Porting CFD Codes towards Grids. A Case Study

Dana Petcu[1,2], Daniel Vizman[3] and Marcin Paprzycki[4]

[1] Computer Science Department, Western University,[2] Institute e-Austria, Timişoara,
[3] Physics Department, Western University of Timişoara, Romania
[4] Computer Sciecne Institute, SWPS, Warsaw, Poland
petcu@info.uvt.ro, vizman@physics.uvt.ro, marcin.paprzycki@swps.edu.pl

Abstract. In this paper we discuss an application of a modified version of a graph partitioning-based heuristic load-balancing algorithm known as the *Largest Task First with Minimum Finish Time and Available Communication Costs*, which is a part of the *EVAH* package. The proposed modification takes into account the dynamic nature and heterogeneity of grid environments. The new algorithm is applied to facilitate load balance of a known CFD code used to model crystal growth.

1 Introduction

Computational fluid dynamics codes are computationally demanding, both in terms of memory usage and also in the total number of arithmetical operations. Since the most natural methods of improving accuracy of a solution are (1) refining a mesh or/and (2) shortening the time step, either of these approaches result in substantial further increase of both computational cost and total memory usage. Therefore, a tendency can be observed, to use whatever computational resources are available to the user and, particularly among researchers working in the CFD area, there exist an almost insatiable demand for more powerful computers with ever increasing size of available memory. One of possible ways to satisfy this demand is to divide up the program to run on multiple processors. In the last twenty years several codes have been introduced to facilitate parallel computational fluid dynamics.

Parallel CFD codes have been typically developed assuming a homogeneous set of processors and a fast network. Recent ascent of grid technologies requires re-evaluation of these assumptions as the very idea of computational grids is based on combining heterogeneous processors and using substantially slower networks connections (typically the Internet or a corporate LAN). This makes the environment much different than parallel computers or clusters of workstations connected using a fast switch. Furthermore, the migration process of CFD codes designed for parallel computing architectures towards grids must take into account not only the heterogeneity of the new environment but also the dynamic evolution of the pool of available computational resources. At the same time we have to acknowledge that to be able to fully rewrite the existing codes is usually not a viable option (e.g. because of the cost involved in such an endeavor).

In this context let us note that large body of research has been already devoted to dynamic load balancing in heterogeneous environments, and more recently in grid environments (e.g. in [2] dynamic load-balancing in a grid environment is used for a geophysical application).

Approaches to load-balancing in distributed systems can be classified into three categories: (1) graph-theoretic, (2) mathematical programming based, and (3) heuristic. Graph-theoretic algorithms consider graphs representing the inter-task dependencies and apply graph partitioning methodologies to obtain approximately equal partitions of the graph such that the inter-node communication is minimized. A CFD simulation using such an approach is described in [5]. The mathematical programming method, views the load-balancing as an optimization problem and solves it using techniques originating from that domain. Finally, heuristic methods provide fast solutions (even though usually sub-optimal ones) when the time to obtain the exact optimal solution is prohibitive. For example in [1] a genetic algorithm is used as an iterative heuristic method to obtain near optimal solutions to a combinatorial optimization problem that is then applied to job scheduling on a grid.

When approached from a different perspective, we can divide load management approaches into (1) system level, and (2) user-level. A system-level centralized management strategy, which works over all applications uses schedulers to manage loads in the system. It is typically based on rules associated with job types or load classes. An example of the the user-level individual management of loads in a parallel computing environment is the *Dynamic Load Balancing* (*DLB* [8]) tool that lets the system balance loads without going through centralized load management and, furthermore, provides application level load balancing for individual parallel jobs. Here, a CFD test case was used as an example. System load measurement of the *DLB* is modified using average load history provided by computing systems rather than by tracking processing of tasks.

The *EVAH* package [3] was developed to predict the performance scalability of overset grid applications executing on large numbers of processors. In particular, it consists of a set of allocation heuristics that consider the constraints inherent in multi-block CFD problems.

In this paper we analyze and modify a graph partitioning-based heuristic algorithm available within the *EVAH* package, the *Largest Task First with Minimum Finish Time and Available Communication Costs* (*LTF_MFT_ACC*). The main drawback of this algorithm is that it assumes that grid-available resources are homogeneous. For instance, to show its efficiency tests performed on an Origin2000 system and were reported in [3]. We propose a modification of the *LTF_MFT_ACC* algorithm that can be applied in the case of a heterogeneous computing environment (such as a typical grid is supposed to be). Inspired by the *DLB* tool, our algorithm takes into account (1) the history of the computation time on different nodes, (2) the communication requirements, and (3) the current network speeds. To study the robustness of the proposed improvements, the modified *LTF_MFT_ACC* algorithm is used to port an existing CFD code into a heterogeneous computing environment.

