Performance Evaluation of CUDA programming for machining simulation

Felix Abecassis, Sylvain Lavernhe, Christophe Tournier, Pierre-Alain Boucard

International conference on Graphics Engineering
June 19th – 21th, 2013 Madrid,Spain
INGEGRAF – ADM – AIP PRIMECA

Performance evaluation of CUDA programming for machining simulation

ABECASSIS., Felix [(a)], LAVERNHE., Sylvain [(a)], TOURNIER., Christophe [(a)], BOUCARD., Pierre-Alain [(b)]
[(a)] LURPA, ENS Cachan, Université Paris Sud 11, 61 av. du Président Wilson, 94235 Cachan, France
[(b)] LMT-Cachan (ENS Cachan/CNRS/UPMC/PRES UniverSud Paris), 61 av. du Président Wilson, 94235 Cachan,France

Article Information

Keywords:
Machining simulation,
GPU computing,
CUDA architecture,
5-axis milling,
CAM software

Corresponding author:
Lavernhe Sylvain
Tel.: 33 1 47 40 29 85
Fax.:33 1 47 40 22 20
e-mail:lavernhe@lurpa.ens-cachan.fr
Address: LURPA, ENS Cachan, Université Paris Sud 11, 61 av. du Président Wilson, 94235 Cachan, France

Abstract

5-axis milling simulations in CAM software are mainly used to detect collisions between the tool and the part. They are very limited in terms of surface topography investigations to validate machining strategies as well as machining parameters such as chordal deviation, scallop height and tool feed. Z-buffer or N-Buffer machining simulations provide more precise simulations but require long computation time, especially when using realistic cutting tools models including cutting edges geometry.

Thus, the aim of this paper is to evaluate Nvidia CUDA architecture to speed-up Z-buffer or N-Buffer machining simulations. Several strategies for parallel computing are investigated and compared to CPU conventional and parallel computing, relative to the complexity of the simulation. Simulations are conducted with a professional CAD/CAM workstation equipped with a Nvidia Quadro4000 graphic card.

1 Introduction

All manufactured goods present surfaces that interact with external elements. These surfaces are designed to provide sealing or friction functions, optical or aero dynamics properties. The technical function of the surface requires specific geometrical deviations, which can be performed with specific manufacturing process. In particular, to reduce the cycle time of product development, it is essential to simulate the evolution of geometrical deviations of surfaces throughout the manufacturing process.

Currently, simulations of the machined surfaces in CAM software are very limited. These simulations are purely geometric where cutting tools are modelled as spheres or cylinders without any representation of cutting edges. In addition, these simulations do not include any characteristic of the actual process that may damage the surface quality during machining. Finally, these simulations do not provide the required accuracy within a reasonable computation time and the selection of an area in which the user wishes to have more precision is impossible.

However, there are many methods to perform machining simulations in the literature: methods based on a partition of the space by lines [1], by voxels [2] or by planes [3], methods based on meshes [4], etc. If we consider the Zbuffer or Nbuffer methods applied to a realistic description of both the tools and the machining path, earlier work have shown that it is possible to simulate the resulting geometry only on small portions of the surface in a few minutes [5]. Simulation results are very close to experimental results but the simulated surfaces have an area of a few square millimeters with micrometer resolution (fig. 1).

Therefore, the limits in terms of computing capacity and simulation methods restrict the realistic simulations of geometrical deviations. Regarding the hardware, NVIDIA has recently developed CUDA, a parallel processing architecture for faster computing performance, harnessing the power of GPUs.

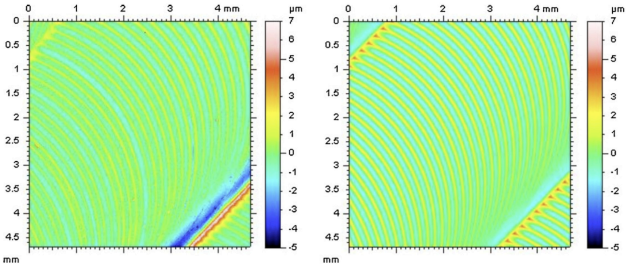

Fig. 1 Real (left) and simulated (right) topography

The aim of this paper is thus to propose an evaluation of this architecture to improve the computation time and therefore computing complexity. After having presented the machining simulation algorithm to be parallelized, constraints related to the use of the CUDA architecture are exposed. Then, different solutions to the problem of parallelization with CUDA are investigated. Finally a comparison of the computing time between GPU and CPU processors is proposed on several 3 and 5-axis milling examples with real time zooming.

