Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

Two-level parallelization of a fluid mechanics algorithm exploiting hardware heterogeneity

Immo Huismann [*], Jörg Stiller, Jochen Fröhlich

Lehrstuhl für Strömungsmechanik, Technische Universität Dresden, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 20 December 2014
Received in revised form 20 April 2015
Accepted 14 May 2015
Available online 22 May 2015

Keywords:
Parallelization
Heterogeneous computing
MPI
GPGPU
OpenMP
OpenACC

ABSTRACT

The prospect of wildly heterogeneous computer systems has led to a renewed discussion of programming approaches in high-performance computing, of which computational fluid dynamics is a major field. The challenge consists in harvesting the performance of all available hardware components while retaining good programmability. In particular the use of graphic cards is an important trend. This is addressed in the present paper by devising a hybrid programming model to create a heterogeneous data-parallel computation with a single source code. The concept is demonstrated for a one-dimensional spectral-element discretization of a fluid dynamics problem. To exploit the additional hardware available when coupling GPGPU-accelerated processes with excess CPU cores, a straight-forward load balancing model for such heterogeneous environments is developed. The paper presents a large number of run time measurements and demonstrates that the achieved performance gains are close to optimal. This provides valuable information for the implementation of fluid dynamics codes on modern heterogeneous hardware.

1. Introduction

The use of high-performance computing (HPC) is a widely employed approach in science and engineering, of which computational fluid dynamics (CFD) is a major field. Current supercomputers incorporate a multitude of nodes, each comprising multiple CPUs, connected with each other to enable large-scale simulations. Yet, the structure of this type of hardware has grown increasingly more complex in recent years. The incorporation of multi-core components generated a new layer in the hierarchy of the system, while accelerator hardware is presently leading to an even more heterogeneous landscape. The two fastest supercomputers ranked on the TOP500 list [24], for example, incorporate accelerators, either Xeon Phi or general purpose GPUs (GPGPUs). As the hardware becomes more heterogeneous, the software needs to adapt in turn to attain optimal performance.

Traditionally, distributed-memory single-core systems were programmed using a message-passing approach, typically realized with the Message Passing Interface (MPI) [12]. The in-house finite volume code of one of the authors is a typical example [13]. With more cores per CPU, the message-passing approach leads to a higher amount of communication, since the computational domain is decomposed into more parts which need to exchange information. A common approach is hybrid parallelization utilizing OpenMP [5] inside a multi-core component and MPI to communicate between those, as discussed in [18]. Accelerated systems are programmed in a similar fashion, though with specialized programming languages that only exploit the accelerators, whereas the steering CPU processes maintain communication via MPI, e.g. [14].

Traditionally, data-parallel CFD codes are conceived either for CPUs or for GPGPUs, mainly due to different programming approaches being used. Nowadays some programming models and libraries allow to compile for different kinds of hardware, resulting in a code capable of running on clusters of either hardware type [4,25,10]. Yet, with the hardware environments growing more and more diverse, having to run programs simultaneously on different kinds of hardware often becomes inevitable [11].

The topic has therefore entered the agenda of researchers in computer science as well as researchers in the application field. In particular the latter strive to employ their intimate knowledge about physical and algorithmic properties to optimize computational performance, so as to run more and larger simulations at the same computational cost. On the other hand, this approach has its limits as it impacts on the versatility, so that more general software concepts are required. Several different approaches exist and it is difficult – as experienced by the authors – to assess the potential of each in order to decide about future algorithmic developments.

* Corresponding author.
E-mail addresses: Immo.Huismann@tu-dresden.de (I. Huismann), Joerg.Stiller@tu-dresden.de (J. Stiller), Jochen.Froehlich@tu-dresden.de (J. Fröhlich).

http://dx.doi.org/10.1016/j.compfluid.2015.05.012
0045-7930/© 2015 Elsevier Ltd. All rights reserved.

Homogeneous CPU clusters are nowadays well understood, as are heterogeneous CPU clusters [29]. While GPGPU-computing added a new layer of complexity and complications to HPC, even complex algorithms are now running on clusters of such hardware, sometimes enabling tremendous speedups over the traditional CPU clusters while using similar programming techniques [22,21]. But fully heterogeneous systems require the orchestration of multiple compute units and the best way to optimal performance is yet hidden.

Some task-based frameworks allow programming such systems [3] and researchers from the application field already combined OpenMP and CUDA to establish a collaboration of CPU and GPGPU, either in small-scale tests [9], or in large-scale simulations [30]. In the latter reference a speedup of about two is gained by utilizing the CPU in addition to the GPGPU while maintaining computational efficiency at 89%. Yet the reported implementation costs are substantial, as the computational parts of the code are effectively doubled, with one implementation for the GPGPU, one for the CPU and an additional steering logic. In [30] load balancing was established by fixing the ratio of CPU and GPGPU computation for all problem sizes. An iterative load balancing scheme, as employed in [9], can also be advantageous. In both studies the performance gain which can be achieved remains unclear and the implementation costs for the GPGPU enhancements are felt as a drawback.

The present paper seeks to provide improvements for load-balancing as well as for ease of implementation. While [30,9] utilized OpenMP in combination with CUDA to exploit the compute capabilities on one node, we aim to create a single-source implementation by combining MPI with the pragma-based language extensions OpenMP and OpenACC, compiling for each kind of hardware at a time and using a message-passing layer rather than a shared-memory layer. This approach is illustrated using a minimal solver featuring the essential components of a CFD code. The parabolic system being solved contains a nonlinear right-hand side and can be seen as a representative of a typical CFD subproblem obtained when solving a transport equation with the convection term discretized explicitly in time. Often, a linear elliptic system has to be solved, e.g., when discretizing friction terms implicitly or solving a Poisson equation for the pressure, both used in [20], for example. Since the explicit time stepping shares some similarity with an iterative scheme, load balancing information on such an explicit scheme can provide hints in this direction as well, thus avoiding further algorithmic parameters. The basic discretization scheme is a spectral-element method (SEM), which bears substantial perspectives for usage with CFD as already demonstrated in [26]. Most important for heterogeneous hardware is that this approach contains parameters to change granularity by switching from low to high approximation order [6].