The paper is organized as follows. Section 2 describes the CFD code and its parallel implementation. Section 3 presents the modified *LTF_MFT_ACC* algorithm. Section 4 discusses the results of our initial test.

2 Crystal growth simulation

Materials processing systems are often characterized by the presence of a number of distinct materials and phases with significantly different thermo-physical and transport properties. Understanding of the complex transport phenomena in these systems is of vital importance for the design and fabrication of various desired products as well as optimization and control of the manufacturing process. It is well known that numerical simulation prove to be an effective tool for the understanding of the transport mechanisms. However, in computational practice, three-dimensional simulations are necessary to yield a reliable description of the flow behavior. Usual computational methods applied to these problems include finite difference, finite volume, finite element, and spectral methods.

In particular, let us consider the Czochralski process [11] of bulk crystal growth that features a rod holding an oriented seed crystal which is lowered through the top surface of the molten liquid contained in a crucible. With thermal control to maintain the upper surface of the fluid at the melt temperature, growth begins on the seed and when the crystal reaches a specified diameter, the rod is slowly withdraw to continue growth (Figure 1.a). The flow in the melt, from which the crystal is pulled, is transient and, depending on the size of the crucible, mostly turbulent.

The silicon melt flow into a rotating crucible is governed by the three-dimensional partial differential equations describing mass, momentum, and heat transport. Solution methods that employ finite volume (see e.g [4]) require generation of the solution grid that conforms to the geometry of the flow region (a grid of small volume elements for which the average values of flow quantities are stored). An important issue for the quality of the numerical simulations is the choice of the (discretizing) grid. Here, both the numerical resolution and the internal structure of the grid are very important. The second item can be seen, is the refining of the grid towards the walls of the melt container, which is necessary to properly resolve the boundary layers of the flow.

As far as the solution was concerned, a matched multiblock method was used in our simulations (here, the grid lines match each other at the block conjunction). Here, a multiblock structured grid system [13] uses advanced linear solvers, for the inner iteration, and a multigrid technique for the outer iterations. Furthermore, the computational domain is divided into blocks consisting of control volumes (from hundreds to millions; see Figure 1.b).

More specifically, the finite volume code *STHAMAS 3D* (developed partially by the second author at the Institute of Materials Science in Erlangen) allows three-dimensional time-dependent simulations on a block-structured numerical grid. *SIP* (*Stone's strongly implicit procedure* [12]) is used to solve the system of linear equation resulting from the discretization of PDEs for three-dimensional

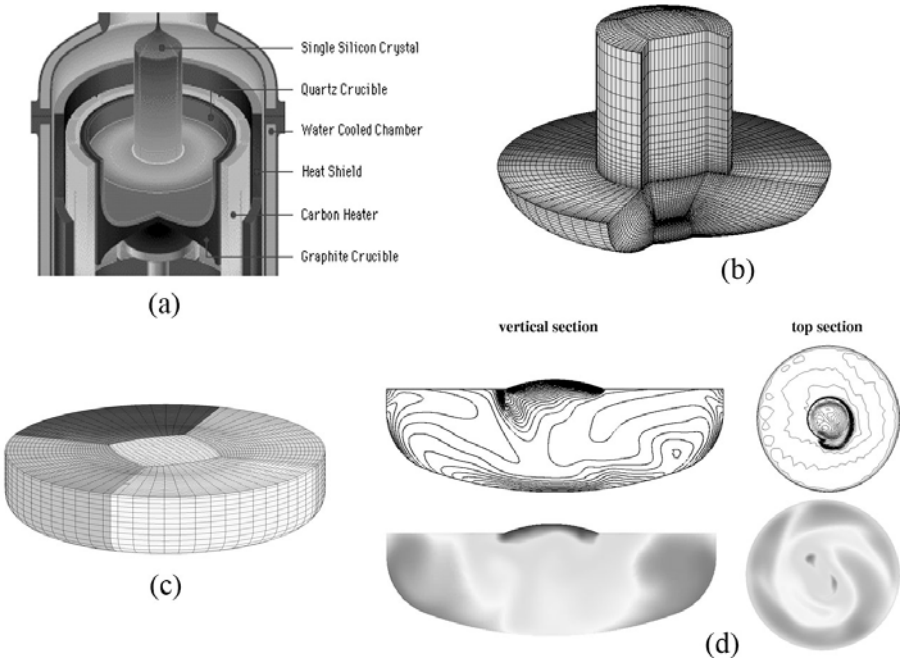

Fig. 1. Crystal growth: (a) device; (b) control volumes; (c) blocks of control volumes; (d) code outputs – isoterms and animation frames