2 Computation algorithm

The computation algorithm relies on relies on the Zbuffer method [1]. This method consists in partitioning the space around the surface to be machined in a set of lines, which are equally distributed in the x-y plane and oriented along the z-axis. The machining simulation is carried out by computing the intersections between the lines and the tool along the tool path. The geometry of the tool is modelled by a triangular mesh including cutting edges, which allows

to simulate the effect of the rotation of the tool on surface topography. The tool path is whether a 3-axis tool path with a fixed tool axis orientation or a 5-axis tool path with variable tool axis orientations.
In order to simulate the material removal, intersections with a given lines are compared and the lowest is registered (fig. 2). The complete simulation requires the computation of the intersections between the *N* lines and the *T* triangles of the tool mesh at each tool posture *P* on the tool path. The complexity *C* of the algorithm is thus defined by:

$$C = N \times T \times P \quad (1)$$

If we consider a finish operation on a 5mm x 5mm patch of surface with a resolution of 3 µm, this leads to 2,250,000 lines. The tool mesh contains 3,000 triangles and the tool path contains 70,000 postures including the tool rotation around the spindle axis. The number of intersection to compute is equal to 4.7×10^{14}. This technique can be accelerated by decreasing the number of tests by first calculating the intersection with the bounding box of the tool and using data structures such as Bounding Volume Hierarchy or Binary Space Partitioning trees.

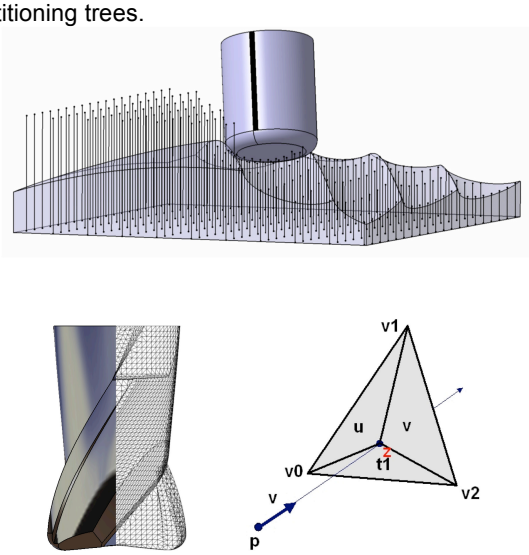

Fig. 2 Zbuffer simulation method

In the case of Z-buffer, all lines are oriented in the same direction, which allows many optimizations on both the number of intersection tests to be performed, and the number of operations required for each test. For each triangle, its projection on the grid and its 2D bounding box is calculated (fig. 3). The lines outside the bounding box (shaded circles) are not tested; they can not have an intersection with the triangle. A double loop is then performed to test the intersection of each line within the bounding box with the 2D projection of the triangle. The lines may intersect with the triangle (green) or not (red). If the intersection exists, the height is calculated and then compared with the current height of the line.
As there are no dependencies between the milling process at different locations on the tool path, each of theses intersections could be carried out simultaneously. However, due to memory limitations and tasks scheduling, the parallel computing of these intersections on graphics processing units with CUDA has to be done carefully.

