# Aerodynamic optimization of supersonic compressor cascade using differential evolution on GPU

Mohamed Hasanine Aissa, Tom Verstraete, and Cornelis Vuik

Citation: AIP Conference Proceedings **1738**, 480077 (2016); doi: 10.1063/1.4952313
View online: http://dx.doi.org/10.1063/1.4952313
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1738?ver=pdfcov
Published by the AIP Publishing

# Aerodynamic Optimization of Supersonic Compressor Cascade using Differential Evolution on GPU

Mohamed hasanine Aissa*, Tom Verstraete* and Cornelis Vuik†

*Von Karman Institute for Fluid Dynamics (VKI) 1640 Sint-Genesius-Rode, Belgium
†Delft University of Technology 2628 CD Delft, the Netherlands

**Abstract.** Differential Evolution (DE) is a powerful stochastic optimization method. Compared to gradient-based algorithms, DE is able to avoid local minima but requires at the same time more function evaluations. In turbomachinery applications, function evaluations are performed with time-consuming CFD simulation, which results in a long, non affordable, design cycle. Modern High Performance Computing systems, especially Graphic Processing Units (GPUs), are able to alleviate this inconvenience by accelerating the design evaluation itself. In this work we present a validated CFD Solver running on GPUs, able to accelerate the design evaluation and thus the entire design process. An achieved speedup of 20x to 30x enabled the DE algorithm to run on a high-end computer instead of a costly large cluster. The GPU-enhanced DE was used to optimize the aerodynamics of a supersonic compressor cascade, achieving an aerodynamic loss minimization of 20%.

**Keywords:** CFD RANS , Navier-Stokes, GPU, Turbomachinery application, Aerodynamics optimization
**PACS:** 47.11.-j

## INTRODUCTION

Differential evolution [1] is a robust heuristic optimization method of order zero, which means that no gradient information is required. The method builds first a random initial set of designs and evaluates them. Then a ranking drives the algorithm, based on a user-fixed fitness function, through the process of building the next generation. Crossover and mutation are the essential tools to conserve diversity and evolve the population while selection imposes environmental pressure allowing only the fit designs to survive to the next generation. Unlike gradient based methods, DE has built-in mechanisms allowing to avoid being trapped in local minima. Zero order methods have however a major drawback: they require a high number of evaluations to converge. Besides the fact that an entire population has to be evaluated at every generation, the improvement per generation can be very small, as a result of the essential mechanism to keep healthy diversity avoiding premature convergence, leading to a high number of iteration to achieve a certain improvement.

A key aspect of population based methods allows to alleviate this drawback: the designs inside one generation are fully independent and thus can be evaluated in parallel. The target is then to accelerate every design evaluation and run multiple evaluations simultaneously, which requires access to high performance computing (HPC) systems. The typical choice is between a standard cluster of CPUs and shared memory architectures. Standard clusters offer indeed a large number of processors, which potentially can lead to higher speedups, but the relation of speedup to number of cores, called scalability, is not taken for granted. It requires an intensive work on interprocessor communication and data partitioning among processors to reach an acceptable scalability. Shared Memory system, on the other hand, do not have this drawback but the number of processors sharing a memory contingent is limited. The Graphics Processing Unit (GPU), which can be considered as a *shared-memory* system, offers more cores than classical CPU based shared-memory systems. Every GPU core is less powerful than a CPU core, but their very large number allows a considerable performance increase in terms of computational time (speedup). These promising GPUs can be programmed using an automated high-level porting tool or better a low-level programming language. While automated high-level porting tools, such as openACC [2], accelerate CPU applications by just adding few directives to the code, low-level programming languages, such as CUDA [3] and OpenCL, offer the full flexibility of code rewriting. Moreover, they are reported in the literature[4] to be more efficient than high-level porting tools.

CUDA, a C-like programming language, is used to write applications that run on Nvida GPUs. Moreover,many known libraries have a GPU enabled version (cuBLAS, cuRAND,cuSpare, PETSc ...). This facilitates the code writing process. CUDA functions, which are called kernels, consist of a set of instructions intended to run on a high number of CUDA cores. The more independent instructions an application has, the more it will profit from GPU architecture.

