PARALLELIZATION OF AN ULTRASOUND RECONSTRUCTION ALGORITHM FOR NON DESTRUCTIVE TESTING ON MULTICORE CPU AND GPU

Antoine Pédron[1]*, Lionel Lacassagne*[1]*, Franck Bimbard*[2] *and Stéphane Le Berre*[1]

[1] CEA, LIST, F-91191 Gif-sur-Yvette, France
[2] Institut d'Electronique Fondamentale, UMR 8622, Université Paris-Sud 11, F-91405 Orsay, France

ABSTRACT

The CIVA software platform developed by CEA-LIST offers various simulation and data processing modules dedicated to non-destructive testing (NDT). In particular, ultrasonic imaging and reconstruction tools are proposed, in the purpose of localizing echoes and identifying and sizing the detected defects. Because of the complexity of data processed, computation time is now a limitation for the optimal use of available information. In this article, we present performance results on parallelization of one computationally heavy algorithm on general purpose processors (GPP) and graphic processing units (GPU). GPU implementation makes an intensive use of atomic intrinsics. Compared to initial GPP implementation, optimized GPP implementation runs up to $\times 116$ faster and GPU implementation up to $\times 631$. This shows that, even with irregular workloads, combining software optimization and hardware improvements, GPU give high performance.

Index Terms— non-destructive testing, ultrasonic reconstruction, parallelization, general purpose processors, graphic processing units

1. INTRODUCTION

Non destructive testing (NDT) consists in examining specimen or material integrity without damaging it. Ultrasonic Testing is one of the most used method, because of its sensitivity, its depth penetration, and its accuracy for positioning and dimensioning internal defects as well as its simplicity of implementation.

Ultrasonic testing consists scanning the component under examination and acquiring signals at each probes positions. Acquired data are processed for visualization and analysis purpose, and different views are generally proposed:

- *Ascan*: ultrasonic signal's curve (amplitude in function of time)
- *Bscan*: two dimensional cross-section image view, which visualizes the amplitude of signals through a colormap. The horizontal axis corresponds to a one dimensional probe scanning while time of flight is represented on vertical axis.
- *Cscan*: two dimensional presentation as a top view of the testing. The two axis correspond to two dimensional scanning of the probe and data is cumulated along signals.

Besides these views visualizing raw data, different reconstruction algorithms can be proposed to give a representation of the data in real space.

This work is done in the context of the development at CEA LIST of the CIVA software platform dedicated to simulation and processing of NDT data [1]. Because of the complexity of the data to be processed, computation time remains a limitation for an optimal use of reconstruction algorithms. Acceleration of reconstruction and visualization algorithms would be a great step forward for the tools of analysis and automatic diagnosis. In this perspective, parallelization and exploitation of GPU are investigated. An ultrasound reconstruction algorithm called True Cumulated Views (TCV) has been chosen for optimization and parallelization with the two most common hardware platforms used nowadays, general purpose processors (GPP) and graphic processing units (GPU). GPU are well known to give high speedups, particularly when computations are highly parallel [2].

The article is organized as follow: first of all, section 2 presents TCV algorithm and software optimization. Then, parallel implementations are detailed in section 3. After that, section 4 shows experimental results. Finally, conclusion is in section 5.

2. TRUE CUMULATED VIEW ALGORITHM

The *True Cumulated View* algorithm (TCV) offers a complete visualization of the data through top, side and front views of the specimen (figure 1). This algorithm postulates that both the specimen geometry and material properties are known. In complex cases, the geometry can be given as a CAD description. Ultrasound reconstruction is a widely discussed topic and usually, reconstructions are based on Bscans. TCV reconstruction is differentiated by the fact that inputs are not Bscans but raw signals and geometry information [3]. In subsection 2.1, a theoretical view of the algorithm is given. Then, 2.2 presents a pseudo-code based on initial implementation. Finally, 2.3 shows software optimization.

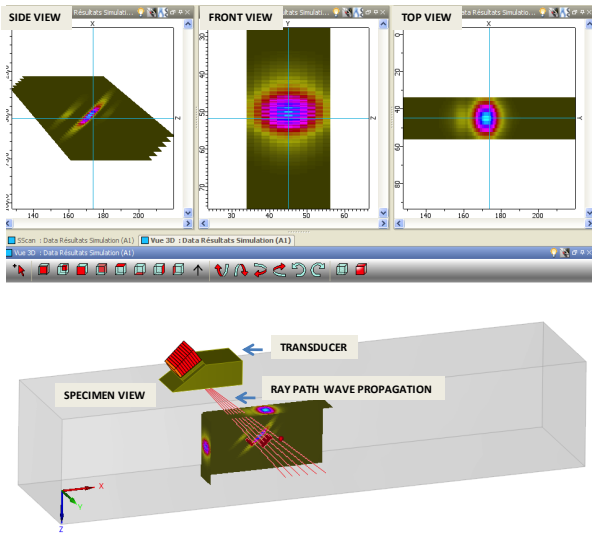


Fig. 1. In the top part : views computed by True Cumulated View algorithm. In the bottom part : views assembled on a 3D model.

