Optimized Parallel Approach for 3D Modelling of Forest Fire Behaviour

Gilbert Accary[1], Oleg Bessonov[2], Dominique Fougère[3], Sofiane Meradji[3], and Dominique Morvan[4]

[1] Université Saint-Esprit de Kaslik, B.P. 446 Jounieh, Lebanon
[2] Institute for Problems in Mechanics of Russian Academy of Sciences, 101, Vernadsky ave., 119526 Moscow, Russia
[3] Laboratoire de Modélisation en Mécanique à Marseille, L3M–IMT, La Jetée, Technopôle de Château-Gombert, 13451 Marseille Cedex 20, France
[4] Université de la Méditerranée, UNIMECA, 60, rue Joliot Curie, 13453 Marseille Cedex 13, France

gilbertaccary@usek.edu.lb, bess@ipmnet.ru, fougere@l3m.univ-mrs.fr, sofiane@l3m.univ-mrs.fr, dominique.morvan@univmed.fr

Abstract. In this paper we present methods for parallelization of 3D CFD forest fire modelling code on Non-uniform memory computers in frame of the OpenMP environment. Mathematical model is presented first. Then, some peculiarities of this class of computers are considered, along with properties and limitations of the OpenMP model. Techniques for efficient parallelization are discussed, considering different types of data processing algorithms. Finally, performance results for the parallelized algorithm are presented and analyzed (for up to 16 processors).

1 Introduction

This work is carried out within the context of the European integrated fire management project (Fire Paradox) aiming to obtain a full-physical three-dimensional model of forest fire behaviour. The proposed approach accounts for the main physical phenomena involved in a forest fire by solving the conservation equations of physics applied to a medium composed of solid phases (vegetation) and gas mixture (combustion gases and the ambient air). The model consists in coupling the main mechanisms of decomposition (drying, pyrolysis, combustion) and of transfer (convection, diffusion, radiation, turbulence, etc.) taking place during forest fire propagation [1]. This multiphase complete physical approach already exists in 2D approximation [2] and consists in solving the described model in a vertical plane defined by the direction of fire propagation. The 3D extension of the existing model will enable to render 3D effects observed in real fires and to represent the real heterogeneous structure of the vegetation. The CFD code under development is currently at the stage of predicting turbulent gas flows and has been validated on several benchmarks of natural, forced, and mixed convection [3].

The extended 3-dimensional formulation requires much more computational resources than the previous 2D model. The new model needs substantially bigger grids ($N_x \times N_y \times N_z$ vs. $N_x \times N_y$ grid points), more complicated discretizations (more terms in the equations), additional grid compression in problematic areas (because of non-flat fire interfaces), more robust and expensive algebraic solvers. As a result, the total computational complexity of the algorithm increases by two orders of magnitude or more.

In order to be able to perform precise computations in reasonable time, it is necessary to exploit efficiently all available resources and improve computational performance by combining the following considerations: efficient numerical method and procedure, robust algebraic solvers, optimization of the code for modern superscalar microprocessors with memory hierarchies, and parallelization of the algorithm for moderate number of processors. However, this last consideration remains the most efficient way for increasing the speed of computations.

The next important point is the choice of a parallel computer architecture and parallelization model for this work. Generally, distributed memory parallel computers (clusters) are used for large-scale computations. However, such parallel computers, used with the appropriate MPI message-passing model, result in very complex algorithms and require tight optimization of communication exchanges [4]. In addition, a model with relatively slow communication exchanges can't be efficiently used for many algorithms [5]. Finally, it is difficult to implement a portable code that would work on any parallel platform with required efficiency.

Thus, shared-memory computer architecture was chosen as a target for the new parallel code. An OpenMP parallelization model without explicit exchanges is used for the algorithm [6]. This model, which is the natural choice for shared-memory computers, is just an extension of high level languages (Fortran, C). With appropriate programming, the code may work on a parallel system with any number of processors. Consequently, the new code becomes portable and compatible with many parallel platforms.

However, implementation of the shared-memory paradigm encounters another difficulty: almost all modern shared-memory systems with moderate or high number of processors (4, 8, 16 and more) belong to the class of Non-uniform Memory Access (NuMA) computers. It means that every processor or group of processors (processor node) is directly connected only to its own (local) memory while an access to the non-local (remote) memory is performed through intermediate communication network. Due to such organization, remote accesses become much slower than local ones. This restriction requires a special approach for the organization of parallel algorithms in order to ensure that most or all accesses from every processor node occur within this node's local memory.

