N. D. Gonçalves[*], *S. P. Pereira, L. L. Ferrás, J. M. Nóbrega, O. S. Carneiro*

IPC/i3N – Institute for Polymers and Composites, University of Minho, Guimarães, Portugal

Using the GPU to Design Complex Profile Extrusion Dies

In the present work the benefits of using graphics processing units (GPU) to aid the design of complex geometry profile extrusion dies, are studied. For that purpose, a 3D finite volume based code that employs unstructured meshes to solve and couple the continuity, momentum and energy conservation equations governing the fluid flow, together with a constitutive equation, was used. To evaluate the possibility of reducing the calculation time spent on the numerical calculations, the numerical code was parallelized in the GPU, using a simple programing approach without complex memory manipulations. For verification purposes, simulations were performed for three benchmark problems: Poiseuille flow, lid-driven cavity flow and flow around a cylinder. Subsequently, the code was used on the design of two real life extrusion dies for the production of a medical catheter and a wood plastic composite decking profile. To evaluate the benefits, the results obtained with the GPU parallelized code were compared, in terms of speedup, with a serial implementation of the same code, that traditionally runs on the central processing unit (CPU). The results obtained show that, even with the simple parallelization approach employed, it was possible to obtain a significant reduction of the computation times.

1 Introduction

The need for multitask optimization has been, for long, a priority during our evolution. The idea that two persons usually perform better than one, is intrinsic to our existence, and this reflects in our actions, creations and way of life. With the rapid growth of science, sophisticated machines able to perform Men's work were invented, and this influenced, in part, our concept of parallel work, extending the parallel multitasking concept to the world of computers.

The desire to get more computing power and better reliability by orchestrating a number of low cost computers has given rise to the creation of computational clusters, with this invention being (arguably) attributed to Gene Amdahl of IBM (Amdahl, 1967), who in 1967 published a seminal paper on parallel processing (Amdahl's Law). Until now, the cluster concept continues to hold, but the evolution led to the creation of more powerful computers with more than one core, allowing the employment of the parallel computing concept in a single computer (Reilly, 2003).

When thinking about the optimization of engineering/physics problems, we realize that most of them result in the numerical solution of differential or partial differential equations. This numerical solution is usually expensive because it requires the resolution of large systems of equations. The fact that the physical models are usually nonlinear, forces an iterative procedure, enhancing the need for more computational power.

The current industrial problems are becoming more and more complex on a daily basis, consuming computer resources and demanding heavy computations. If we want to obtain acceptable computational times, we must take advantage from all the parallel computational power available in a computer. In this framework, graphics processing units (GPUs), for a long time only seen as powerful tools to enhance the video games graphics, are now a speedup enhancer for large dimension engineering problems (Elsen et al., 2008).

Therefore, since 1999 we have witnessed an increasing interest on GPUs, and the graphics processors have evolved from fixed function pipelines towards fully programmable floating point pipelines (Owens et al., 2008). NVIDIA (2013) has developed the CUDA programming toolkit, which includes an extension of the C language and facilitates the programming of GPUs for general purpose applications, by preventing the programmer to deal with the graphic details of the GPU (Castro et al., 2011).

The literature is rich in methods for the GPU parallel computation of a matrix solution. Bolz et al. (2003) implemented two basic computational kernels: a sparse matrix conjugate gradient solver and a regular-grid multigrid solver, showing that real-time applications, ranging from mesh smoothing and parametrization to fluid and solid mechanics solvers, could greatly benefit from these. Later, Krüger and Westermann (2003) introduced a framework for the implementation of linear algebra operators on programmable graphics processors (GPUs). They proposed a stream model for arithmetic operations on vectors and matrices for the efficient communication on modern GPUs. In order to assess their model, they performed simulations of the 2D wave equation and the incompressible Navier-Stokes equations, using direct solvers for sparse matrices. These two articles are perhaps the most cited on the GPU spe-

[*] Mail address: Nelson D. Gonçalves, IPC/i3N – Institute for Polymers and Composites, University of Minho, Campus de Azurém, 4800-058 Guimarães, Portugal
E-mail: nelsondfg@hotmail.com
nelson.goncalves@dep.uminho.pt

cific literature. However, there are other works that deserve our attention. Fatahelian et al. (2004) performed an in-depth analysis of dense matrix-matrix multiplication, which reuses each element of input matrices *O*(n) times. Although its regular data access pattern and highly parallel computational requirements suggested an efficient evaluation on GPU of matrix-matrix multiplications, they found that these are less efficient than current cache-aware CPU approaches. Hall et al. (2003) studied more efficient algorithms that make the implementation of large matrix multiplication on upcoming GPU architectures more competitive, using only 25 % of the memory bandwidth and instructions of previous GPU algorithms. Ohshima et al. (2007) proposed a new parallel processing environment for matrix multiplications by using both CPUs and GPUs. They decreased 40.1 % the execution time of matrix multiplications when compared with using the fastest of either CPU only case or GPU only case. Monakov et al. (2010) presented a new storage format for sparse matrices that better employs locality, has low memory footprint and enables automatic specialization for various matrices and future devices via parameter tuning.

A quick literature survey shows that the use of GPUs for increasing the performance of computations depends on the class of problems we study. In this work we are interested in the numerical solution of the Navier-Stokes equations. Although some limitations exist in regard to Computational Fluid Dynamics (CFD), we can find in the literature successful works regarding the solution of Euler and Navier-Stokes equations. In CFD problems several unknowns, e. g. pressure and velocity, distributions are calculated by solving systems of equations. The number of unknowns can be very large (several millions) which demands a lot of memory and computational time. This is even more demanding when dealing with optimization problems, where several simulation trials have to be done. Thus, any contribution to speedup the calculation, as the one obtained by code parallelization, may have a huge impact in areas that make use of CFD.