*International Conference of Numerical Analysis and Applied Mathematics 2015 (ICNAAM 2015)*
AIP Conf. Proc. 1738, 480077-1–480077-4; doi: 10.1063/1.4952313
Published by AIP Publishing. 978-0-7354-1392-4/$30.00

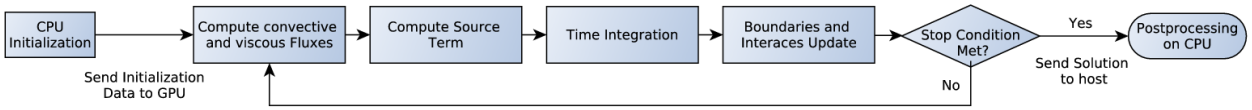


**FIGURE 1.** The Solver computes firsts inviscid and viscous Residuals and source term. Seconds the time integration takes places and finally boundaries and mesh interfaces are updated

Porting applications to GPU is, therefore, about the loosening of data dependencies. GPUs evolved, in facts, from graphics pipeline responsible of updating very fast display content. As such, the more an application is similar to image processing, which is about setting the value of a high number of pixels at the same time, the faster it will run on the GPU.

Design optimization in general profited from GPU potential [5] used to accelerate, among others, the flow and the structure evaluations. In the CFD area, particle-based solver (e.g. Lattice Boltzmann equations) are reported [6] to profit the most from GPUs. These types of solvers provide a large number of independent simple arithmetic instructions that are proportional to the number of particle used. Finite Volume (FV) solvers, on the other hand, compute fluxes on cell faces involving a number of neighbors that increases with the order of the solver. Consequently, FV solvers run more arithmetic operation and present more data dependency. In general, achieved speedups are less than with particle based solvers. Nevertheless a good management of GPU cores and GPU features, like multi-stream, can lead to speedups of 1 to 2 orders of magnitude.

In this work, a second order finite volume steady RANS CPU code has been rewritten using the CUDA language in order to run on the high-performance Tesla K40 GPU [7]. The GPU RANS solver has been then integrated in an in-house optimizer based on DE algorithm [8]. The GPU-enhanced optimizer has been finally used to optimize a supersonic compressor cascade.

## DESCRIPTION OF THE FLOW SOLVER: THEORY AND IMPLEMENTATION

The flow solver implements Reynolds-Averaged Navier Stokes (RANS) equations with the finite volume method (see Figure 1). The numerical approximation of viscous fluxes employs a central scheme. The convective fluxes are approximated using a Roe upwind scheme. Second order accuracy is achieved through the MUSCL (Monotone Upstream-Centered Schemes for Conservation Law) approach[9] with limiter. To account for turbulence the solver uses Spalart-Allmaras (SA) one-equation turbulence model. The time stepping is performed with the explicit second order 4 stage Runge-Kutta method.

A CFD solver is based in general on functions that alter the content of a mesh cell by cell. The GPU programming approach consists of replacing the serial CPU function with a GPU kernel that launches a large number of threads, which are run by CUDA cores. Every thread can be responsible for the content of one mesh cell.

Basic GPU programming is able to accelerate applications to a certain extend, but special GPU features like multi-streaming can boost the achieved acceleration furthermore. Multi-streaming, which consist of launching many kernels simultaneously, provides CUDA cores with more independent mesh cells to update, which allows more CUDA cores to be simultaneously active. The portion of simultaneous active cores is called the occupancy, which increases dramatically through the multi-streaming, hence achieving additional speedup as later illustrated.

## VALIDATION AND BENCHMARK

For viscous turbulent flows the nozzle guidance vane LS89 [10] has been used for validation. The inlet flow is subsonic but experiences an acceleration on the suction side of the vane reaching supersonic conditions which leads to a normal shock (Figure 2a). Figure 2b shows the isentropic Mach number profile of the turbine inlet guidance vane matching experimental results. The shock occurring on $\frac{\bar{X}}{X_0} \approx 0.9$ is well resolved and precisely predicted, which guarantees a confident level of accuracy.