Thereby, in the presented paper we will describe the mathematical model and numerical method, strategy of OpenMP parallelization on NuMA computers, results of parallelization efficiency of the new 3D code, and summary with conclusions.

2 Mathematical Model and Numerical Method

We consider Newtonian fluid whose flow is governed by non-stationary Navier-Stokes equations in Boussinesq approximation. The model is also capable to handle the Low Mach number approximation in the context of perfect gas [3]. The set of equations consists of the continuity equation, the momentum equations in three spatial dimensions ($i = 1, 2, 3$) and the equations for energy and turbulent quantities. The generalized governing equation for all variables is expressed in the following conservative form:

$$\frac{\partial}{\partial t}(\rho\phi) + \frac{\partial}{\partial x_i}(\rho\phi u_i) = \frac{\partial}{\partial x_i}\left(\Gamma\left(\frac{\partial\phi}{\partial x_i}\right)\right) + S_\phi \quad \text{with} \quad \phi = 1, u_1, u_2, u_3, T, k, \epsilon$$

where ϕ represents the transported variable; ρ and u_i are respectively the local density and the i-th component of velocity; Γ – the effective diffusion coefficient; S_ϕ – the source term for the corresponding variable.

The Finite Volume discretization is applied to the non-uniform Cartesian staggered grid. Second-order discretizations are used, employing the quadratic upstream interpolation of advective terms with flux limiters.

The transport equations are solved by a fully implicit segregated method based on the SIMPLER algorithm [7]. The non-symmetric linear systems obtained from the discretized equations are solved by the BiCGStab iterative method, while the symmetric linear system of the pressure equation is solved by the Conjugate Gradient method (CG). The use of under-relaxation techniques, when necessary, allows better convergence and stability of the solution.

The code is applicable for simulation of flows in rectangular domains. Validation of the sequential version of the code has been performed for several common benchmarks (lid driven cavity, differentially heated cavity etc.).

3 OpenMP Parallelization on NuMA Computers

We will consider the strategy of OpenMP parallelization using the SGI Altix 350 shared memory system with non-uniform organization. It consists of 10 processor nodes, each with two Intel Itanium 2 processors (1.5 GHz, L3-cache 4 Mbyte) and 4 Gbyte of the local memory. Processor nodes are interconnected by the special NuMA-link interfaces through the high-speed switch that provides accesses to non-local (remote) memories. Logically, the considered system belongs to the shared-memory class, when every process may transparently access any memory location in a system. However, remote accesses are much slower than local ones. For example, the peak memory read rate (throughput) within a node is equal to 6.4 Gbyte/s, while the peak throughput of NuMA-links is two times less.

Direct measurements show that the speed of regular read accesses achieves 6.1 GByte/s for local memories, and only 2.4 GByte/s for remote locations. This speed is very important for many computational algorithms that perform processing of data in big 3-dimensional arrays. Performance of such memory-bound algorithms depends on the memory throughput almost linearly.

Therefore, it is necessary to ensure that all processes of a parallel program access only (or mostly) data located within a local memory. On the system level, it can be done by the special utility that affiliates (bounds) every process to its own processor. This binding is needed to avoid migration of processes between processors and to guarantee that every processor executes only one process. In a multi-user computer system, some discipline must be established in order to avoid interference of processes from different programs.

On the application level, it is important to organize an algorithm in such a way that every thread (branch) of a parallelized algorithm would process only (mostly) a corresponding piece of data. Additionally, these data must be distributed between processor node's memories by the appropriate way (in the beginning of the execution). If these requirement are not fulfilled, parallel performance may drop two times or more.

The same rules and restrictions apply to another types of NuMA computer systems. For example, systems built on AMD Opteron processors also use relatively slow interprocessor links. In these systems, processors are interconnected into a mesh that imposes an additional limitation: access to some particular memory location may pass through several intermediate (transit) processors if the target processor (who owns the required location) is not connected directly to the requesting one. Therefore, Opteron-based systems (with mesh topology) may become less flexible and less efficient for OpenMP parallelization, in comparison to switch-based systems (with star topology).