The first generation of GPU hardware allowed high speedups, but only single precision was used (Elsen et al., 2008; Hagen et al., 2006; Brandvik and Pullan, 2008). For the second generation of GPU, initially lower speedups were reported in the literature because of the employment of double precision numbers (Cohen and Molemaker, 2009; Corrigan et al., 2009), but Kampolis et al. (2010) and Asouti et al. (2010) reported double precision speedups for 2D and 3D Navier-Stokes solvers of circa 20 ×. A very recent paper on the GPU performance for a finite-difference compressible Navier-Stokes solver, suitable for direct numerical simulation (DNS) of turbulent flows, also revealed speedups of 22 ×. However, in order to obtain such performance, all the above mentioned implementations required a complex and efficient manipulation of the several memories available on the GPU. Aiming to evaluate the performance of the Fermi GPU generation, Pereira et al. (2013) obtained maximum speedups of 20 ×, solving the 2D Navier-Stokes equations together with an inelastic constitutive equation for simple benchmark flows. The results showed that it was possible to obtain a significant better performance, without complex memory manipulations, which are only accessible to programmers specialized in GPU.

The objective of this work is to assess the performance of a GPU parallelized 3D Navier-Stokes solver, using inelastic fluids governed by the Bird-Carreau constitutive equation, and its employment on the design of industrially relevant extrusion dies for the production of complex profiles, such as a catheter, for medical applications, and a wood plastic composite profile (WPC), with application on the building industry. In this way, we extend to 3D the previous work by Pereira et al. (2013), where the GPU parallelization was developed just for 2D case studies, but still using the same straightforward implementation (without any complex memory manipulations).

The remaining of this paper is organized as follows. In section 2 we describe the relevant governing equations, the numerical procedure and the code parallelization. In section 3 we present the code verification, discuss the performance analysis for three benchmark problems and, finally, evaluate the benefits of employing the GPU parallelized code on the design of two extrusion dies for the production of complex profiles. The paper ends with the conclusions in section 4.

2 Governing Equations and Numerical Procedure

2.1 Code Implementation

In this work we consider the isothermal incompressible fluid flow that is governed by the continuity,

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{1}$$

and momentum,

$$\rho\left(\frac{\partial u_i}{\partial t} + \frac{\partial u_j u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\eta(\dot\gamma)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right), \tag{2}$$

conservation equations, where ρ is the fluid density, t is the time, u_i is the i^th velocity component, p is the pressure and $\eta(\dot\gamma)$ is a non-constant viscosity modeled using the Bird-Carreau constitutive equation, given by

$$\eta(\dot\gamma) = \eta_\infty + \frac{\eta_0 - \eta_\infty}{\left(1 + (\lambda\dot\gamma)^2\right)^{\frac{1-n}{2}}}, \tag{3}$$

where $\dot\gamma$ is a function of the second invariant of the rate of deformation center $\left(\dot\gamma = \sqrt{2 \mathrm{tr} D^2},\ \ D = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right)$, η_0 is the zero shear-rate viscosity, η_∞ is the viscosity at very high shear rates, λ is a characteristic time, tr stands for the trace of the matrix and n is the power-law index.

For the wall velocity boundary condition, we assumed the usual no-slip condition,

$$u_i = 0. \tag{4}$$

A fully-implicit Finite Volume based numerical method is used to solve Eqs. 1 to 3. The method employs a time marching pressure-correction algorithm, formulated with a collocated variable arrangement and unstructured meshes (Gonçalves et al., 2013). The governing equations are integrated, in space, over the control volumes (cells with volume V_P) forming the computational mesh, and along time, over a time step (Δ_t). The volume integration benefits from the Gauss divergence theorem, and the subsequent surface integrals are discretized, with the help of the midpoint rule, so that sets of linearized algebraic equations are obtained, for each velocity component u_i, having the following general form:

$$a_P u_P = \sum_{nb} a_{nb} u_{nb} + S_u. \tag{5}$$

In these equations a_P and a_{nb} are the coefficients accounting for advection and diffusion contributions, S_u is a source term encompassing all contributions not included in the before mentioned coefficients, the subscript P denotes the cell under consideration and subscript nb its corresponding neighbor cells.

The set of algebraic equations (Eq. 5) are sequentially solved for the Cartesian velocity components by an iterative solver. The newly computed velocity field usually does not satisfy the continuity equation (i. e. Eq. 1), thus it needs to be corrected by an adjustment of the pressure gradients that drive it. This is accomplished by means of a pressure-correction field obtained from a discrete Poisson equation, derived from a discretized form of the continuity equation in combination with the momentum equation. This pressure correction equation is solved by a Jacobi iterative solver. The correction of the velocity field follows the SIMPLE strategy proposed by Patankar (1980). On the SIMPLE iterative procedure, the viscosity is updated at the end of each iteration step using the model given by Eq. 3.

For more details on the numerical implementation see Gonçalves et al. (2013).

2.2 CPU and GPU Implementations

With the second generation of GPUs, the Fermi architecture was introduced, and GPUs became more suitable for scientific computations. The GPUs comprises different types of memories, from fast to slow, and with this new architecture memory access is automatic, without the need of programmer's intervention, thus facilitating the implementation of random access memory algorithms.

Figure 1 shows the sequence of tasks for the GPU code implementation, where the white boxes indicate the routines executed on CPU and the gray boxes contain the routines executed on GPU. The full CPU implementation is similar to that shown in Fig. 1, without the "coloring scheme" that is not required on serial implementations.