problems (it is applicable to seven-diagonal coefficient matrices that are obtained when central-difference approximation is used to discretize the problem). *SIMPLE* algorithm (*Semi-Implicit Method for Pressure-Linked Equations*, [9]) is used for the pressure correction and the implicit Euler method is applied for time integration. The *SIP* and the *SIMPLE* were studied and compared (in [6]) with other solvers and shown to be very robust. A simple example of the graphical output of the code is presented in Figure 1.d.

Time-dependence and three-dimensionality coupled with extensive parameter variations require a very large amount of computational resources and result in very long solution times. The most time-consuming part of the sequential code *STHAMAS 3D* is the numerical solution obtained, using the SIP, on different blocks of CVs. In order to decrease the response time of *STHAMAS 3D*, a parallel version was recently developed by the first two authors and compared with other similar CFD codes (see [10]). It is based on a parallel version of the SIP solver, where simultaneous computations are performed on different blocks mapped to different processors (different colors in Figure 1.c). After each inner iteration, information exchanges are performed at the level of block surfaces (using the MPI library). Thus far, the new parallel *STHAMAS 3D* was tested only utilizing homogeneous computing environments, in particular, a clusters of workstations and a parallel computer.

For completeness it should be noted that a different parallel version of a crystal growth simulation has been reported in [7]. It utilizes a parallel version of the *SSOR* preconditioner and the *BiCGSTAB* iterative solver.

3 Load balancing strategy

In the *Largest Task First with Minimum Finish Time and Available Communication Costs* (*LTF_MFT_ACC*) algorithm from the *EVAH* package [3] a task is associated with a block in the CFD grid system. The size of a task is defined as the computation time for the corresponding block. According the *Largest Task First* (*LTF*) policy, the algorithms first sorts the tasks in a descending order by their size. Then it systematically allocates tasks to processors respecting the rule of *Minimum Finish Time* (*LTF_MFT*). The overhead involved in this process, due to data exchanges between neighboring blocks, is also taken into account. The *LTF_MFT_ACC* utilizes approximations of communication costs, which are estimated on the basis of the inter-block data volume exchange and the inter-processor communication rate.

In the *LTF_MFT_ACC* algorithm described in [3] for the homogeneous case, the estimated computational time for block i, t_i does not vary with the processor power. To take into account the variation of the computational power of the heterogeneous resources, we have modified the *LTF_MFT_ACC* (Figure 2) as follows. When the load balancing procedure is activated before a specific inner iteration of the simulation, several counters are started and they stop only at the end of the inner iteration. Those counters are measuring:

- the computer power, conceptualized as the time of performing a single cycle involving floating point operations; this information is further used to rank the resources;
- the computation time spent working on each block at an inner iteration; this information and the relative computer power are used to assume what will be the time spent working on a given block by other processor(s);
- the required volume of data to be exchanged between neighboring blocks (number of elementary data items);
- samples of communication times between each pair of processors collected for several volumes data (further time values are estimated by a linear interpolation).

The *DLB* tool [8] uses as inputs for the load balance strategy also the timing of computation for parallel blocks and the size of the interface of each block with its neighbors. Those times are not referring to a specific inner iteration of the CFD simulation, but to an average load history provided by computing systems that are part of the grid that is used to solve the problem.