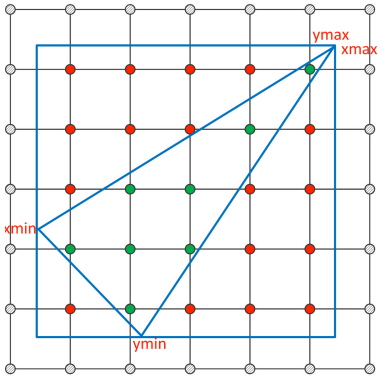

Fig. 3 2D bounding boxe

3 Compute Unified Device Architecture

CUDA is a parallel computing platform that unlocks programmability of NVIDIA graphic cards in the goal of speeding up the execution of algorithms exhibiting a high parallelization potential. Algorithms are written using a modified version of the ANSI C programming language (CUDA C) and can consequently be executed seamlessly on any CUDA capable GPU [6].
The strength of the CUDA programming model lies in its capability of achieving high performance through its massively parallel architecture. In order to achieve high throughput, the algorithm must be divided into a set of tasks with minimal dependencies. Tasks are mapped into lightweight threads, which are scheduled and executed concurrently on the GPU. 32 threads are grouped to form a warp. Threads within the same warp are always executed simultaneously; maximum performance is therefore achieved if all the 32 threads are executing the same instruction at each cycle. Warps are themselves grouped into virtual entities called blocks; the set of all blocks forms the grid, representing the parallelization of the algorithm (fig. 4).

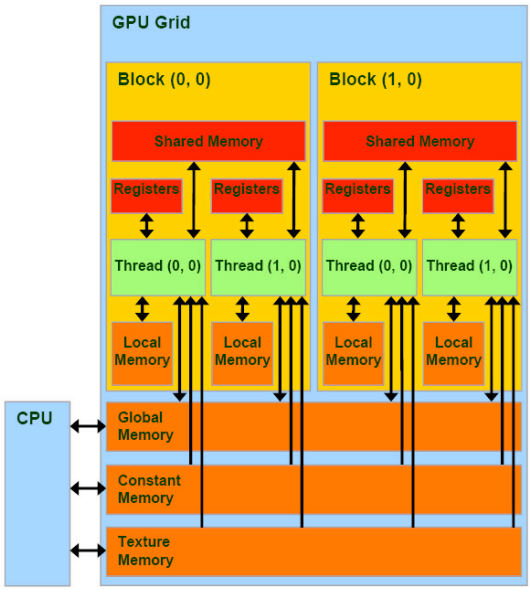

Fig. 4 Cuda architecture

Threads from the same block can be synchronized and are able to communicate efficiently using a fast on-chip memory, called shared memory, whereas threads from different blocks are executed independently and can only

communicates through global (GDDR) memory of the GPU.
The number of threads executing simultaneously can be two orders of magnitude larger than on a classical CPU architecture. As a consequence, task decomposition should be fine-grained opposed to the traditional coarse-grained approach for CPU parallelization. The combination of this execution model and memory hierarchy advocates a multi-level parallelization of the algorithm with homogeneous fine-grained tasks; dependent tasks should be mapped to the same warp or block in order to avoid costly accesses to global memory. In order to harness the power of the massively parallel architecture of GPUs, a complete overhaul of the algorithm is often required [7]. The scalability of the CUDA programming models stems from the fact that, thanks to this fine-grain task model, the number of threads to execute generally exceeds the number of execution units on the GPU, called CUDA cores, for those threads. As more CUDA cores are added, e.g. through hardware upgrades, performance will increase without requiring changes to the code. Another benefit of this fine-grained decomposition is the ability to hide latency, if a warp is waiting for a memory access to complete, the scheduler can switch to another warp by selecting it from a pool of ready warps. This technique is called Thread Level Parallelism (TLP). Latency hiding can also be done within a thread directly, by firing several memory transactions concurrently, this technique is called Instruction Level Parallelism (ILP).
Despite continuous efforts by NVIDIA to improve the accessibility of CUDA, the learning curve remains steep. Attaining high performance requires careful tuning of the algorithm through a detailed knowledge of the CUDA execution model.

4 Parallel computation strategies

The basic algorithm consists in determining whether there is an intersection between a line and a triangle associated to a position of the tool. Given this three variables on which the algorithm iterates during the sequential computation, there are numerous possible combinations to affect threads and browse the set of lines, triangles and positions. The most interesting possibilities are developed hereafter.

4.1 The line approach

Each CUDA thread is assigned to a line of the Zbuffer grid. A thread calculates the intersection of the line with the tool for all positions along the path. To reduce the number of intersection tests, the intersection with the bounding box of the tool in a given position is first calculated. If there is intersection, each triangle of the tool is then tested. The advantage of this approach is that there is no need to use atomic operations since a single thread calculates the minimum height of cut for each line. However, this approach cannot be used with bounding boxes which significantly increases the number of calculations.