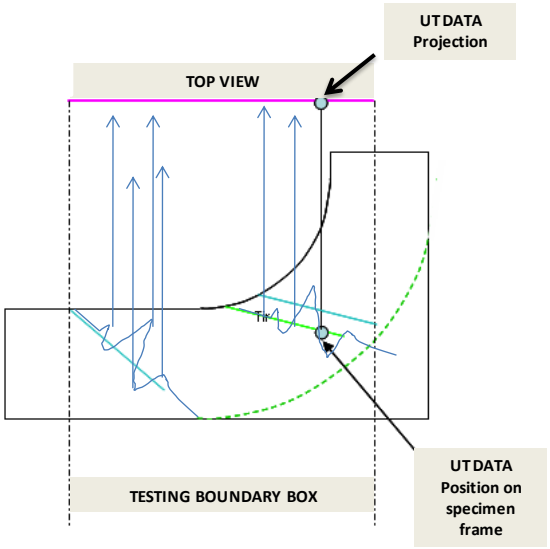


Fig. 2. Top View computation pattern.

2.1. Algorithm description

The inputs are:

– N signals (N is the number of ultrasonic shots) : amplitude in function of time. The signals are results of an experimental acquisition or a simulation calculation.

– N ray-paths : geometrical polylines, which represent the path of theoretical beam deviation inside the specimen. At each vertex of the polyline is associated the time of flight value of the wave to reach this point. Time of flight between two vertex are obtained through simple linear interpolation. These ray-paths are precalculated with CIVA simulation code [4].

– Area of reconstruction : 2D regular grid (that can be confused with the final image).

The Output is :

– An image : amplitude is assigned to each point of the reconstruction area and is represented by a color code.

Description of the algorithm (illustrated on figure 2):

1. First step: Projection of the signal to the ray-path. Each value (amplitude,time) of the signal, can be localized on the ray-path and converted to (amplitude, (x,y,z)) value.
2. Second step: for each value (amplitude, (x,y,z)), projection of the rectangular area to the final grid (figure 3). Then, dilation of the amplitude value to a rectangular area around the point (x,y,z). For each pixel, selection of the max value between projected value and current pixel image value.

This algorithm has an **irregular workload** whose intensity depends on the data. Ray-paths can be either well aligned or disorganized, short or long, with a few or many segments and in most cases can be source of memory conflicts in case of parallelization.

2.2. Initial implementation

The initial implementation is a direct mapping of the *physical* algorithm. Sequential implementation (Algo. 1) samples each ray-path along and, in order, each sample is projected on the surface plan with a dilation of its energy on its neighborhood using a max operator. Dilation operation is very heavy on resources because of the number of memory accesses needed.

Input: raypaths array r
Input: signals arrays $sigs$
Input: sampling step δt
Input: dilation window size $(\delta u, \delta v)$
Output: image T of size $n \times m$
for *each ray-paths* r_i **do**
  for *each segments* s_j *of* r_i **do**
    get end points(position, time):
      $(M^i_j(s_j), t^i_j)$ and $(M^i_{j+1}(s_j), t^i_{j+1})$
    sample segment with δt
    for *each sample* e_k *of* s^i_j **do**
      get amplitude a from $sigs$ for k on r_i
      $e_k(x\prime, y\prime) \leftarrow$ project3Dto2D$(e_k(x, y, z))$
      compute $T[y\prime, x\prime] \leftarrow \max(a, T(y\prime, x\prime))$
      dilate$(\delta u, \delta v)$ on $T[y\prime, x\prime]$

Algorithm 1: Initial implementation of TCV algorithm.

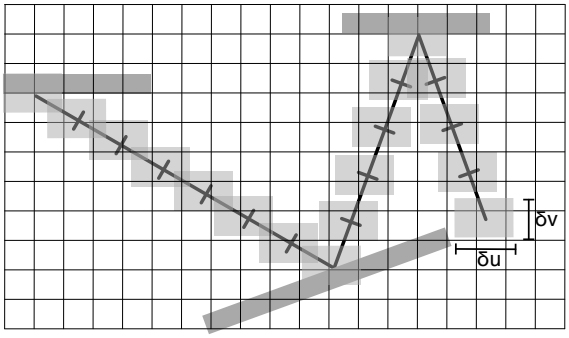


Fig. 3. Illustration of strides between two projections. Energy dilation is applied to the grey area.

