 90 compiler.

In order to analyse the performance of our algorithm we quantify the processing power of the heterogeneous distributed environment as a number of virtual processors (VP). One virtual processor is defined as the fastest processor in the system. In our case one virtual processor is equivalent to a SUN Ultra-10 (440Mhz, 128MB) workstation. For example if we have six SUN Ultra-10 (440Mhz, 128MB) workstations, one SGI-O2 (270Mhz, 128MB) and two SGI-O2 (200Mhz, 64MB) workstations the number of virtual processors is $VP = 7.5$.

We use the following notations: p - number of workstations; T_{comp} - computation time per integration step; T_{com} - communication time per integration step; T_p - execution time per integration step, $T_p = T_{comp} + T_{com}$.

We run the code for both methods LUSGS and OSGCR, and different configurations of workstations. The results of these runs are presented in Table 1. In order to compare the efficiency of our load balancer we used as a base line the execution time obtained using an equal allocation. The speedup [3], using equal and balanced allocation for the OSGCR method is presented in Figure 3.

Having a number of different workstations on our system the load balancer is able to determine the number of workstations for which we can obtain maximum performance. In our experiments the optimal configuration for a total of

p (Vp)	PE mesh	Allocation method	LUSGS T_{com}/T_p(sec)	OSGCR T_{com}/T_p(sec)
1 (1)	1x1	-	-/16.74	-/52.02
9 (7.5)	3x3	equal	3.2 / 5.2	11.9 / 17.2
		balanced	1.5 / 3.1	5.09 / 9.83
15 (11.2)	3x5	equal	2.5 / 3.5	8.2 / 11.6
		balanced	1.5 / 2.3	3.5 / 7.4
28 (21.7)	7x4	equal	1.5 / 2.1	6.1 / 7.8
		balanced	0.9 / 1.5	3.2 / 4.3
33 (23.6)	11x3	equal	1.7 / 2.1	4.5 / 5.9
		balanced	1.2 / 1.5	3.23 / 4.09
35 (35.0)	7x5	equal	0.7 / 1.1	2.13 / 3.40
		balanced	0.7 / 1.1	1.6 / 3.0
50 (41.0)	5x10	equal	1.6 / 2.0	4.4 / 5.3
		balanced	1.1 / 1.3	2.8 / 4.2
60 (52.9)	15x4	equal	1.2 / 2.0	3.6 / 4.4
		balanced	0.8 / 0.9	2.13 / 2.7
63 (54.0)	21x3	equal	1.25 / 2.1	3.2 / 3.9
		balanced	0.74 / 0.9	2.49 / 2.89

Table 1. Execution and communication time per integration step for LUSGS and OSGCR.

63 workstations was determined by the load balancer to be formed by 60 workstation out of 63. In this case some of the slowest workstations were eliminated (by the load balancer) from the configuration (see Table 1) because their inclusion would lead to worse performance.

An important percentage of the execution time is due to the communication time. In Table 1 we shown the communication time for all configurations and allocation methods.

Our load balancer takes into consideration the memory size of each machine in making the allocation decision. First the load balancer allocates the grid points according to our algoritm. We have added a technique in the load balancer which checks if the allocation exceeds the memory size of a machine. Such machines are eliminated from the distributed system and we get a lower q. Then a new allocation is computed using our algorithm. As an example we considered the q=63 processors case in which the load balancer without taking into account the memory size suggests p=60 processors system as the best configuration. We run the load balancer considering the memory size and in this case it excludes 18 processors. By not utilizing 18 workstations, the number of workstations becomes p=45 and the execution time for OSGCR is reduced from 2.7 seconds to 2.5 seconds. These results are shown in Table 2. As the table shows, the memory limitation appears only in the case of OSGCR method.

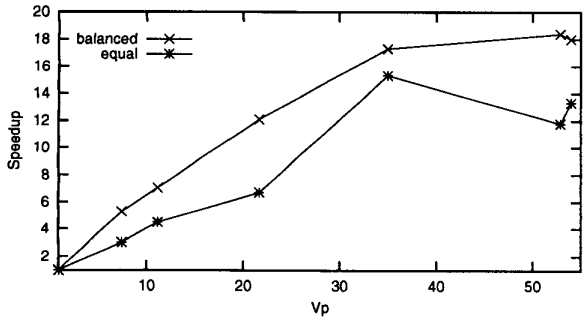

Figure 3. Speedup vs. number of virtual processors for OSGCR.

Configuration	Memory taken into account	T_p(sec.)	
		LUSGS	OSGCR
$63 -> 60$	No	0.9	2.7
$63 -> 45$	Yes	0.9	2.5

Table 2. Execution time for LUSGS and OSGCR using balanced allocation.

References

[1] P. E. Crandall and M. J. Quinn. Non-Uniform 2-D Grid Partitioning for Heterogeneous Parallel Architectures. In *Proc. of the 9th Intl. Parallel Proc. Symp.*, pages 428–435, April 1995.

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

[3] D. Grosu. Some Performance Metrics for Heterogeneous Distributed Systems. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, volume 5, pages 1261–1268, August 1996.

[4] D. M. Nicol. Rectilinear Partitioning of Irregular Data Parallel Computations. *J. of Parallel and Distributed Systems*, 23:119–134, 1994.

[5] G. R. Srinivasan and J.D. Baeder. TURNS: A Free-Wake Euler/Navier-Stokes Numerical Method for Helicopter Rotors. *AIAA Journal*, 31(5):959–962, May 1993.

[6] A. W. Wissink, A. S. Lyrintzis, and A. T. Chronopoulos. A Parallel Newton-Krylov Method for Rotary-wing Flowfield Calculations. *AIAA Journal*, 37(10):1213–1221, October 1999.

[7] A. W. Wissink, A. S. Lyrintzis, and R. C. Strawn. Parallelization of a Three-Dimensional Flow Solver for Euler Rotorcraft Aerodynamics Predictions. *AIAA Journal*, 34(11):2276–2283, November 1996.

[8] S. Yoon and A. Jameson. A Lower-Upper Symmetric Gauss Seidel Method for the Euler and Navier Stokes Equations. *AIAA Journal*, 26:1025–1026, 1988.