4 Tests

The initial *LTF_MFT_ACC* algorithm was applied in [3] to a selected CFD problem, a Navier-Stokes simulation of vortex dynamics in the complex wake of a

Input:
Current distribution of the N blocks on P processors: $p(i) \in \{1, \ldots, P\},\ i = 1, \ldots, N$
Output:
New distribution of the N blocks on P processors: $p'(i) \in \{1, \ldots, P\},\ i = 1, \ldots, N$

Preliminaries, using the current distribution:
Record the computation time for each block: $T_i,\ i = 1, \ldots, N$
Record the quantity of data to be send/receive between blocks: $V_{j,k},\ j, k = 1, \ldots, N$
Estimate communication time between each pair of processors depending on the quantity of transmitted data: $Send(p, q, dim),\ Recv(p, q, dim),\ p, q = 1, \ldots, P,\ p \neq q$
Record the time spent to perform a standard test: $c_p,\ p = 1, \ldots, P$
Compute the relative speeds of computers: $w_p \leftarrow c_p / \min_{p=1,\ldots,P} c_p,\ p = 1, \ldots, P$
Normalize computation time for each block: $t_i \leftarrow T_i / w_{p(i)}, i = 1, \ldots, N$

To do:
Sort $t_i, i = 1, N$ in descending order
Set costs $C_p = 0, p = 1, \ldots, P$
For each $t_i, i = 1, \ldots, N$ do
  Find the processor q with minimum load: $?q,\ C_q = \min_{p=1,\ldots,P} C_p$
  Associate block i with processor q, $p'(i) \leftarrow q$
  Modify the costs: $C_q \leftarrow C_q + w_q t_i$
  For each processor $o \neq q$ having assigned a task j sending a message to task i,
    $C_o \leftarrow C_o + Send(o, q, V(j, i))$
    $C_q \leftarrow C_q + Recv(q, o, V(j, i))$
  For each processor $o \neq q$ having assigned a task j receiving a message from task i,
    $C_o \leftarrow C_o + Recv(o, q, V(i, j))$
    $C_q \leftarrow C_q + Send(q, o, V(i, j))$

Fig. 2. Modified *LTF_MFT_ACC* algorithm

region around hovering rotors. The overset grid system consisted of 857 blocks and approximately 69 million grid points. The experiments running on the 512-processor SGI Origin2000 distributed-shared-memory system showed that the *EVAH* algorithm performs better than other heuristic algorithms.

The CFD test case from the [8] used a three-dimensional grid for a heat transfer problem. The grid consisted of 27-block partitions with 40x40x40 grid points on each block (1.7 millions of grid points). For a range of relative speeds of computers from $1 \div 1.55$ a 21% improvement in the elapsed time was registered.

In our tests we have considered a three-dimensional grid applied to the crystal growth simulation using STHAMAS 3D with 38-block-partitions with a variable number of grid points: the largest one had 25x25x40 points, while the smallest one had 6x25x13 points (total of 0.3 millions of grid points in the simulation).

We considered two computing environments:

- a homogeneous one a Linux cluster of 8 dual PIV Xeon 2GHz Processors with 2Gb RAM and a Myrinet2000 interconnection (http://www.oscer.edu);
- a heterogeneous one – a Linux network of 16 PCs with variable computational power, from an Intel Celeron running at 0.6GHz, with 128Mb RAM to an

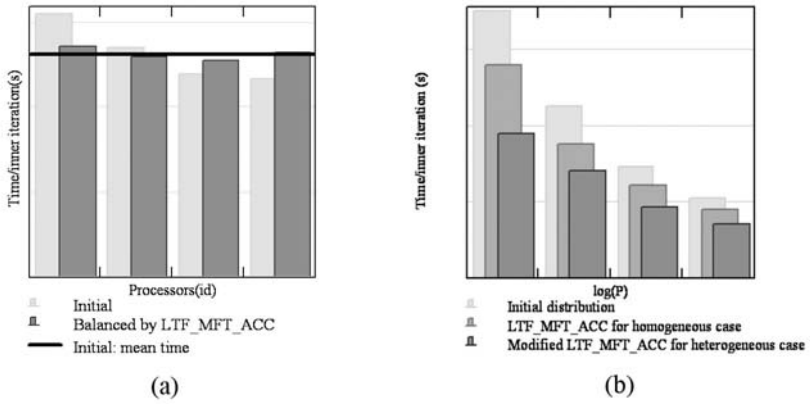

Fig. 3. Load balancing results : (a) in the case of 4 processors of the homogeneous environment, the *LTF_MFT_ACC* algorithm reduces the differences between the computation time spent by each processor in the inner iteration; (b) in the case of 2, 4, 8 and 16 processors of the heterogeneous environment, the *LTF_MFT_ACC* reduces the time per inner iteration, but a further significant reduction is possible using the modified *LTF_MFT_ACC* algorithm

Intel PIV running at 3GHz and with 1Gb RAM; these machines were connected through an Ethernet 10 Mbs interconnection (http://www.risc.uni-linz.ac.at).

