Performance Analysis of the Multi-pass Transformation for Complex 3d-Stencils on GPUs

S. Tabik [1], L. F. Romero [1], E. L. Zapata [1]

Abstract— **Complex iterative 3d stencils based on a series of multiple simpler stencils with different computation intensities cannot be handled properly using standard techniques on the GPU. This work demonstrates that decomposing these kind of stencils into a sequence of up to a specific number of simpler stencils and further optimizing each individual kernel provides the best overall performance. We focus on the family of PDE-based denoising methods, which can be reformulated as sequence of multiple stencils-based tasks. The performance results and analysis show that there exists an optimal level of splitting-coalescence of these stencils-based tasks that reaches the best compromise between better use of fast-memories and higher concurrency.**

Keywords— **complex stencils, decomposing-coalescing tasks, GPU accelerators, multi-pass optimization.**

I. Motivation and Related Works

THERE exists a large number of works that focuses on optimizing naive 2d and 3d stencil kernels on multi-core and many-core systems. These works can be divided into two classes. The first class focuses on applying transformations such as ghost-zone, time-tiling, or a mix of both [9], [7], [5]. While the second category focuses on building auto-tuning environments which determine the subset of optimizations that improves the performance of the stencil on each specific architecture [3].

Using ghost-zone optimization, which implies enlarging the size of the halo and performing redundant computation, reduces communications between tiles on the GPU but only for iterative 2d stencils with low computation intensity [7], [9]. Time tiling, which consists of tiling the matrix along the time dimension, enhances data locality but again only for simple memory-bound iterative 2-d stencil codes [5], [9].

All these standard optimizations are essential to improve the performance of very simple stencils but they do not require or simply are not applicable to complex stencils, such as iterative compute-intensive 3d-stencils.

On the other hand, the multi-pass optimization, which consists of decomposing the CUDA-kernel into multiple CUDA-kernels, has not been analyzed in a wider context. There exist few works in the literature that employed this optimization to improve a specific application. Indeed, none of them has analyzed at what extent it is beneficial to the performance for a whole family of algorithms. For instance, Lee et al. claimed that applying two-passes approach to a memory-bound neuroimaging algorithm increases concurrency. While, Abdelfattah et al. [1] used two-passes optimization to separate two stages with different computation patterns in the symmetric matrix-vector product code.

The main contributions of this work are: 1) Understanding when the multi-pass optimization can be beneficial to complex stencils representative of PDE-based denoising methods and 2) presenting an interesting pattern of PDE-denoising methods that substantially helps optimizing them for both many and multi-cores.

[1]Dpto. de Arquitectura de Computadores, Univ. Malaga, e-mail: stabik@uma.es

II. PDE-denoising methods: Complex Stencils

Denoising algorithms are essential tools in image processing in many bioimaging modalities. The most sophisticated and powerful methods are those that solve Partial Differential Equations (PDE). This family of algorithms, like for instance the ones described in [2], [4], [6], apply an iterative sequence of mathematical kernels, formulated as stencils with different computation intensity, where the input image is partially or entirely read and updated several times during each denoising iteration till obtaining the final filtered 3d-image after a total number of about 60 to 100 iterations.

A. A Case Study: Anisotropic Nonlinear Diffusion Algorithm

For simplicity, let's focus on a particular case study, the Anisotropic Nonlinear Diffusion (AND) algorithm to understand the features of the considered family of methods. AND-algorithm can be formulated as a stream of four stages, where the noisy image travels through these four methods of different data dependencies, data access patterns and arithmetic intensity. That is, each single iteration consists of:

- First a Gaussian smoothing is applied to the initial 3d-image. Each pixel is updated using a convolution along the X-, Y- and Z-axes, i.e., using the six nearest neighbors. The update of each pixel needs 12 flops (6 reads+1 write). Which is comparable to a 7-points stencil read from and wrote to the 3d-image as shown in Figure 1(a).
- Then, the structure tensor, ST, of the obtained 3d smoothed image is calculated. Each pixel of ST, which consists of a structure of 6 floats,