The hardware considered in this paper is one socket comprising one CPU and one GPGPU, a typical building block of a supercomputer. In the present case study the goal is to maximize the performance of this building-block and thus exploit the heterogeneity of this system. First, the performance in the homogeneous cases of CPU and GPGPU is studied in order to gain reference data for load balancing in the heterogeneous case. In the main part of the paper, the performance of the heterogeneous case and the quality of the load balancing are addressed. These data, as a case study, provide quantitative information relevant for devising parallelization strategies for larger application problems in fluid mechanics.

2. Model problem

The test case considered represents the combustion of a premixed gas in one dimension [27]. For suitably chosen parameters and initial conditions the problem is governed by the reaction–diffusion equation

$$\partial_t T = \partial_{xx} T + q(T) \tag{1}$$

$$q(T) = \frac{\beta^2}{2}(1-T)\exp\left(\frac{\beta(T-1)}{1+\alpha(T-1)}\right), \tag{2}$$

where $T(t,x)$ is the temperature, t the time and x the location, while β is the Zeldovich number determining the stiffness of the problem, and α the activation energy of the reaction. In this formulation the temperature is normalized to the non-dimensional interval $[0,1]$, where zero implies the unburned and one the fully burned state. Further information on physical issues and a derivation of the equation can be obtained in [28].

Eq. (1) is solved on the domain $\Omega = (0, x_{\text{end}}]$ in the time interval $(0, t_{\text{end}}]$ with the Dirichlet boundary condition $T(t,0) = 1$ at the left boundary and the Neumann boundary condition $\partial_x T(t, x_{\text{end}}) = 0$ at the right boundary. The initial condition is a stable reaction front at $x = x_f$ for $\beta \to \infty$, for which an analytic solution exists:

$$T(0,x) = \begin{cases} 1 & \text{for } x < x_f \\ \exp(x_f - x) & \text{for } x \geqslant x_f. \end{cases} \tag{3}$$

These conditions result in a reaction front traveling from left to right, as depicted in Fig. 1, with a speed very close to unity due to the normalization employed [27].

This test case can be readily extended towards more dimensions, multiple species, and can as well be coupled with a flow field. As discussed in the previous sections, it represents the most important elements of a typical CFD algorithms.

3. Discretization method

Eq. (1) is discretized in time with an explicit backward-differencing scheme of second order [17], while for the spatial discretization a spectral element method (SEM) is employed which is described in [8]. With the spectral-element formulation the domain is decomposed into n_e elements, as depicted in Fig. 2. On each of these elements the temperature is approximated using Lagrange polynomials of order p, so that $p+1$ degrees of freedom are present in each element. For this set of basis functions, the coefficients of a variable, e.g. the temperature T, in one element Ω_e at time t^n are denoted as

$$\underline{T}_e^n = \left(T_{0,e}^n \quad T_{1,e}^n \quad \cdots \quad T_{p,e}^n\right)^{\mathrm{T}}, \tag{4}$$

where the nomenclature shown in Fig. 2 is used. The matrix comprising all coefficients will be referred to as

$$\underline{\underline{T}}^n = \begin{pmatrix} T_{0,1}^n & \cdots & T_{0,n_e}^n \\ T_{1,1}^n & \cdots & T_{1,n_e}^n \\ \vdots & \ddots & \vdots \\ T_{p,1}^n & \cdots & T_{p,n_e}^n \end{pmatrix} = \left(\underline{T}_0^n \ldots \underline{T}_e^n \ldots \underline{T}_{n_e}^n\right). \tag{5}$$

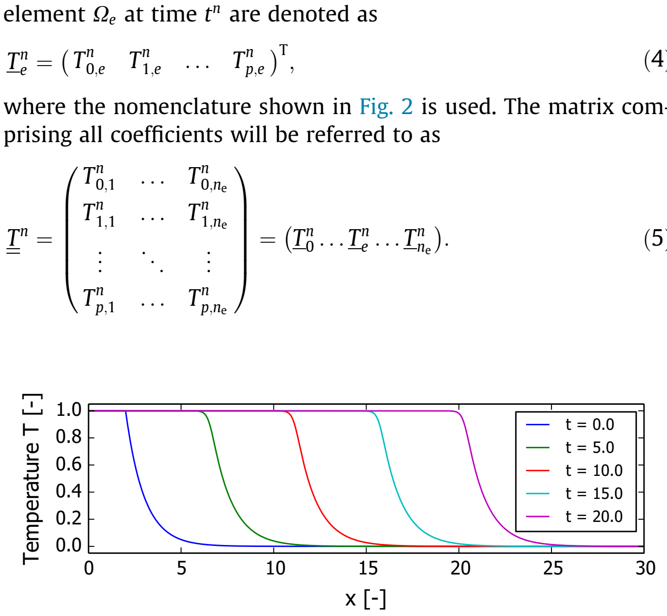

Fig. 1. Temperature distribution at several instants in time of a simulation of a reaction front according to (1)–(3) initialized with parameters $x_f = 2, \alpha = 0.8$ and $\beta = 10$.