4.2 The tool position approach (**coarse**)

Each thread is assigned to a position of the tool and applies the Z-buffer algorithm for every triangle of the tool mesh for this position. The granularity of tasks is high: if the amount of triangles to be processed is large, each thread will run for a long time. If the computation time between threads is heterogeneous, some threads of a warp may no longer be active, and therefore the parallelism is lost. A thread may affect the cutting height of several lines so multiple threads can update a line and global memory access conflicts appear. Atomic operations proposed by CUDA are then used to allow concurrent update the height of the lines. This method is the one implemented in the CPU configuration.

4.3 The tool triangle approach (**inverted**)

Each thread is assigned to a triangle of the tool mesh and applies the Z-buffer algorithm for the triangle for all positions along the path. In the case of the 3-axis machining, the dimensions of the triangle projection on the grid remain constant, which allows optimization possibilities. However, in 5-axis milling, this property is no longer valid. Each position involves a new transformation matrix to be recovered in memory, which increases significantly the number of memory transactions. This method is advantageous in the case of a 3-axis tool path but is not suited for 5-axis tool path. The same remarks apply the previous approach on the granularity of tasks and conflicts in global memory access.

4.4 The triangle and position approach (**fine**)

In this approach, each thread is assigned to a single triangle in one position. As the granularity is smaller, the risk of a bad balance workload disappears. However, in return the number of threads is much higher: the management and scheduling of billions of threads involve additional workload for the task scheduler CUDA and thus a significant increase in computation time.

5 Numerical investigations

5.1 Cases study

Different configurations of trajectories and tools have been used for testing. A first setting, called **random** (fig.5), for which random positions are generated on a plane and a second setting, where a plane is machined along a downward spiral called **spiral** (fig.6). In both cases, the ball-end tool is 100 times smaller compared to the dimensions of the surface. On the other hand, an industrial configuration is proposed, which consists in the 3-axis machining of a plastic injection mold to produce polycarbonate optical lenses for ski mask (fig. 7).

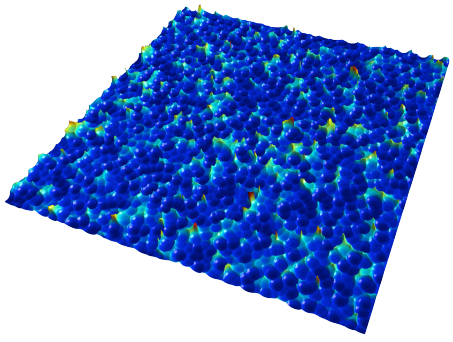

Fig. 5 Random case

Both the **roughing** with a torus tool and the **finishing** operation with the ball end tool are used in the proposed benchmark. The finishing simulation is compared to an

industrial CAM software simulation in terms of visible defects and computation time. At last, a small scale simulation benchmark, called **Cutting**, is proposed with the mesh of a ball end mill including worn cutting edges and spindle rotation (fig. 8) which lead to numerous tool positions and large tool mesh size (tab. 1).