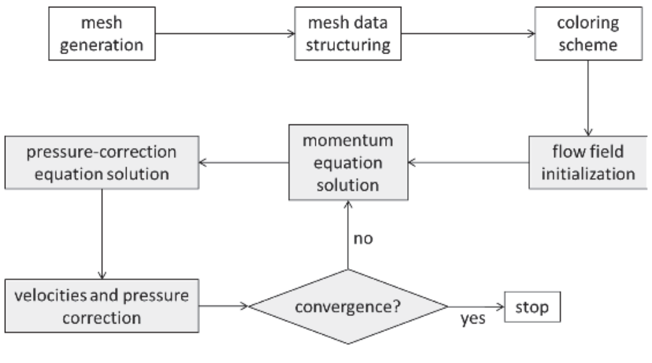

mesh generation
mesh data structuring
coloring scheme
pressure-correction equation solution
momentum equation solution
flow field initialization
no
velocities and pressure correction
convergence?
yes
stop

Fig. 1. Scheme used for coupling velocity and pressure fields. The gray boxes represent the routines ported to the GPU

To port the numerical code to the GPU, we used the most basic procedure, running on the GPU the most time consuming routines, and minimizing the time required for the data transfer between the CPU and GPU, which is done only at the start and end of the SIMPLE algorithm. Therefore, the mesh generation, data structuring and coloring scheme routines are executed in the CPU, while the heaviest part of the algorithm is ported to the GPU, as can be seen in Fig. 1. Note that the advantages obtained through the SIMPLE algorithm parallelization would be the same if alternative algorithms (SIMPLER or PISO (Patankar, 1980)) were employed.

It is important to notice that, as discussed in Pereira et al. (2013), all the SIMPLE iterative procedure (gray boxes in Fig. 1) must run exclusively on the GPU. Otherwise, the time required to exchange the information between the GPU and the CPU, at the end of each iteration, would surpass the advantages obtained by porting the code to the GPU.

In order to avoid information loss, that occurs when two or more threads (running in parallel) try to access simultaneously the same memory address, we adopted a coloring scheme (Kampolis et al., 2010) when assembling the systems of equations. Accordingly, we colored differently those control volume faces that contribute to the same term of the system of equations coefficient matrix diagonal. In this way we guarantee that implicit contributions to that matrix diagonal are not inserted in a concurrent manner.

For both systems of equations (that are solved at each iteration), we chose the point-iterative Jacobi method, which uses a matrix stored in the compressed sparse row format.

The GPU used in the simulations was the NVIDIA G-force GTX480, and the CPU was an Intel Core i7-950 Processor (3.06 GHz) with 8 GB RAM, making use of just one core.

3 Case Studies

The results obtained from the numerical solutions, namely Poiseuille flow, lid-driven cavity flow, flow around a cylinder, and flow in a catheter and a wood plastic composite profile dies, will now be presented, together with a speedup comparison between the serial and parallel code implementations. It is important to notice that for all the tested case studies the results and number of iterations obtained with serial and parallel versions of the code were equal.

3.1 Benchmark Problems and Code Verification

Initially, and in order to verify the code implementation, simple flows like the flow around a cylinder, flow in a simple channel and the lid-driven cavity flow were studied (see Fig. 2). Our results were compared with the analytical solutions (Poiseuille flow for Re = 80), and with benchmark solutions that exist in the literature for both the flow around a cylinder (Re = 5) (Bharti et al., 2006) and the lid-driven cavity flow (Re = 100) (Ghia and Ghia, 1982).

In the flow around a cylinder problem, the results were analyzed comparing the length of wake (or recirculation), L_W^*, a dimensionless length, corresponding to the ratio between the length of the recirculation formed on the back side of the cylinder (L_W^*) and the cylinder diameter (D), and the angle of separation, θ_S, the angle between the symmetry line and the flow separation from cylinder surface (see Fig. 2B). The differences, between our results and the ones given in (Bharti et al., 2006), obtained for these two parameters, were circa 3.2 %, for L_W^*, and 1.7 %, for θ_S, with a mesh comprising circa 8 times more cells along the cylinder surface than the one used by Bharti et al. (2006), who referred that those results were obtained with an accuracy of 1 to 2 %.

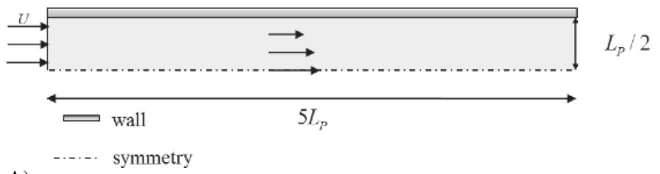
As shown in Fig. 3 for the Poiseuille and lid-driven cavity flows, accurate results were obtained, therefore validating our implementations in GPU and CPU. Note that for the lid-driven cavity case study the fluid was assumed to be Newtonian. To validate the code for non-Newtonian fluids, the Poiseuille flow was solved considering a Power-law constitutive equation, assuming the following parameters: $K = 1\,000$ Pa s^n and $n = 0.3$.

In Fig. 4 we present the speedups obtained for these three benchmark problems, as a function of the number of cells used in the numerical tests. As shown, a substantial increase of the code performance is obtained for the three cases. The maximum speedups achieved were $3.7\times$ for the Poiseuille flow, $7\times$ for the flow around a cylinder and $3.5\times$ for the lid-driven cavity flow. It can also be seen that the variation of the speedup with the number of cells leads to a sigmoidal shape, for all cases, evidencing that the performance scales with the mesh size.

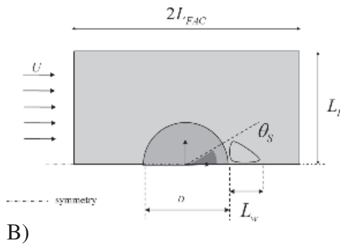
Note that for very coarse meshes, the GPU parallel implementation takes more time to perform the simulations than the single CPU. This happens because the time taken to exchange information between the CPU and the GPU is comparable to the time consumed to solve the problem. For the most refined meshes the time required for the exchange of information between the GPU and the CPU is residual.