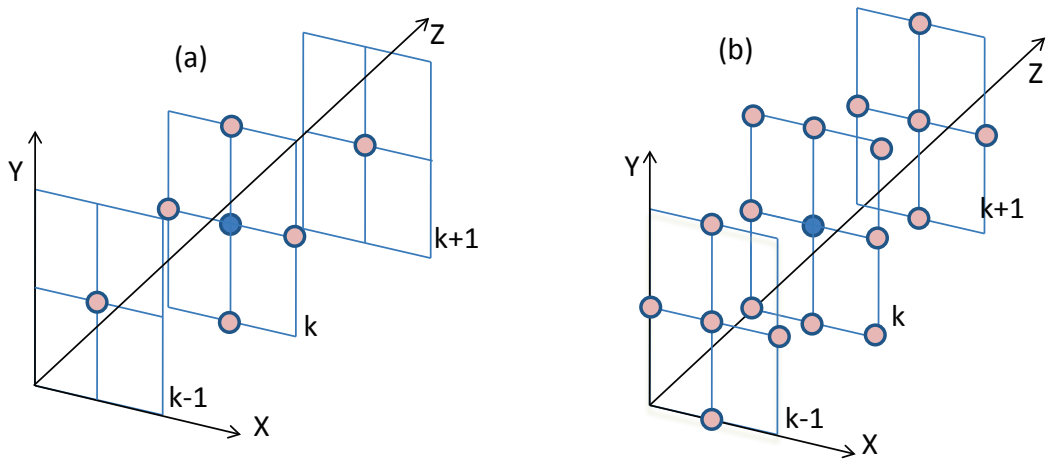

Fig. 1. (a) 7-point stencil, i.e., each pixel needs 7 nearest neighbors including itself to be updated. (b) 19-point stencil, i.e., each pixel needs 19 nearest neighbors to be updated.

is calculated using its corresponding six nearest neighbors from the smoothed image. The data access pattern is a 7-points stencil read from the 3d-image of size $N_x \cdot N_y \cdot N_z$ and wrote to ST of size $N_x \cdot N_y \cdot N_z \cdot 6$.

- Afterwards, the Diffusion Tensor, DT, is calculated. Each pixel of DT is calculated by calculating the eigenvalues and eigenvectors of the corresponding symmetric 3x3-matrix using the iterative Jacobi method. The elements of the symmetric 3x3-matrix are initialized using the values of the corresponding ST pixel. This is comparable to 1-point stencil read and wrote to ST. This stage is clearly compute intensive.
- Finally, each pixel of the 3d-image is updated using 7 neighbor points from ST and 19 neighbor points from the 3d-image. Which is a multiple 7-points and 19-points stencil that reads data from the 3d-image and ST and writes the result in the 3d-image. Updating each pixel of the 3d-image needs 26 reads, one write and 68 flops.

III. Splitting-Coalescence of Stencils on GPU: Multi-pass Approach

Commonly, parallel computing PDE-based methods on clusters of muli-processors employs the SPMD model, where each processor denoises its own 3d-block of the image during the whole iterative process. At the end of each iteration processors that work on neighbor blocks intercommunicate their halos [10].

An intuitive CUDA implementation of this family of methods can consider either encapsulating the whole complex stencil into one pass or splitting it into multiple passes. The performance of the one kernel version can be delimited by the smaller size of faster memory resources during the kernel execution while the multiple kernels version may be penalized by higher scheduling overheads and traffic between off-chip and on-ship memories, which is necessary for inter-kernels communication. However, more combinations can be also considered by merging different successive stencils, 2'-passes and 3'-passes. One of the objectives of this work is to analyze to what extent one should split or coalesce these stencils to obtain the best overall performance on GPU accelerators.

We developed six implementations of AND-method by spiting it into one-, two-, three- and four passes. Applying two- and three-passes optimization has two possible versions. Stencil computation is performed along the z-dimension using space-tiling. First, the 2D tiles and necessary halos are loaded into the shared memory, where each threadblock computes its corresponding output 2D tile, similar to the strategy employed in [8]. Here is a brief description of each implementation.

- The 1-pass approach, encapsulates the whole method, Gauss smoothing, structure tensor calculation, diffusion tensor calculation and, the PDE solution, i.e., Gauss+ST+DT+PDE, into one pass. Updating one 2D tile of the output 3d-image needs to upload to shared memory 4 tiles from the original image (i.e., the corresponding input tile + 3 halos) and 3 tiles of ST (i.e., the corresponding ST tile to be calculated + 2 ST-tile-halos). Recall that three tiles of the original 3d-image are required to compute one ST tile. The 2d-tiles with $Z = 0$ and $Z = Nz$ of the output 3D-image and ST are calculated using a mirroring operation of the 2D-plan with $Z = 2$ and $Z = Nz - 2$ respectively.
- The 2-passes implements Gauss+ST+DT in one pass and PDE in a second pass. The first pass updates each ST-tile using 1 tile of the input 3d-image; the halo pixels are uploaded into registers. The PDE pass needs one ST-tiles and 3 tiles of the input 3d-image; the halo pixels are uploaded into registers.
- The 2'-passes implements Gauss+ST in one pass and DT+PDE in a second pass. The first pass updates each ST tile using one tile of the input 3d-image; the halo pixels are uploaded into registers. The DT+PDE pass needs 3 ST-tiles and 4 tiles of the input 3d-image.