Generally, the OpenMP extension to a high level language (Fortran in our case) is very simple and complements this language by several comment-like directives. These directives instruct a compiler how to perform parallelization of a program. The most important and popular directive is "PARALLEL DO" which is usually applied to an outermost "`do`" statement (for nested loops) (see example on Fig. 1, left). In accordance with the number of processors requested, iterations of this loop are evenly distributed between branches (threads) of a program for execution in different processors. This corresponds to the geometric splitting of a processed data array (3-dimensional, as a rule) into sub-arrays by the last spatial dimension (Fig. 1, right).

```
!$OMP DO
      do K=1,Nz
        do J=1,Ny
          do I=1,Nx
      Wo3(I,J,K)=Wo2(I,J,K)+
     &      beta*Wo3(I,J,K)
          enddo
        enddo
      enddo
!$OMP END DO
```

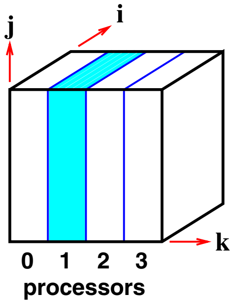

Fig. 1. Example of "PARALLEL DO" directive (left); geometric splitting of data array by this directive (right)

The OpenMP parallelization model is very convenient for ”true” shared-memory computers with uniform memory. For these computers, it is possible to split a multidimensional computational domain by any spatial direction. For non-uniform systems, only splitting by the last direction ensures that necessary portions of data are fully located within the corresponding processor node’s memory. In order to avoid remote memory accesses, algorithms must be rearranged. Some sorts of algorithms (for example, those with recursive dependences in all spatial directions) can’t be parallelized easily and efficiently within the OpenMP model. On the other hand, algorithms of the ”explicit” nature, that pass sequentially through data arrays and use small local data access patterns (stencils), may benefit from this model. Accesses to remote memory occur only within boundaries between subdomains in this case.

One-dimensional splitting of multidimensional arrays imposes another limitation on the OpenMP model for NuMA computers: subdomains become very ”narrow” by this dimension, and, as a result, accesses to remote memory through boundaries become frequent enough (compared to the number of local accesses). Also, the last dimension may become not divisible by the number of processors that results in a bad load balance. These limitations restrict the degree of efficient parallelization by moderate number of processors (typically 8–16).

Unfortunately, OpenMP in the current state has no special tools or directives for NuMA parallelizations. Therefore, only indirect techniques (as described in the current paper) may by applied to customize parallelization methods for this sort of computers.

4 Parallelization Approach and Results

In the current implementation, the considered CFD code has the ”explicit” nature, i.e. it doesn’t employ direct implicit solvers. Most part of its computational time (about 80 %) is consumed by two Conjugate Gradient type solver routines – CG (for pressure) and BiCGStab (for transport equations). These routines process data arrays with 7-point local stencils and therefore perform remote memory accesses only when processing data near subdomain boundaries. As a result, these CG-type routines can be efficiently parallelized using the OpenMP model for NuMA. Another time-consuming routines also belong to the ”explicit” class and can be parallelized without difficulties.

In order to ensure that data are correctly distributed within local memories of corresponding processor nodes, it is necessary to perform special initialization of all important data arrays. Neither the current OpenMP standard, nor the OpenMP-aware compiler used in this work (Intel Fortran 9.1) have any tools for explicit data distribution. To provide this distribution, a simple routine is used that initializes all arrays in nested loops with ”PARALLEL DO” directives. This routine is called in the beginning of the code when memory pages for arrays are not yet allocated. Since this allocation occurs ”by demand”, it is necessary to issue the first request to any element of data from the same processor node, which will be used for further processing of this element. Therefore, parallel loops for initialization of data must be organized similarly to data-processing "`do`" loops with exactly the same splitting of outermost iterations between processors (Fig. 1).

Validation of the parallelized code and measurements of its parallelization efficiency were performed on the benchmark problem of natural convection in differently heated cavity [8]. We used the Boussinesq flow configuration with Rayleigh number Ra $= 10^6$ and grid size $60 \times 60 \times 60$. Performance results are presented on Fig. 2. In the table, results of relative acceleration (compared to the previous grade with half number of processors), absolute acceleration (compared to one processor) and parallelization efficiency are shown.