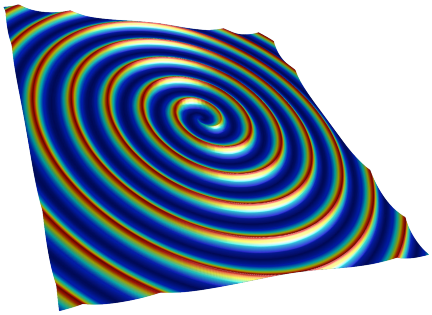

Fig. 6 Spiral case

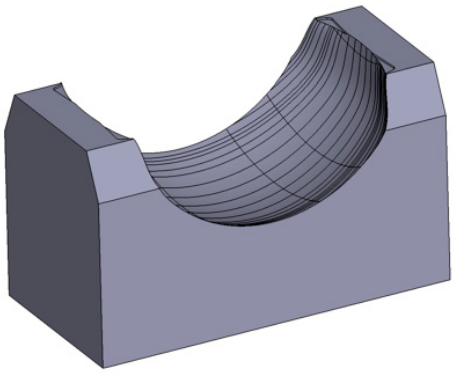

Fig. 7 Ski mask injection mold

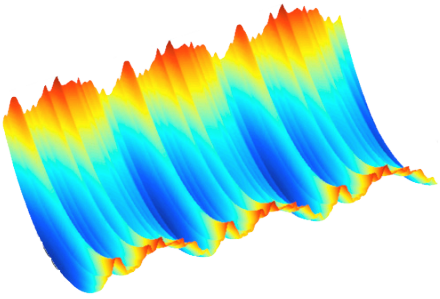

Fig. 8 Local simulation with worn cutting edge

One of the objectives is to be able to dynamically zoom on the part and update the simulation. Thus the grid size is constant and set to 1024x1024 lines, regardless of the zoom factor. Therefore, the above configurations are also studied with or without a zoom on the surface, which changes the size of the problem (tab. 1 and tab. 2).

	Tool position P	Tool	Mesh size T
Random	200000	Sphere	320
Spiral	200000	Sphere	1984
Roughing	345848	Torus	25904
Finishing	3015072	Sphere	1984
Cutting	144001	Sphere	36389

Tab. 1 Without zoom

	Tool position P	Tool	Mesh size T
Random	135	Sphere	320
Spiral	206	Sphere	1984
Finishing	129	Sphere	1984

Tab. 2 With zoom

5.2 Hardware configurations

Hardware configurations used for the benchmarks are the followings:

CPU: Intel Xeon X5650 - 2.67GHz 6 cores, 12 virtual cores (hyperthreading) OpenMP, SSE

GPU: Nividia Quadro 4000 - 0.95 GHz 8 multiprocessors, 256 CUDA Cores

5.3 Computation time analysis

Excluding cases of zoom, GPU implementation is on average 5 times faster than the CPU implementation with the engine **GPU Coarse** (1 thread handles all triangles in one position) (tab. 3).
When the number of triangles in the mesh of the tool is too low, the number of threads used in the engine **GPU Inverted** is much lower than the number of threads that can be executed simultaneously on all multiprocessors. Performance is worse (random, spiral and finishing). When the number of triangles increases (random to spiral, spiral to roughing and roughing to cutting), performance improves accordingly.
With the engine **GPU Fine** (1 thread processes one triangle in one position), there are too few lines in the bounding box of each triangle because the triangles are very small compared to the size of the grid. Each thread is little busy and time is lost to launch these threads.

Case	**Time (ms)**	**GPU Speed-up**		
	CPU	Coarse	Inverted	Fine
Random	469	6.2	0.3	2.4
Spiral	1414	4.7	2.5	1.9
Roughing	37766	5.8	4.7	1.8
Finishing	218203	5.5	1.3	2.7
Cutting	22882	8.3	7.8	2.3

Tab. 3 Configurations without zoom

In micro geometry configurations, the GPU speed-up doesn't meet expectations (tab. 4). Since the number of positions with zoom is much lower than the number of threads that can be executed simultaneously on all GPU multiprocessors, the granularity of the simulation engine **GPU Coarse** is too high: the available parallelism is not exploited. The computation engine must exhibit a much finer granularity, such as the **GPU Fine** engine, that should be used to make best use of the number of threads.

Case	**Time (ms)**	**GPU Speed-up**		
	CPU	Coarse	Inverted	Fine
Random	195	0.1	0.2	1.8
Spiral	368	0.2	0.9	3.2
Finishing	825	0.08		0.3

Tab. 4 Configurations with zoom

Performance tests with or without zoom shows that the CUDA kernel must be chosen depending on the simulation configuration. Conversely, it is not a problem

with the CPU engine because the number of available threads is much lower than the number of positions.
During the initial implementations, the acceleration factor was more important between GPU and CPU, but successive optimizations have, systematically, further improved the CPU engine rather than GPU implementations. The difference between the two versions is reduced over the optimizations.