3.2 Design of Profile Extrusion Dies

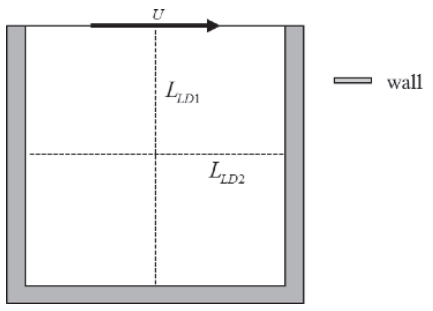
The previous results were obtained for quite simple geometries, unlike the complexity of current industrial problems. To design extrusion dies for the production of profiles comprising more complex geometries, extensive experience in the extrusion process is required, as well as the performance of several trials in order to achieve acceptable results. One of the main difficulties on extrusion die design is the achievement of a balanced flow at the flow channel outlet. To overcome these difficulties, numerical codes can be a valuable design aid, allowing to minimize the resources spent on the experimental trial-and-error process. With these tools the designer can improve the extrusion die channel geometry, by using either numerical based trial-and-error procedures (Szarvazy et al., 2000) or automatic algorithms that search for an optimized geometry guided by an objective function, without any user intervention (Nóbrega et al., 2000).



A)

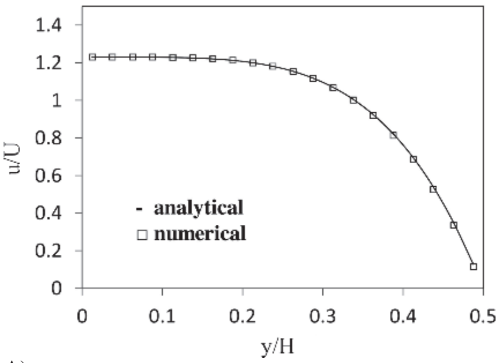


B)



C)

Fig. 2. Geometries of the case studies used for the code verification and speedup calculations: (A) Poiseuille flow, (B) flow around a cylinder, (C) lid-driven cavity flow



A)

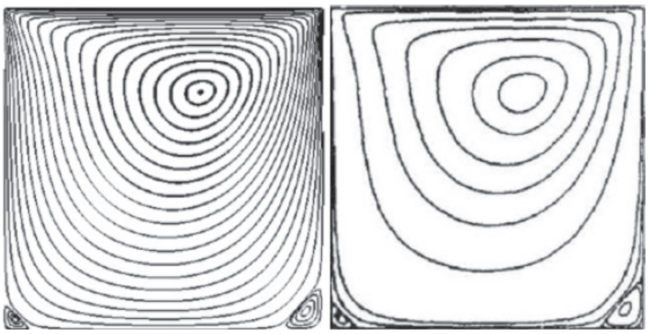

B)

Fig. 3. Results obtained for the case studies employed for the code verification: (A) velocity profile for the Poiseuille flow, (B) streamlines predicted for the Lid-driven cavity flow: left, obtained with the developed numerical code; right, presented by Ghia (Ghia and Ghia, 1982)

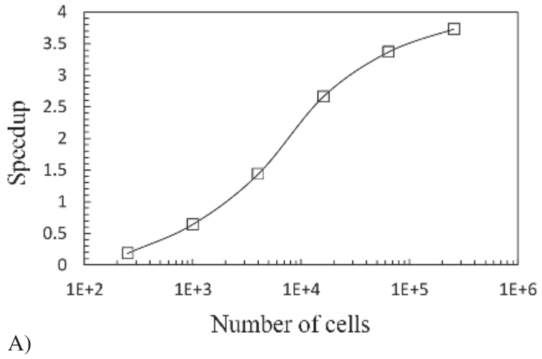
In order to evaluate the advantage of the employment of the parallelized version of the numerical code on the design of more complex geometries, two additional case studies will be considered, comprising the design of extrusion dies for the production of a medical catheter and a WPC decking profile.

3.2.1 Medical Catheter Extrusion Die

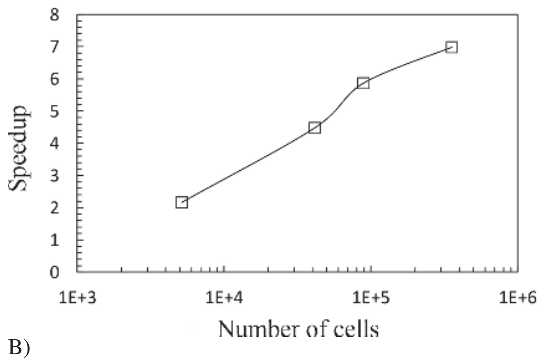
Medical catheters are devices that can be used to aid the treatment of diseases or the execution of surgical procedures, for instance by facilitating the insertion of drugs or surgical instruments in the patient's body. Catheters can be used on several applications, e. g., cardiovascular, urological, gastrointestinal, neurovascular and ophthalmic. Each application requires a specific catheter that can comprise several holes (lumens) that can possess different diameters. Due to its constant cross section, catheters are produced by extrusion, using medical grade materials.

As shown in Fig. 5A, the catheter geometry considered on this work has five channels (lumens). Due to symmetry reasons only half geometry was taken into account. It should be noted that the relative location of the channels are not expected to affect the catheter performance, since the lumens functionality is maintained when their diameter is assured (Fig. 8A). Thus, the problem to be solved for this profile is to identify the best location of the lumen that ensures the most balanced flow distribution. Accordingly, the geometry was parameterized with the location of the lumen centers (see Fig. 5B) (Gonçalves et al., 2013).