	No. of processors				
	1	2	4	8	16
time (seconds)	1966	1448	523	246	173
relative speedup	–	1.36	2.77	2.13	1.42
total speedup	–	1.36	3.76	7.99	11.36
efficiency	–	68%	94%	100%	71%

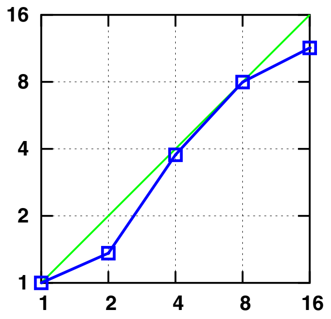

Fig. 2. Parallelization results for the benchmark problem

Relative acceleration for two processors is not high because both processors compete for the same memory, and performance is limited by its throughput. On the other hand, for 4 and 8 processors we see a superlinear speedup owing to the help of a large 4 MByte L3-cache in each processor. As a result, total acceleration for 4 and 8 processors corresponds to the linear profile. For 16 processors, some negative effects are accumulated: load disbalance (60 is not divisible by 16) and influence of big boundaries (1 boundary grid point per 2 or 3 internal points). Due to these effects, parallelization efficiency drops. It follows that the reasonable degree of efficient parallelization for this configuration is 8, at most 16, that corresponds to the goal of the current work.

The presented parallel code is based on a serial code that was initially optimized for modern pipelined processors with memory hierarchies. Further optimization of the code will be devoted to the acceleration of algebraic solvers by applying efficient preconditioners. It was demonstrated that the explicit-class (local) Jacobi preconditioner can be easily parallelized. However, for more efficient implicit (global) line-Jacobi preconditioner, new parallelization technique must be developed with parallel solution of tri-diagonal linear system. This parallelization will be based on the previous work [4]. Another direction of the development of the current CFD code will consist in incorporation of the radiation transfer algorithm. This algorithm can't be parallelized by geometric manner and will need a special approach based on the concept of input data parallelism.

5 Conclusion

In this work we developed the strategy of OpenMP parallelization for NuMA computers and parallelization method for 3D CFD code for modelling of forest fire behaviour, taking into account restrictions and limited flexibility of the current state of the OpenMP environment. This new method allows to achieve good parallelization efficiency for moderate number of processors (up to 16). The obtained results correspond to the general goal of the work – to obtain a tool for performing precise 3D computations in reasonable time.

Acknowledgements. This work was supported by the European integrated fire management project (Fire Paradox) under the Sixth Framework Programme (Work Package WP2.2 "3D-modelling of fire behaviour and effects"), and by the Russian Foundation for Basic Research (project RFBR-05-08-18110).

References

1. Morvan, D., Dupuy, J.L.: Modeling of fire spread through a forest fuel bed using a multiphase formulation. Combust. Flame 127, 1981–1994 (2001)
2. Morvan, D., Dupuy, J.L.: Modeling the propagation of a wildfire through a Mediterranean shrub using a multiphase formulation. Combust. Flame 138, 199–210 (2004)
3. Le Quéré, P., et al.: Modelling of natural convection flows with large temperature differences: A Benchmark problem for Low Mach number solvers. Part 1. Reference solutions. ESAIM: Math. Modelling and Num. Analysis 39(3), 609–616 (2005)
4. Bessonov, O., Brailovskaya, V., Polezhaev, V., Roux, B.: Parallelization of the solution of 3D Navier-Stokes equations for fluid flow in a cavity with moving covers. In: Malyshkin, V. (ed.) PaCT 95. LNCS, vol. 964, pp. 385–399. Springer, Heidelberg (1995)
5. Bessonov, O., Fougère, D., Roux, B.: Parallel simulation of 3D incompressible flows and performance comparison for several MPP and cluster platforms. In: Malyshkin, V. (ed.) PaCT 2001. LNCS, vol. 2127, pp. 401–409. Springer, Heidelberg (2001)
6. Dagum, L., Menon, R.: OpenMP: an industry-standard API for shared-memory programming. IEEE Computational Science and Engineering 5(1), 46–55 (1998)
7. Moukalled, F., Darwish, M.: A unified formulation of the segregated class of algorithms for fluid flow at all speed. Numer. Heat Transfer, Part B 37, 103–139 (2000)
8. Bessonov, O., Brailovskaya, V., Nikitin, S., Polezhaev, V.: Three-dimensional natural convection in a cubical enclosure: a bench mark numerical solution. In: de Vahl Davis, G., Leonardi, E (eds.) CHT'97: Advances in Computational Heat Transfer. Proc. of Symposium, Cesme, Turkey. Begell House, Inc., New York, pp. 157–165 (1998)