Once the accuracy of the solver proved, a performance benchmark has been performed. A supersonic compressor cascade with a multi-block mesh has been chosen for performance comparison with a CPU serial code. Table 1 shows the speedup of 1 GPU over 1 CPU for different mesh resolutions. A higher mesh resolution presents more cells in

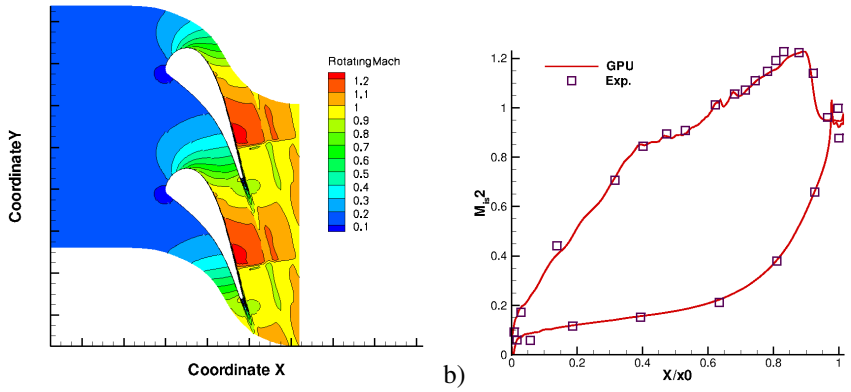


**FIGURE 2.** a)Mach contours of the transonic LS89 [10] turbine guide vane test case , b) Computed and experimental distributions of isentropic Mach number on LS89 inlet guidance vane surface ($M_{2_{is}} = 1.02$ $P_{01} = 1.605bar$)

**TABLE 1.** Speedup of Tesla K40 over Xeon(R) CPU E3-1240v3 for different mesh resolution

| Number of Cells | Speedup No Multi-stream | Speedup multi-stream | speedup improvement due to multi-stream |
|---|---|---|---|
| 25k | 8.17 | 12.74 | 55.5% |
| 100k | 17.22 | 21.37 | 24% |
| 400k | 26.9 | 28.05 | 4.1% |

every mesh block and thus decreases the number of inactive threads advantaging the GPU over the CPU. The increase of the resolution improves the speedup until a maximal speedup is asymptotically approached. The mesh has reached a size, for which all threads are mapped to cells and the portion of inactive threads is reduced to a minimum. These findings confirm the effect of multi-streaming (see section 2) on the speedup. For low resolution meshes (e.g 25K cells) multi-streaming increased the occupancy and thus improved well the speedup (55%). For higher resolutions (e.g 400k) the occupancy approaches its limit which made the multi-streaming improvement minimal (4.1%).

## APPLICATION ON OPTIMIZATION OF SUPERSONIC COMPRESSOR CASCADE

After validation the GPU-accelerated CFD solver has been integrated in an in-house optimizer called CADO [8] using the differential evolution algorithm. This GPU enhanced algorithm is used to optimize a supersonic compressor cascade designed initially for an inlet Mach number of 1.3 and a pressure ratio of 1.67. Total pressure and temperature are imposed at the inlet in addition to the flow angle while static pressure is imposed at the outlet. The flow around the pre-compression baseline design (Figure 3a) experiences a bow shock ahead of the profile leading edge, which propagates toward the adjacent blade to form a passage shock. The passage shock interacts with the adjacent blade boundary layer and induces a boundary layer separation. The achieved flow turning is 4 degrees and the static pressure ratio is 1.67, which is mainly achieved through the normal passage shock.

The optimization objective is to minimize the losses in terms of mass averaged entropy generation ($\Delta S = \frac{S_{OUT} - S_{IN}}{S_{REF}}$) at the outlet for the same flow conditions. The optimization problem has one constraint, which is to keep the flow turning equal or greater than the baseline case. The blade geometry is defined by superimposing a parametric thickness profile to a parametric camberline. The optimization variables are the y-coordinate of 5 control points of the blade angle distribution B-spline. The thickness distribution remains on the other hand fixed during the optimization. A population of 40 individuals have been evolved for 9 generations through the differential evolution method to reach at the end a total improvement of 20% in terms of entropy generation.