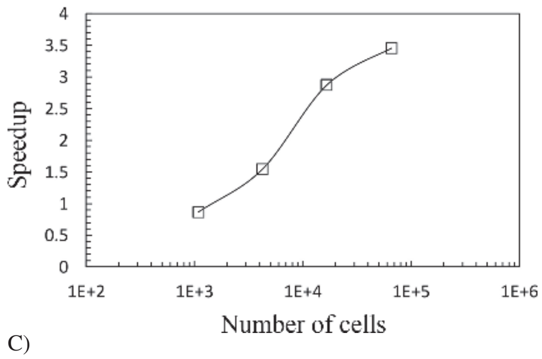
The material employed for the production of the catheter was a polypropylene homopolymer extrusion grade, Novolen PPH 2150, from Targor, which rheological behavior was experimentally characterized in capillary and rotational rheometers, at 230 °C (Nóbrega et al., 2003). The shear viscosity data were fitted to a Bird-Carreau constitutive equation, considering $\eta_\infty = 0$ Pa s, which yield the following parameters: $\eta_0 = 5.58 \times 10^4$ Pa s, $\lambda = 3.21$ s and $n = 0.3014$. For the



A)

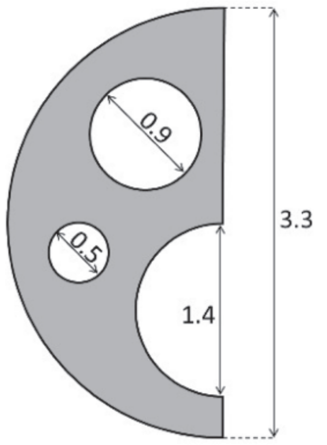


B)

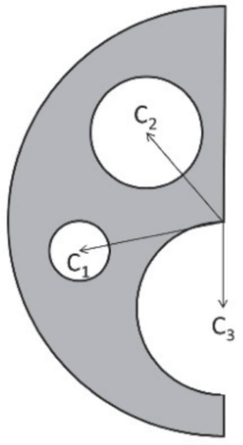


C)

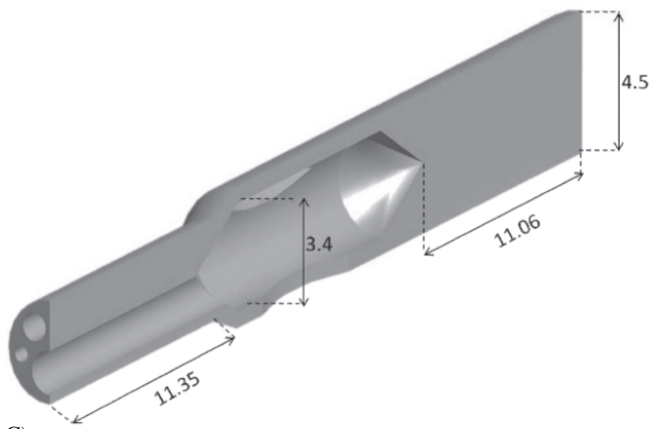
Fig. 4. Speedup obtained for the benchmark case studies: (A) Poiseuille flow, (B) flow around a cylinder, (C) lid-driven cavity flow



A)



B)



C)

Fig. 5. Extrusion die geometry for the production of a catheter profile: (A) cross-section dimensions (mm), (B) location of the channels and (C) flow channel dimensions (mm)

density, a typical value for polypropylene was considered ($\rho = 900$ kg m^{-3}).

To evaluate the dependency of speedup between the CPU and the GPU versions of the code, regarding to the number of cells, three meshes were used with 12382, 57085 and 389102 cells, being the last illustrated in Fig. 6.

For this specific problem a maximum speedup of 6.4 × was obtained (see Fig. 7), with the highest value corresponding to the most refined mesh.

The outflow distribution for the initial trial is illustrated in Fig. 8A; it can be seen that higher values of the velocity occur in regions where the restriction to the flow is lower. In order to balance the outlet flow, several different locations for the lumens were used, being the most balanced geometry obtained on the sixth trial, illustrated in Fig. 8B. More details on the optimization process are given on (Gonçalves et al., 2013).

The improvements obtained during the optimization process can be evaluated by the evolution of the objective function (Gonçalves et al., 2013) calculated for each trial (Fig. 9). Note that as the tool performance is improved, the objective function decreases. Both results, velocity field and objective function, evidence a significant improvement of the tool effectiveness.

Each run made on the CPU serial code took about 2 h 50 min of computation time, while for the parallelized GPU code the same problem took circa 27 min. Therefore, the full optimization process takes about 17 h 15 min and 2 h 40 min to run on the CPU and GPU, respectively, i.e., a significant reduction on the total computation time was obtained.

3.2.2 Wood-Plastic Composite Extrusion Die

The second complex geometry considered in this work is an extrusion die for the production of a wood-plastic composite (WPC) profile. WPCs are mainly made of wood particles dispersed in a thermoplastic matrix. Their main applications lie in the civil construction area, being an alternative to solid wood that requires more maintenance and has less flexibility in terms of geometry.

The dimensions of the cross-section of the initial trial die are shown in Fig. 10A. Due to symmetry reasons, only half geometry was considered. The achievement of a balanced flow at the outlet can be sought, keeping the outside contour shape, and modifying the dimensions of the torpedoes of the extrusion die that shape the hollow sections of the profile (Fig. 10B). These changes in the torpedoes can be done easily, since they are removable, and do not affect the profile functionality, which is mainly determined by its outer contour.

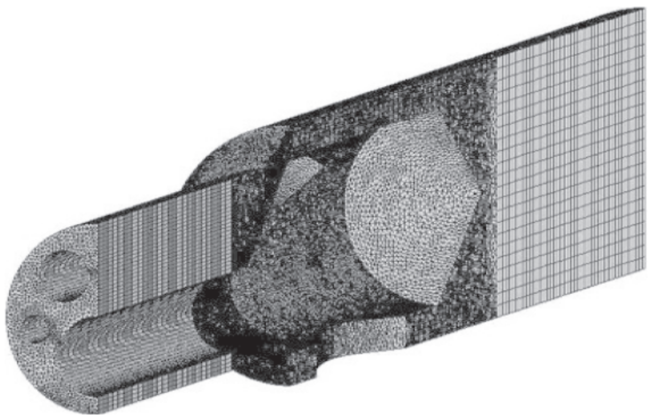


Fig. 6. Typical mesh used on the medical catheter case study, comprising 389 102 cells