2.3. Software optimization

Major optimization has been done on the dilation step. Dilation area is constant for a full view reconstruction. With this hypothesis, every projected point is processed in the same way. Because of the idempotency property of the max operator, it is possible to postpone the on-the-fly $\delta u \times \delta v$ dilation by a unique post-processing dilation. Instead of applying a $\delta u \times \delta v$ dilation kernel for each projected point, this kernel is applied only once to all pixels on the image. On-the-fly dilation is replaced by a post-processed dilation performed after the processing of all ray-paths and using kernel separability to reduce compute complexity [5]. Moreover, this step can be even more optimized by implementing Van Herk algorithm [6] [7]. This type of optimization is well-know and used in other projection algorithms such as maximum intensity projections [8].

3. PARALLEL IMPLEMENTATIONS

Ultrasonic image reconstruction has to deal with important amount of data. Numerous elements are processed with the same operation. This pattern can be mapped onto GPU computing pattern which is based on large data sets with high parallelism and minimal dependency between data elements. Unlike GPP, GPU has to run a large number of threads to be efficient. GPU threads are not like CPU ones : launching a kernel with ten thousands threads takes about 12 μs with synchronization [9]. As a result, parallelization strategies are usually different. Two different strategies are discussed in this section, both on GPP and GPU, in order to give a fair comparison. Both GPP and GPU have two implementations : one with dilation like the initial implementation and another one with post-processed dilation as discussed in subsection 2.3.

3.1. GPP implementations

The GPP approach is straight-forward and, if using a p-core GPP, ray-paths are distributed among p threads (figure 4). These threads are created by using the OpenMP library that automatically distributes a loop computation without modifying the existing code. In addition, each of the previous threads runs the TCV algorithm and generates its own image. TCV algorithm being irregular, it cannot be vectorized by using SIMD instructions.

Then the images are merged by extracting the maximum value in each pixel (reduction). This step is parallelized by using the OpenMP library and, being regular, is also vectorized with SSE2 SIMD instructions. Then, the dilation step, which is discussed in section 2.3, is also parallelized but not vectorized.

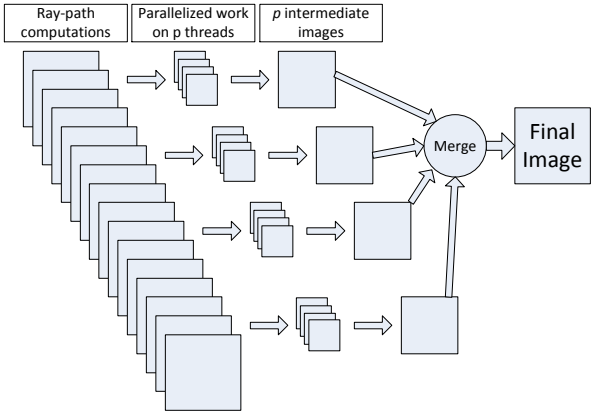
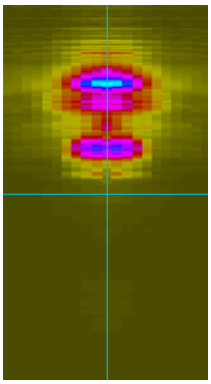


Fig. 4. Parallelization scheme for a p-core GPP.

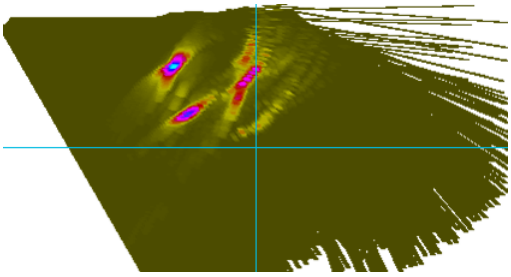
3.2. GPU implementations

GPU implementation strategy is slightly different. Assuming that GPU need to launch a very large number of threads (thousand of them), multiple image reconstruction cannot be used. Instead, one unique output image is used. Since different threads can potentially access the same address at the same time, dilation has to be done synchronously to be thread safe. These operations exist on Nvidia GPU since *compute capability 1.1* hardware and are named *atomics*. They allow to create a mutual exclusion section to enforce serial access to memory. At the moment, sections are restricted to basic operators like max, inc, add, or a few other unitary operators. These intrinsics are not advised for an intensive usage like TCV which can potentially write hundreds of millions points.