Thus, the interval of the relative speeds of computers used in the second case reached $1 \div 2.9$.

Initially blocks of the discretization were distributed uniformly between the processors (e.g. in the case of two processors, first 19 blocks were send to the first processor, and the last 19 blocks were send to the second processor).

Due to the different number of grid points in individual blocks, the initial *LTF_MFT_ACC* algorithm running in the homogeneous environment recommended a new distribution of the nodes. Also the modified *LTF_MFT_ACC* algorithm made such a recommendation. For example, a reduction of 6% of the computation time required by an inner iteration was registered by applying both algorithms (the original one and the modified one) in the case of using 4 processors (Figure 3.a).

In the case of the heterogeneous environment, the *LTF_MFT_ACC* algorithm performs better: we observe a reduction ranging from 14% to 20% of the time spent by the CFD code in the inner iteration. A further reduction of the time was obtained when applying the modified *LTF_MFT_ACC* algorithm — varying from 20% to 31% (Figure 3.b). Comparing the time results with the ones for the initial distribution, a total reduction time obtained in our experiments varies from 33% to 45%.

5 Further improvements

The proposed load balancing technique shows to be useful in the considered case, a version of a CFD code running within heterogeneous or grid environments. Tests must be further performed to compare several dynamic load balancing techniques with the proposed one, not only in what concerns the influence of the computer power variations as in this paper, but also of the network speed variation. A particular grid testbed running MPICH-G2 applications will be used in the near future to perform such tests.

References

1. J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd, Agent-based grid load balancing using performance-driven task scheduling. In Procs. of IPDPS03, IEEE Computer Press (2003).
2. R. David, S. Genaud, A. Giersch, B. Schwarz, and E. Violard, Source code transformations strategies to load-balance grid applications. In Procs. GRID 2002, M. Parashar (ed.), *LNCS* **2536**, Springer (2002), 82–87.
3. M.J. Djomehri, R. Biswas, N. Lopez-Benitez, Load balancing strategies for multi-block overset grid applications, NAS-03-007, Available at www.nas.nasa.gov/News/ Techreports/2003/PDF/nas-03-007.pdf.
4. J.H.Ferziger, M.Perić, *Computational Methods for Fluid Dynamics*, Springer (1996).
5. H. Gao, A. Schmidt, A. Gupta, P. Luksch, Load balancing for spatial-grid based parallel numerical simulations on clusters of SMPs, In *Procs. Euro PDP03*, IEEE Computer Press (2003), 75–82.
6. O. Iliev, M. Scäfer, A numerical study of the efficiency of SIMPLE-type algorithms in computing incompressible flows on streched grids. In Procs. LSSC99, M. Griebel, S. Margenov, P. Yalamov (eds.), *Notes on Numerical Fluid Mechanics* **73**, Vieweg (2000), 207–214
7. D. Lukanin, V. Kalaev, A. Zhmakin, Parallel simulation of Czochralski crystal growth. In Procs. PPAM 2003, R. Wyrzykowski et al, *LNCS* **3019** (2004), 469–474
8. R.U. Payli, E. Yilmaz, A. Ecer, H.U. Akay, and S. Chien, DLB A dynamic load balancing tool for grid computing. In *Procs. Parallel CFD04*, G. Winter, A. Ecer, F.N. Satofuka, P. Fox (eds.), Elsevier (2005), 391–399
9. M. Perić, A finite volume method for the prediction of three-dimensional fluid flow in complex ducts, Ph.D. Thesis, University of London (1985).
10. D.Petcu, D.Vizman, J.Friedrich, M.Popescu, Crystal growth simulation on clusters. In *Procs. of HPC2003*, I. Banicescu (ed.), Simulation Councils Inc. San Diego (2003), 41–46
11. P.A. Sackinger, R.A. Brown, J.J. Brown, A finite element method for analysis of fluid flow, heat transfer and free interfaces in Czochralski crystal growth, *Internat.J. Numer. Methods in Fluids* **9** (1989), 453–492.
12. H. L. Stone, Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Num. Anal.* **5** (1968), 530–558.
13. D. Sun, A Multiblock and Multigrid Technique for Simulations of Material Processing, Ph.D. Thesis, State University of New York (2001).