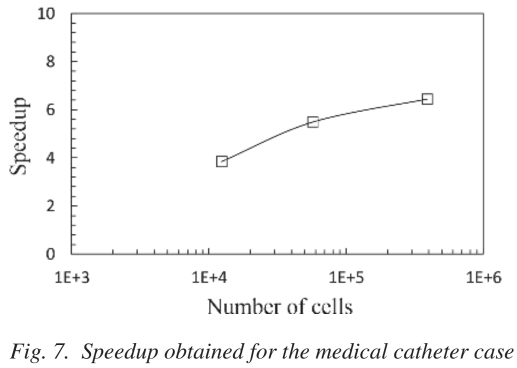


Fig. 7. Speedup obtained for the medical catheter case study

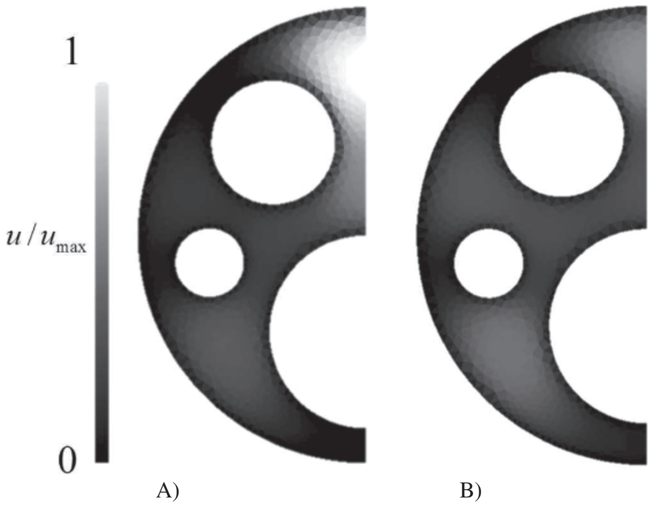


Fig. 8. Velocity field obtained for the outflow of a catheter profile: (A) initial trial, (B) final trial

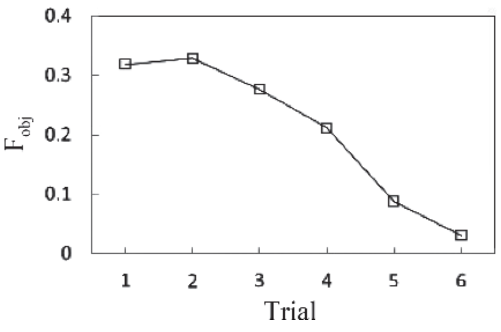


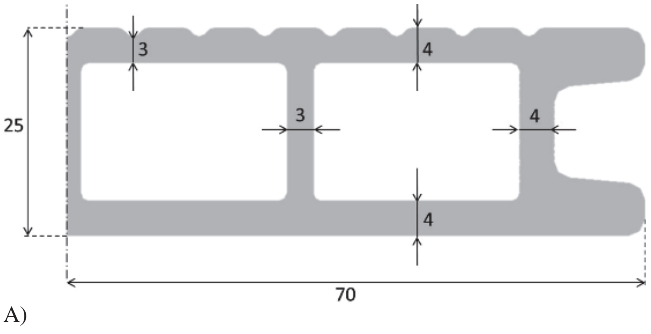
Fig. 9. Evolution of objective function along the optimization process for the medical catheter case study

To characterize the material, we used a capillary rheometer with dies of 2 mm diameter and L/D of 16 and 4 (L and D stand for the length and diameter of the die, respectively), to perform the Bagley correction. The tests were performed at a temperature of 190 °C. The experimental data (shear viscosity versus shear-rate) were fitted in order to obtain the Bird-Carreau model parameters that resulted in: n = 0.32, $\eta_0 = 53993$ Pa s, $\eta_\infty = 0$ Pa s and $\lambda = 2.36$ s. Other characteristics of the WPC are: density 1 200 kg m^{-3}, specific heat 1 300 J kg^{-1} K^{-1} and thermal conductivity of 0.08 W m^{-1} K^{-1}.

To analyze the evolution of the speedup obtained with the parallelized version of the code relatively to the serial one, as function of the number of cells, three different meshes with 57 170, 230 282 and 814 032 cells, were considered (Fig. 11, shows the most refined mesh employed).

A maximum speedup of 6.7 × was achieved (Fig. 12) for the finest mesh employed. However, between the two most refined meshes, there was only a residual increase on the speedup obtained. This asymptotic behavior is common to all the solved problems as well in other parallel processes. The speedup obtained in any code parallelization has a maximum achievable value. When coarse meshes are employed, that maximum value is not obtained because the additional operations required for parallelization, in what concerns to data manipulation, have a non-negligible weight on the total computation time. By refining the mesh, the relevance of those operations diminishes and the maximum speedup value is approached.

The flow distribution of the initial trial at the extrusion die outlet cross section is shown in Fig. 13A, where it can be seen that highest average velocities occur in sections where the flow restriction is lower.



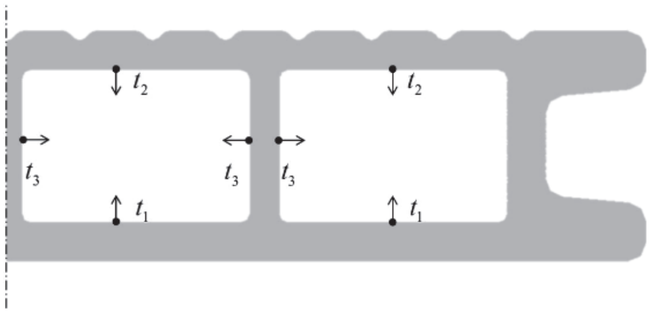


Fig. 10. Extrusion die geometry for the production of a WPC profile: (A) dimensions (mm);, (B) parameters employed for optimization purposes

As in the previous case, the flow balance of this die was also optimized, and a similar objective function was used to drive the process. In Fig. 14 it can be seen that the objective function value decreases significantly from the first to the last trial.