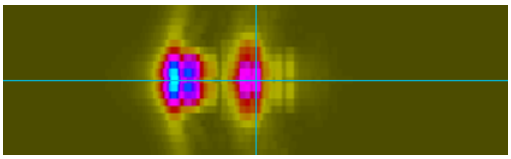
Atomics are slow with 1.x hardware, but Nvidia announced with the latest generation of GPU (2.x) that they have improved their *atomic* instructions up to a factor $\times 20$ [10][11]. Two aspects have to be differentiated: instruction latency and lock latency. On one hand, instruction latency is the cost of the



(a) Front View

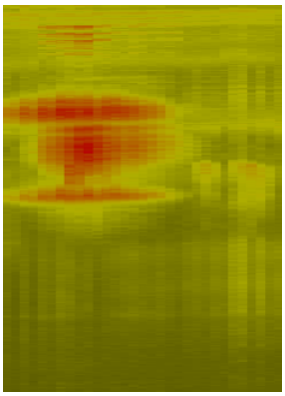


(b) Side View



(c) Top View

Fig. 5. Reconstruction of EXZ dataset.



(a) Front View

(b) Side View

(c) Top View

Fig. 6. Reconstruction of PMF dataset.

access, alone. On the other hand, lock latency is the duration of the memory lock in order to execute a thread safe operation. If access are sparse enough, lock latency will not impact on performance because no thread will access the same memory at the same time. When collisions occur, and multiple threads access the same data concurrently, *atomic* intrinsics compute time is increased because it has to add instruction latency to lock latency.

Like CPU, two implementations have been realized. Both TCV kernels remove the upper loop of algorithm and instead, one thread per ray-path is launched. Each thread computes its own ray-path projection, using the same image as all threads, with the use of `atomicMax` operator. Basic implementation apply the dilation in the inner-loop, optimized kernel only write one pixel per projected point. The latter implementation uses another kernel to apply dilation which is based on CUDA SDK image convolution [12]. Since computations are very irregular on TCV kernel, depending on numerous parameters (dataset, view, zoom, image size, etc.), thread block size has been set to 64 (this value is discussed in the following section).

4. BENCHMARKS

Benchmarks have been realized on a quad Core Intel E5472 Harpertown at 3.0 GHz and two Nvidia GPU : Tesla C1060 with 240 cores running at 1.3 GHz and Tesla C2070 with 448 cores running at 1.5 GHz. GPP codes were using Visual Studio 2010 compiler with SSE intrinsics and OpenMP 2.0. For GPU, CUDA 3.2 runtime has been used with version 266.58 of driver.

Two ultrasound defect response datasets have been selected as a representative subset of data that TCV use to deal with. PMF is a 150k ray-paths with 1k sample per signal set representing a 150M point projection for a full reconstruction whereas EXZ is a 15k ray-path with 2k samples (30M point projection). PMF has regular ray-paths compared to EXZ's which is a weld evaluation. Full reconstruction of different views are shown on figures 5 and 6.

First subsection presents a benchmark of *atomics* performance on TCV computation. Then, subsection 4.2 discusses the block size parameters. Subsection 4.3 shows up access sparsity and 4.4 discusses gpp scalability on TCV computation. Finally, subsection 4.5 deals with the overall performance results for a complete reconstruction on GPP and GPU.