5.4 Simulation results analysis

Regarding the ski mask injection mold, a tangency discontinuity has been introduced along the vertical symmetry axis in the middle of the lens. The machining simulation should emphasize this geometrical deviation. Despite using the best settings offered by the CAM software, the rendering of the simulation is not enough precise to detect the groove generated during tool path computation (fig. 9). On the other hand, the proposed simulation allows without any zoom to highlight this defect (fig. 10).

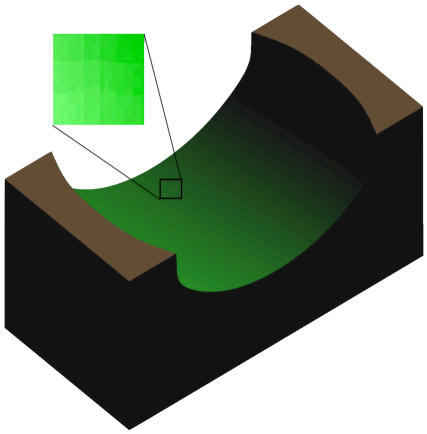

Fig. 9 CAM Software simulation

6 Conclusion

Machining simulations in CAM software are mainly used to detect collisions between the tool and the part on the whole part surface. It is difficult, if not impossible, to restrict the simulation to a delimited area in which the accuracy is much better to control the surface topography. To overcome this issue, this paper presents some opportunities to speed-up machining simulations and to provide multi scale simulations based on Nvidia CUDA architecture.

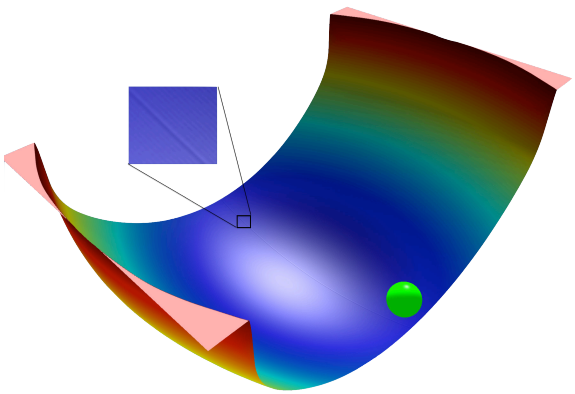

Fig. 10 3-axis GPU mask simulation

The results show that the use of the CUDA architecture can significantly improve performance on the computation time. However, if the granularity of tasks is not set correctly, the massively parallel CUDA architecture will not be used and the implementation may be slower than CPU. It is therefore necessary to adapt the parallelization strategy for the type of simulation, namely large scale or small scale.
Regarding the future works, faster simulation software could help to introduce stochastic behaviour and simulate tool wear and the use of abrasive tool could improve the quality of the surface topography simulation.

Acknowledgement

This work is supported by the Farman Institute of Ecole Normale Supérieure de Cachan.

References

[1] R. Jerard, R. Drysdale, K. Hauck, B. Schaudt, and J. Magewick. *Methods for detecting errors in numerically controlled machining of sculptured surfaces*. IEEE Computer Graphics and Applications 9, 1 (1989) pp 26-39.

[2] D. Jang, K. Kim, and J. Jung. *Voxel-based virtual multi-axis machining*. International Journal of Advanced Manufacturing Technology 16 (2000) pp 709-713.

[3] Y. Quinsat, L. Sabourin, and C. Lartigue. *Surface topography in ball end milling process: Description of a 3d surface roughness parameter*. Journal of materials processing technology 195, 1-3 (2008) pp 135-143.

[4] W. He and H. Bin. *Simulation model for cnc machining of sculptured surface allowing different levels of detail*. The International Journal of Advanced Manufacturing Technology 33, (2007) pp 1173-1179.

[5] Y. Quinsat, S. Lavernhe, C. Lartigue. *Characterization of 3D surface topography in 5 axis milling*. Wear 271, 3-4 (2011) pp 590-595.

[6] NVIDIA (2012). CUDA C Programming Guide. http://developer.nvidia.com/cuda/cuda-downloads.

[7] R. Farber, R. CUDA Application Design and Development. Elsevier Science. (2011)

.