For this case the computation time needed to each run made on the CPU serial code was about 7 h 40 min, while for the par-

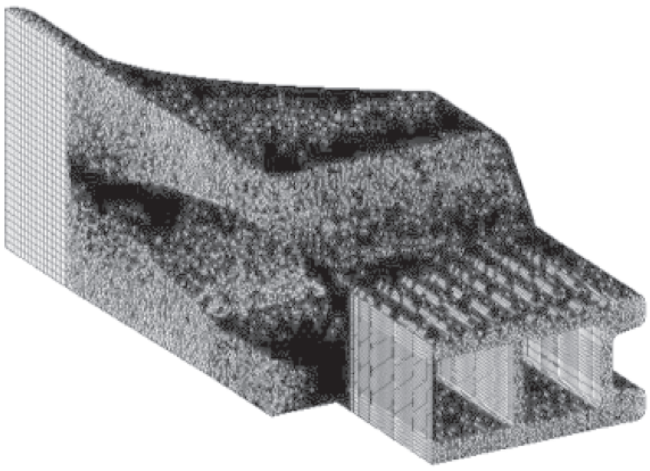


Fig. 11. Typical mesh used on the WPC case study, comprising 814 032 cells

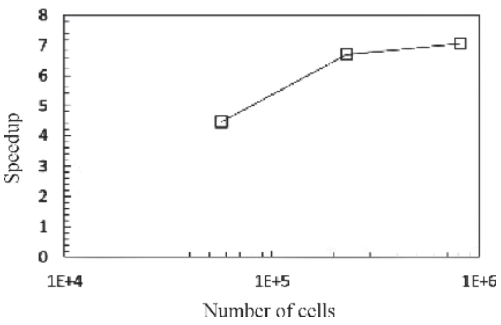


Fig. 12. Speedup obtained for the WPC profile extrusion die flow simulations

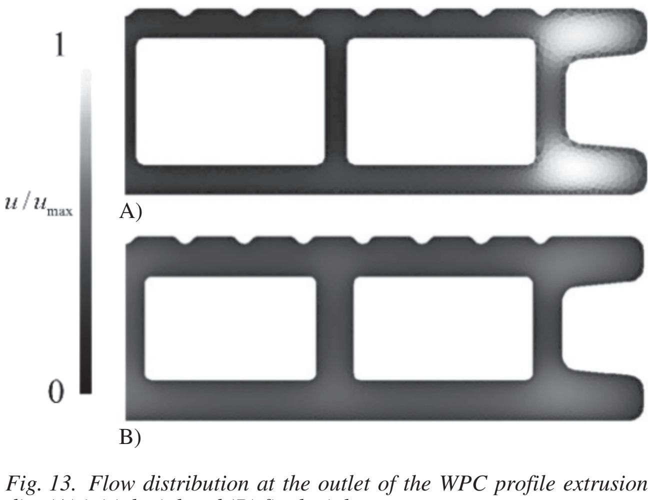


Fig. 13. Flow distribution at the outlet of the WPC profile extrusion die: (A) initial trial and (B) final trial

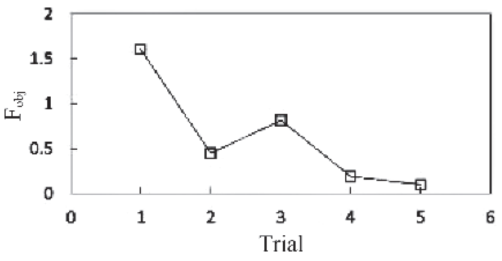


Fig. 14. Extrusion die for the production of the WPC profile: evolution of objective function along the optimization process

allelized GPU code the same problem took circa 1 h 15 min. Since the full optimization process needed five runs, it took about 38 h 30 min and 5 h 30 min to run on the CPU and GPU, respectively. As in the catheter case study, a significant reduction on the total computation time was achieved with the use of the parallelized GPU code.

4 Conclusions

This work presents the GPU parallel implementation of a 3D finite volume flow solver for unstructured meshes. The assessment of the code was carried out using three benchmark problems (Poiseuille flow, flow around a cylinder and lid-driven cavity flow), and its ability to deal with complex problems was illustrated with the design of extrusion dies for the production of a medical catheter and of a wood plastic composite profile. In order to evaluate the advantages of the GPU parallelization, speedups between the serial version of the numerical code, that runs on the CPU, and the GPU parallelized numerical code, were computed. For the tested case studies, speedups ranging from 3.5 × to 7 × were obtained.

In what concerns to the design of complex cross-section geometry profile extrusion dies, two case studies were considered. For the design of the medical catheter extrusion die, six numerical runs were required to attain an acceptable flow balance. The computation time required decreased from 17 h and 15 min to 2 h and 40 min on the serial (CPU) and parallelized (GPU) implementations of the numerical code, respectively. For the wood plastic composite decking profile case, the 5 runs required for the design process took 38 h and 30 min and 5 h and 30 min on CPU and GPU implementations of the numerical code, respectively.

From the results obtained it can be concluded that the GPU parallelization of the numerical code allowed a significant reduction of the time spent in calculation, which will have a noticeable positive impact on the design process. It is important to notice that these results were obtained without any complex memory management on the GPU and, therefore, there is room for additional future improvements.