4.1. Impact of using *atomic* instructions

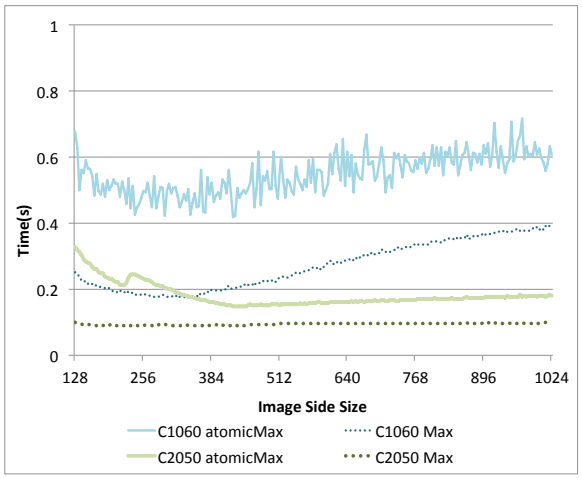


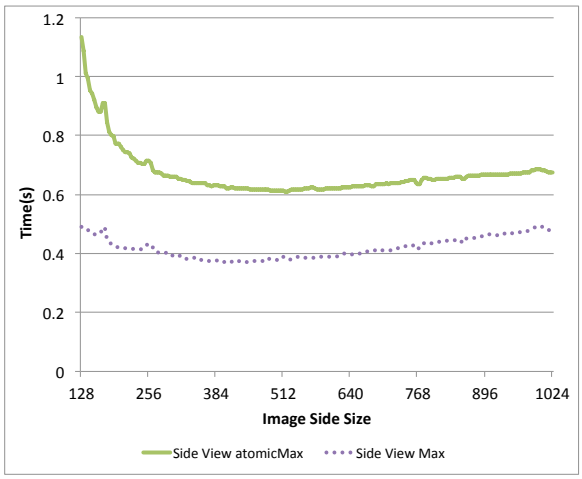
Fig. 7. Pseudo-random `atomicMax` and max operator evaluation for an image size from 128×128 to 1024×1024 on Tesla C1060 and C2070

Given that GPU implementations are using *atomics* intrinsics and pseudo-random accesses, one can wonder about the performance of these operations.

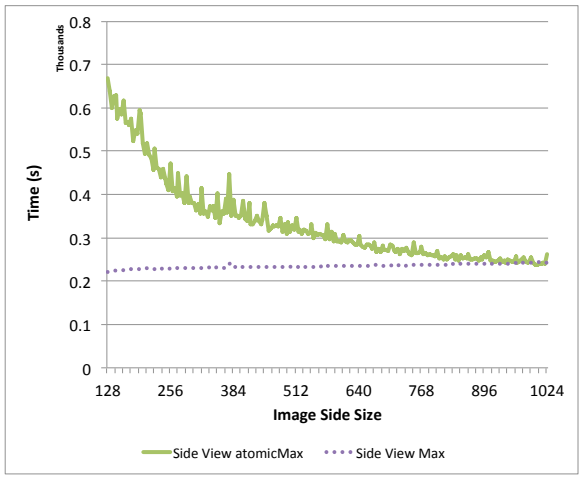
On C1060, global memory is not cached [13] whereas C2070 adds two levels of cache. L1 is local to a streaming multiprocessor whereas L2 is global. L1 cache should not be useful as in our case TCV has an intense atomics use whose memory transactions are done in L2. But since concurrent access intensity is low (*i.e.* compared to a histogram view), one can wonder how caches impact on TCV and more generally what cost has to be paid.

Figure 7 presents the results of a pseudo-random access computation of max and `atomicMax` operators with C1060 and C2070. 150M points are written in a pseudo-random manner in a 1D array so as to be able to compare with the experimental results of TCV.

C1060 `atomicMax` curve is very noisy and basically slow. When compared to a simple max operator (ignoring concurrent accesses), `atomicMax` is about $\times 2.5$ slower. With C2070, synchronization is still expensive when access intensity is high but the gap between `atomicMax` and max has been reduced with a slowdown lower than 50% for images larger than 350×350 meaning that *atomic* instructions latency has been greatly improved.



(a) C1060



(b) C2070

Fig. 8. TCV GPU computation time for a fixed number of pixel accesses (PMF dataset, 150M point projection) with a variation on image size between 128×128 and 1024×1024 for C1060 (a) and C2070 (b).

Figure 8 shows the main step of TCV on C1060 and C2070 with `atomicMax` and max operators. For images larger than 192×192, C1060 curves are outspread of a $1.5\times$ factor meaning that lock latency is basically free when access intensity is high enough. Lock latency shows up for smaller images with up to a $3\times$ factor for 128×128 images. That show how expensive atomics intrinsics are on old generation hardware.

On C2070, both curves are different than C1060's. Max curve is very stable with a 10% compute time increase between 128×128 and 1024×1024. The main difference come from `atomicMax` curve which is deceasing over time like C1060's but instead of following max curve with a factor, it reaches max performance for images greater than 800×800. This means `atomicMax` computation is not more expen-

sive than a simple max operation when concurrency does not impact. Moreover, performance gain is significative between C1060 and C2070 with a $\times 1.5$ to $\times 2.8$ factor between `atomicMax` computation for 1024×1024 images.

In TCV algorithm, the use of *atomics* is mandatory to avoid errors of computations. But depending on algorithm parameters, concurrent accesses are not the major part of the computation. With Fermi boards, *atomics* can be used like others non-atomic operators for random accesses as soon as concurrency intensity is not too high.