A comparison of the flow around the baseline and around the optimized blade shows two main differences: the normal shock turned into a partially oblique one and the passage shock moved downstream approaching the trailing

edge. The change in the flow is due to a reduction in the incidence angle on the leading edge (LE) area which reduces the losses by bringing the bow shock next to the LE and decreasing the expansion at the suction side responsible for the flow acceleration before it reaches the passage chock. Starting from the stagnation point on the LE area the flow is first accelerated on the original blade suction side due to an inappropriate incidence reaching a peak Mach number as high as 1.7. Contrary, on the optimized blade the incidence has been adjusted to the flow direction reducing the peak Mach number to 1.5 only. Further downstream both profiles decelerate the flow on the suction side through a curvature opposed to classical subsonic profiles, achieving a pre-shock Mach number of 1.6 to 1.3 for the original and optimized blade respectively by Prandl-Meyer compression waves (Figure 3).

While this optimization, which required a total of 360 flow evaluations, would run 32 days in a single CPU, it took 2 days on 2xK40 GPUs.

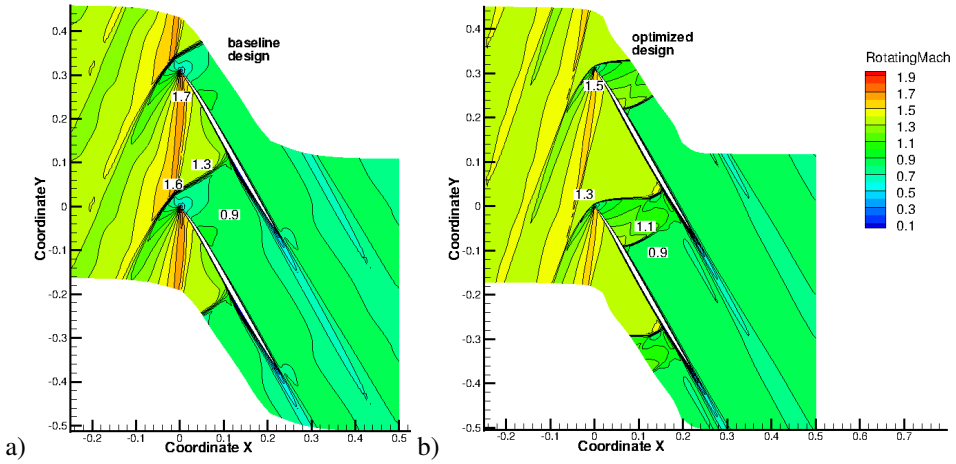


**FIGURE 3.** Mach contours of the baseline (a) and optimized (b) design of the supersonic compressor cascade ($P_{01} = 1bar$ $T_{01} = 300K$ and $\alpha_1 = -64.5$)

## ACKNOWLEDGMENTS

This research activity is funded by a Marie Curie Action as part of the European union's Framework 7 research program (AMEDEO :Project No. 316394).

## REFERENCES

1. R. Storn, and K. Price, *Journal of global optimization* **11**, 341–359 (1997).
2. CAPS Enterprise, Cray Inc., NVIDIA, and the Portland Group. (June 2013).
3. Nvidia, *CUDA C Programming Guide*, PG-02829-001 v7.0, 03/2015.
4. S. Christgau et al., "A comparison of CUDA and OpenACC: accelerating the tsunami simulation easywave," in *27th International Conference on Architecture of Computing Systems*, 2014.
5. M.H. Aissa, T. Verstraete and C. Vuik, "Use of modern GPUs in Design Optimization," in *The 10th ASMO UK / ISSMO conference on Engineering Design Optimization Product and Process Improvement*, 2014, pp. 1–2.
6. P.R. Rinaldi et al., *Simulation Modelling Practice and Theory* **25**, 163–171 (2012).
7. NVIDIA, TESLA K40 GPU Active Accelerator, Board specification (2013).
8. T. Verstraete, "CADO: a computer aided design and optimization tool for turbomachinery applications," in *2nd Int. Conf. on Engineering Optimization, Lisbon, Portugal, September*, 2010, pp. 6–9.
9. B. V. Leer, *J. Computational Physics* **32**, pp. 101–136 (1979).
10. T. Arts, M. Lambert de Rouvroit and A.W. Rutherford, Aero-thermal Investigation of a highly loaded transonic linear Turbine Guide Vane Cascade, TN 174, von Karman Institute for Fluid Dynamics (1990).