References

Amdahl, G., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", Spring Joint Computer Conference, ACM, 483–485 (1967), DOI:10.1145/1465482.1465560

Asouti, V., Trompoukis, X., Kampolis, I. and Giannakoglou, K., "Unsteady CFD Computations Using Vertex-Centered Finite Volumes for Unstructured Grids on Graphics Processing Units", Int. J. Numer. Methods Fluids, **67**, 232–246 (2010), DOI:10.1002/fld.2352

Bharti, R. P., Chhabra, R. P. and Eswaran, V., "Steady Flow of Power Law Fluids across a Circular Cylinder", Can. J. Chem. Eng., **84**, 406–421 (2006), DOI:10.1002/cjce.5450840402

Bolz, J., Farmer, I., Grinspun, E. and Schröder, P., "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid", ACM Trans. Graphics, **22**, 917–924 (2003), DOI:10.1145/882262.882364

Brandvik, T., Pullan, G., "Acceleration of a 3D Euler Solver Using Commodity Graphics Hardware", I46th AIAA Aerospace Sciences Meeting, Citeseer (2008)

Castro, M., Ortega, S., De La Asuncion, M., Mantas, J. and Gallardo, J., "GPU Computing for Shallow Water Flow Simulation Based on Finite Volume Schemes", Comptes Rendus Mécanique, **339**, 165–184 (2011), DOI:10.1016/j.crme.2010.12.004

Cohen, J., Molemaker, M., "A Fast Double Precision CFD Code Using CUDA", Parallel Computational Fluid Dynamics: Recent Advances and Future Directions, 414–429 (2009)

Corrigan, A., Camelli, F., Löhner, R. and Wallin, J., "Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware", 19th AIAA Computational Fluid Dynamics Conference, 1–11, San Antonio, Texas, USA (2009)

Elsen, E., Legresley, P. and Darve, E., "Large Calculation of the Flow over a Hypersonic Vehicle Using a GPU", J. Comput. Phys., **227**, 10148–10161 (2008), DOI:10.1016/j.jcp.2008.08.023

Fatahalian, K., Sugerman, J. and Hanrahan, P., "Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication", ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 133–137 (2004), DOI:10.1145/1058129.1058148

Ghia, U., Ghia, K. N., Shin, C. T., "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method", J. Comput. Phys. **48**, 387–411 (1982), DOI:10.1016/0021-9991(82)90058-4

Gonçalves, N. D., Carneiro, O. S. and Nóbrega, J. M., "Design of Complex Profile Extrusion Dies through Numerical Modeling", J. Non-Newtonian Fluid Mech., **200**, 103–110 (2013)., DOI:10.1016/j.jnnfm.2013.02.007

Hagen, T., Lie, K. and Natvig, J., "Solving the Euler Equations on Graphics Processing Units", Computational Science–ICCS 2006, 220–227 (2006), DOI:10.1007/11758549_34

Hall, J., Carr, N. and Hart, J., "Cache and Bandwidth Aware Matrix Multiplication on the GPU", UIUC Technical Report UIUCDCS-R2003–2328 (2003)

Kampolis, I., Trompoukis, X., Asouti, V. and Giannakoglou, K., "CFD-Based Analysis and Two-Level Aerodynamic Optimization on Graphics Processing Units", Comput. Methods Appl. Mech. Eng., **199**, 712–722 (2010), DOI:10.1016/j.cma.2009.11.001

Krüger, J., Westermann, R., "Linear Algebra Operators for GPU Implementation of Numerical Algorithms", ACM Trans. Graphics, **22**, 908–916 (2003), DOI:10.1145/882262.882363

Monakov, A., Lokhmotov, A. and Avetisyan, A., "Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures", High Performance Embedded Architectures and Compilers, 111–125 (2010)

Nóbrega, J. M., Carneiro, O. S., Pinho, F. T. and Oliveira, P. J., "On the Automatic Die Design for Extrusion of Thermoplastic Profiles", 16th Annual Meeting of the Polymer Processing Society, Shanghai (2000)

Nóbrega, J. M., Carneiro, O. S., Pinho, F. T. and Oliveira, P. J., "Flow Balancing in Extrusion Dies for Thermoplastic Profiles – Part I: Automatic Design", Int. Polym. Proc., **18**, 298–306 (2003), DOI:10.3139/217.1745

NVIDIA, CUDA Home Page, Www.Nvidia.Com (2013)

Ohshima, S., Kise, K., Katagiri, T. and Yuba, T., "Parallel Processing of Matrix Multiplication in a CPU and GPU Heterogeneous Environment", High Performance Computing for Computational Science – VECPAR 2006, 305–318 (2007)

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J. and Phillips, J., "GPU Computing", Proc. IEEE, **96**, 879–899 (2008), DOI:10.1109/JPROC.2008.917757

Patankar, S. V.: Numerical Heat Transfer and Fluid Flow, CRC Press, Boca Raton, Florida, USA (1980)

Pereira, S. P., Vuik, K., Pinho, F. T., and Nóbrega, J. M., “On the Performance of a 2D Unstructured Computational Rheology Code on a GPU”, AIP Conference Proceedings 1526, Novel Trends in Rheology V., Zlin, Czech Republic (2013)

Reilly, E.: Milestones in Computer Science and Information Technology, Greenwood, Westport (2003)

Szarvasy, I., Sienz, J., Pittman, J. and Hinton, E., “Computer Aided Optimisation of Profile Extrusion Dies: Definition and Assessment of the Objective Function”, Int. Polym. Proc., **15**, 28–39 (2000), DOI:10.3139/217.1577

Acknowledgements

The authors gratefully acknowledge funding from Fundacão para a Ciência e Tecnologia (FCT) through the PhD Grant SFRH/BD43632/2008 and FCT (COMPETE Program) under the Projects EXPL/CTM-PRO/1299/2013 and PTDC/MAT/121185/2010, FEDER through the COMPETE 2020 Programme and National Funds through FCT under the project UID/CTM/50025/2013. Socimorcasal is also acknowledged for funding the work related to the WPC forming tool.

Date received: March 24, 2014
Date accepted: April 29, 2015

Bibliography
DOI 10.3139/217.2957
Intern. Polymer Processing
XXX (2015) 4; page 442–450
© Carl Hanser Verlag GmbH & Co. KG
ISSN 0930-777X