4.2. Block Size settings

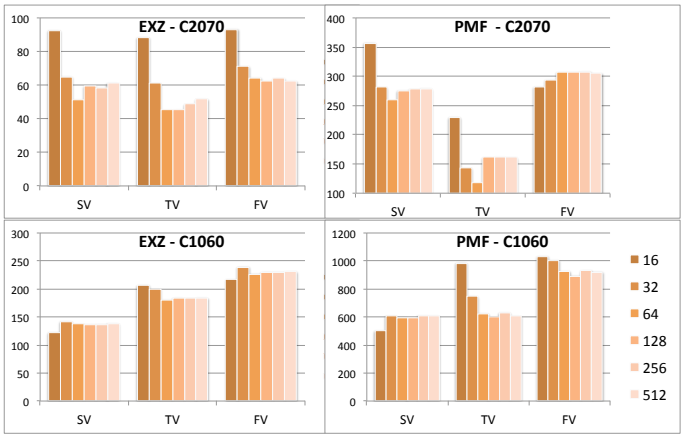


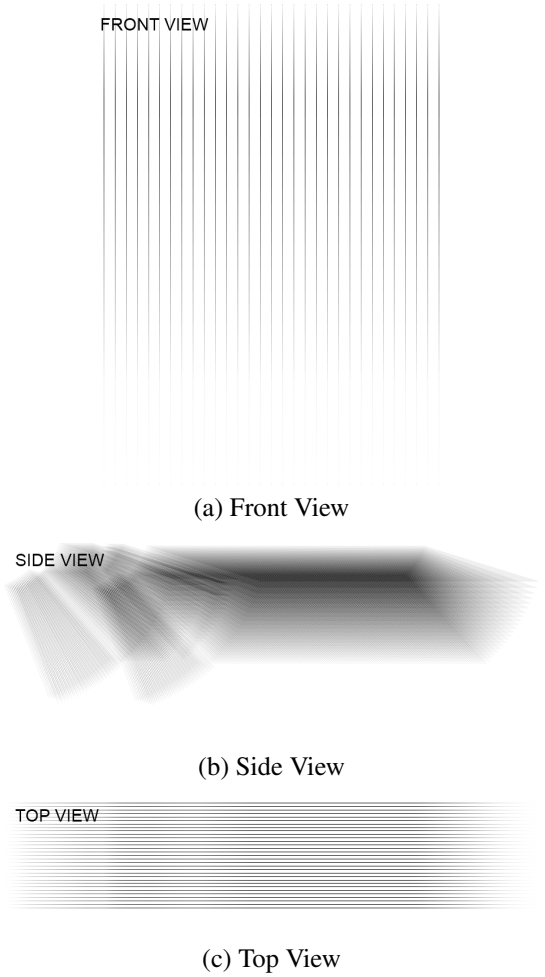
Fig. 9. TCV computation time (ms) in function of thread block size on both PMF and EXZ datasets with side, top and front views (SV, TV, FV) on C2070 (up) and C1060 (down).

In order to get the highest performances on GPU, thread bloc size has to be explored. Block size is a trade between GPU architecture specifications and kernel code. More precisely, GPU streaming multiprocessor capabilities has to be known, such as maximum number of running threads, number of registers, etc. Since TCV computations are irregular and very data dependent, tests have been ran to check if and how a block size has an impact on performance.

Figure 9 shows TCV computation time on EXZ and PMF data samples on C1060 and C2070 with different block sizes. As can be seen for the C1060, depending on the data or view, best block size value varies. The amplitude of variation is different for each computation. On C2070, same observations can be done, with also different amplitude variations. C2070 does not perform well with small bloc size probably because of the dual warp scheduler.

Best overall value for C1060 is 128 unlike C2070 which is 64. Taking into account that the performance variation is small between 64 and 128 block size, 64 has been chosen for all computations.

4.3. Access sparsity



(a) Front View

(b) Side View

(c) Top View

Fig. 10. Histogram of accesses for respectively side, top and front views of PMF dataset with optimized kernel.

In order to get a better understanding of the differences between the three views and associated computation times, an histogram of the different memory access has been realized for the different views. Figure 11 shows that PMF front and top views are very different than side view (same observation for EXZ). For each view, the total amount of memory access is the same. The histogram focus on the *distribution* of these accesses and the memory footprint. For example, on side view, some pixels are written 3000 times, compared to up to 8000 times for the top view and 14000 times for the front view.

This observation is very interesting because it means that even with higher access intensity, atomic *intrinsics* can be faster when locality is higher than fully random accesses. Top view is the fastest because of its memory alignment. Given that atomics are passing through L2 cache, some accesses are using this cache to improve performance. On C1060, beha-

vior is different because its L2 cache is read-only compared to Fermi's which is read-write. View differences can be verified with profiler data where top view implies less *DRAM writes* than other views (table 2) which means that many accesses are using cache features.

	GPP0 (initial)		
	time (ms)	*# DRAM writes*	*ipc*
View type	PMF dataset / 800×800		
Side View	257	2.56e7	0.60
Top View	109	1.04e7	0.87
Front View	295	1.96e7	0.56

Table 2. Profiling sample of PMF dataset processing for the three views on C2070.

Comparison between side and front view is harder because with front view, lock latency is high. What we can see on profiling data is that computation time is linked to *ipc* (instructions per cycle metric). Peak *ipc* performance is 2.0 on Fermi GPU (compute capability 2.0) meaning that only 25% to 33% of peak *ipc* is reached here. The point is that memory aligned view is in favor compared to the two others. This can seem obvious, but even though accesses strides are not predicable, performance is still better with these views.

4.4. GPP scalability

GPP parallelization is very efficient, on both GPP implementations. GPP0 is the initial implementation and GPP1 the optimized one. Figure 11 shows up front view computation time for PMF dataset for different image size for the two implementations. With GPP0 implementation, computation time depends directly on image size. Between a 128×128 reconstruction and 1024×1024, up to a factor $\times 5$ is observed. Even with this gap, GPP scalability remains exactly the same with a $\times 3.8$ factor gain in most cases for a 4 thread parallelization. Same observation can be done for GPP1 implementation even though computation time does not depend on image size as much as GPP0 implementation. Actually, as told in section 2.3, dilation could be even more optimized so as to get a TCV implementation where kernel size does not impact on computation time [6]. That would lower dependency on image size too because in TCV computation, dilation kernel size grow with image size.

These observations are exactly the same for others views and EXZ dataset. Given that efficiency is very stable, we can assume that this algorithm is computation bound and not memory bound. Unlike GPU's, GPP behavior is very stable.

4.5. Full reconstruction analysis

Table 1 presents a performance review of a full reconstruction for 800×800 images for PMF and EXZ datasets where GPP0 and GPU0 are the initial implementation without

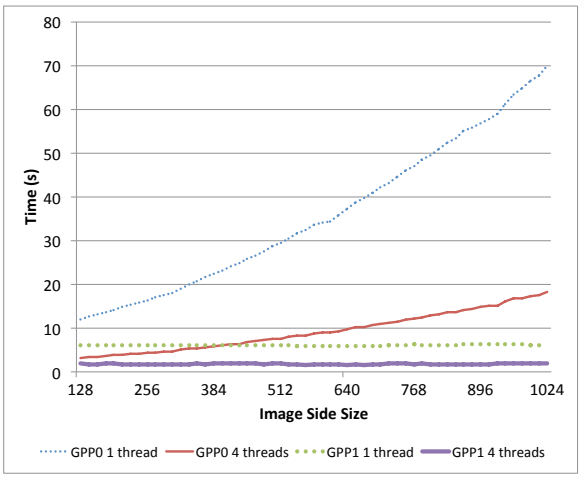


Fig. 11. GPP PMF front view reconstruction time for GPP0 and GPP1 implementations with 1 and 4 threads.

optimization and GPP1 and GPU1 are optimized implementations. Combining software optimization with GPU parallelization, True Cumulated View has been strongly improved and runs up to a factor $\times 631$ compared to initial implementation. GPU transfers are not taken into account since we consider input data to be reused for a large number of TCV runs. Image output transfer is fast enough to be ignored (about 1ms for the image size selected here).

By removing the dilation step from the classical algorithm and postponing it into a post-processing step, one can achieve a speedup of $\times 36$ on GPP, up to a $\times 50$ on C1060 and $\times 26$ on C2070. Combining that with OpenMP parallelization on GPP, a speedup of $\times 116$ can be reached on a quad core (EXZ dataset, top view computation). Note that these speedups depend on the dilation size and so, on control parameters.

Compared to the old generation of GPU, Fermi provides a speedup between $\times 1.9$ and $\times 3.8$ on GPU0 initial implementation and between $\times 2$ and $\times 7$ for GPU1 implementation. Speedups are not in favor of C2070 because L2 cache improve GPU0 implementation performance. One can notice that as all data dependent algorithm, performance may vary: the speedup for EXZ is lower than PMF because of the divergence between ray-paths. GPU1 implementation is fast, but not efficient, mostly because of random accesses.

On GPP, performance is linear and depends only on the number of points projected. Parallelized GPP1 performance is the same on both EXZ and PMF datasets with a 100k point projection per millisecond. Even though performances are a lower than on GPU, GPP are predictable and can give good performance even without SIMD parallelization.

5. CONCLUSIONS AND FUTURE WORK

This article has presented a study on the optimization and parallelization of a non-destructive evaluation ultrasound re-

	GPP0 (initial)		GPP1 (optimized)		GPU0 (initial)		GPU1 (optimized)		GPP0 vs GPU1	GPP1 vs GPU1
	1 thread	4 threads	1 thread	4 threads	C1060	C2070	C1060	C2070	slowest / fastest	fastest / fastest
View type	EXZ dataset / 800 × 800 image reconstruction									
Side View	5.02	1.32	1.16	0.32	0.92	0.34	0.14	0.072	×**69**	×**4**
Top View	43.12	11.37	1.19	0.37	10.36	1.71	0.21	0.068	×**631**	×**5**
Front View	14.07	3.67	1.22	0.36	6.19	1.86	0.24	0.077	×**180**	×**5**
	PMF dataset / 800 × 800 image reconstruction									
Side View	18.6	4.9	4.2	1.1	2.4	1.3	0.7	0.26	×**72**	×**4**
Top View	60.3	15.9	4.2	1.0	10.8	2.8	0.8	0.11	×**548**	×**10**
Front View	62.2	16.1	4.3	1.1	20.0	11.1	1.0	0.29	×**214**	×**4**

Table 1. Overall computation time in seconds for 800×800 True Cumulated View image reconstruction on EXZ and PMF datasets.

construction algorithm.

High performance have been reached more specifically with the latest generation of Nvidia GPU : Fermi. Compared to initial implementation, optimized GPP implementation runs up to $\times 116$ faster and GPU implementation up to $\times 631$. Some architectural optimizations are integrated in hardware instructions such as *atomic* intrinsics and allow the user to reap the benefits of this optimization in a transparent way. The developer avoids a lot of hardware details that has to be known when programming some types of specific parallel architectures.

Software optimizations combined to GPU parallelization allows the user to get a very fast reconstruction in human interactive time which is a major step into NDE ultrasound reconstruction.

These results are indeed very motivating and other algorithms studies are on the way. GPU are on fast growing curve and multicore GPP are evolving with a core number increasing each year and architecture evolutions (Intel Sandy Bridge or AMD Bulldozer). Their evaluation will be included into future studies. Eventually, OpenCL offers a very promising platform and its evolution has to be followed.

6. REFERENCES

[1] "CIVA : State of the art simulation software for Non Destructive Testing," `http://www-civa.cea.fr/`.

[2] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov, "Parallel computing experiences with cuda," *IEEE Micro*, vol. 28, pp. 13–27, 2008.

[3] J. Porre, S. Mahaut, S. Chatillon, and P. Calmon, "Simulation of phased array techniques and model based data reconstruction," *AIP Conference Proceedings*, vol. 760, no. 1, pp. 906–913, 2005.

[4] G.Ribay, C.Poidevin, G.Rougeron, and B.Chassignole L.de Roumilly, "UT Data Reconstruction in Anisotropic and Heterogenous Welds," *8th International Conference on NDE in Relation to Structural Integrity for Nuclear and Pressurised Components Abstracts*, 2010.

[5] Tarik Saidani, Lionel Lacassagne, Joel Falcou, Claude Tadonki, and Samir Bouaziz, "Parallelization schemes for memory optimization on the cell processor: A case study on the harris corner detector," *T. HiPEAC*, vol. 3, pp. 177–200, 2011.

[6] Marcel van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Pattern Recogn. Lett.*, vol. 13, pp. 517–512, July 1992.

[7] Luke Domanski, Pascal Vallotton, and Dadong Wang, "Parallel van Herk/Gil-Werman image morphology on GPUs using CUDA," GTC 2009.

[8] Jos B. T. M. Roerdink, "Multiresolution maximum intensity volume rendering by morphological adjunction pyramids.," *IEEE Trans Image Process*, vol. 12, no. 6, pp. 653–60, 2003.

[9] Vasily Volkov and James Demmel, "LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs," technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, May 2008.

[10] David Patterson, "The top 10 innovations in the new nvidia fermi architecture, and the top 3 next challenges," Tech. Rep., NVIDIA, 2009.

[11] "Whitepaper, NVIDIA's Next Generation CUDA Compute Architecture : Fermi," Tech. Rep., NVIDIA, 2009.

[12] Victor Podlozhnyuk, "Image convolution with cuda," technical report, NVIDIA, June 2007.

[13] Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos, "Demystifying GPU Microarchitecture through Microbenchmarking," *ISPASS*, pp. 235–246, 2010.