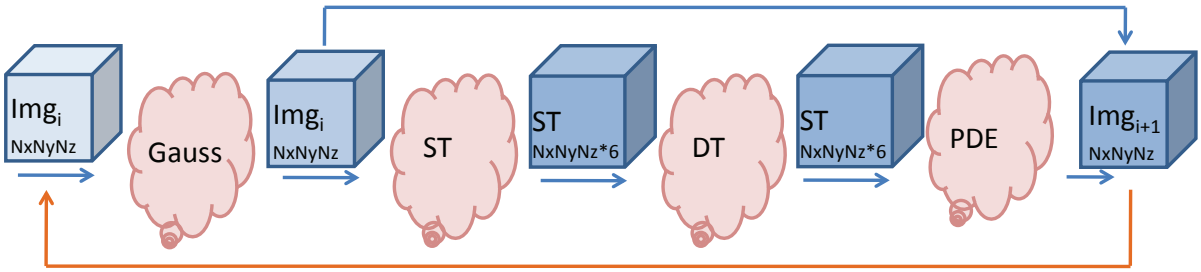

Fig. 2. Pattern of one AND-method iteration: A sequence of four stencil-based stages (in pink color). Gauss() smooths the image, ST() computes the structure tensor, DT() computes the diffusion tensor and PDE() update the image

- The 3-passes version implements Gauss+ST in a first pass, DT in a second pass and, PDE in a third pass. The first pass needs to upload into shared memory one 2D-tile of the input image and one tile of its corresponding ST tile; the halo pixels are uploaded into registers.
- The 3'-passes approach implements Gauss in a first pass, ST+DT in a second pass and, PDE in a third pass. The ST+DT pass needs one tile of 3d-image and one ST-tile.
- Finally, the 4-passes version implements each single stage into one single kernel. Gauss pass needs one 2D-tile of the input image. ST pass needs one tile of ST and one tile of the 3D-image. DT pass needs only to update one ST tile.

IV. Experimental Results and Discussion

This section provides and discusses the performance results of the AND-complex stencil when decomposed into one, two, three and four simpler stencils, each pass is encapsulated into one CUDA kernel, and its comparison with the Pthreads-implementation described in [10]. The evaluations of the CUDA-implementations, written in CUDA C v4.0, were carried out on a single Fermi C2070, with 14 multiprocessors, 448 cores, 48 Kb of shared memory, 32768 registers per block and 1.25 GB of DRAM. While the Pthreads-implementation of the code was executed on 8-core Intel X7550 Beckton using 8threads. The Pthreads-implementation is used only for comparison purposes .

Figure 3 (a) shows the speedup of the different CUDA-passes implementations (i.e., 1-, 2-, 2', 3-, 3'- and 4-passes codes) with respect to the Pthreads-implementation for three 3d-images of sizes $128 \times 128 \times 128$, $256 \times 256 \times 256$ and, $512 \times 512 \times 512$. The found optimal thread-block sizes for the CUDA-codes are 4×32, 8×32, 16×64, 16×16, 16×64 and, 16×64 respectively.

Analyzing the speedup from the 1-pass to the 4-pass versions shows that the 3-passes version, which implements Gauss+ST, DT and PDE into one pass each one, is the fastest version over all the implementations providing a speedup with respect to Pthreads-implementation of up to 290 for the smaller size and 55 for the larger image. In addition, Figure 4(a),(b),(c) shows that the 3-passes version overcomes all the passes versions for the three image sizes by up to $160\times$ and $40\times$ for the smaller and larger image respectively.

A further analysis of registers and shared memory utilization and performance profiling of each pass of the six codes are provided in Table 1 and 2 respectively. Examining the results of the 3-passes code and comparing it with the other versions shows that implementing DT into an independent kernel allows this pass reaching an optimal utilization of on-ship and off-ship memories. In particular, it has a better registers and shared memory utilization, i.e., it applies less stress on this fast memories, from 63 (which is the maximum number of registers that can be used per threads) to 42, in comparison with the pass that merges ST+DT from the 3'-passes version and the one that merges DT+PDE in the 2'-passes version. In addition, implementing DT into one pass reaches a better use of off-ship memories by showing better L1 and L2 hit rates. Moreover, in this 3-passes version DT reaches a higher concurrency, i.e., higher number of active warps per cycle from 15 to 22 when comparing 3'-passes version versus the 3-passes one from Table 2. Notice that DT represents more than 40% of the total runtime of all the implementations and therefore increasing its performance affects the overall performance.

Notice from Table 1 and 2 that merging ST with GAUSS does not affect the concurrency of ST pass. This means that the overhead produced by separating ST and GAUSS into two different passes is higher than the fact of sharing the same fast memories.

On the other hand, the performance of the PDE pass is limited by registers as can be seen from Table 1 and it provides better performance when it does not have to share fast memories.

In summary and from a programming point of view, an effective use of the multi-pass approach as we learnt from the considered case study consists of 1) implementing stencils limited by registers into one pass, 2) merging stencils that have the same data access pattern and use the same data structures into one pass and 3) implementing the most time consuming stencils with high concurrency grad into one pass because merging it with passes with different data utilization penalizes its concurrency.

TABLE I

Registers and shmem utilization per thread (estimated using nvcc 4.0) for 256×256×256 3D-image

	1-pass	2-passes	2'-passes	3-passes	3'-passes	4-passes
Reg count shmem (bytes)	63 29736	63,59 41616,32376	39,63 9080,29728	39,42,63 9080,7776,11664	26,61,63 1304,9072,11664	26,38,42,63 4632,32368,27744,41616

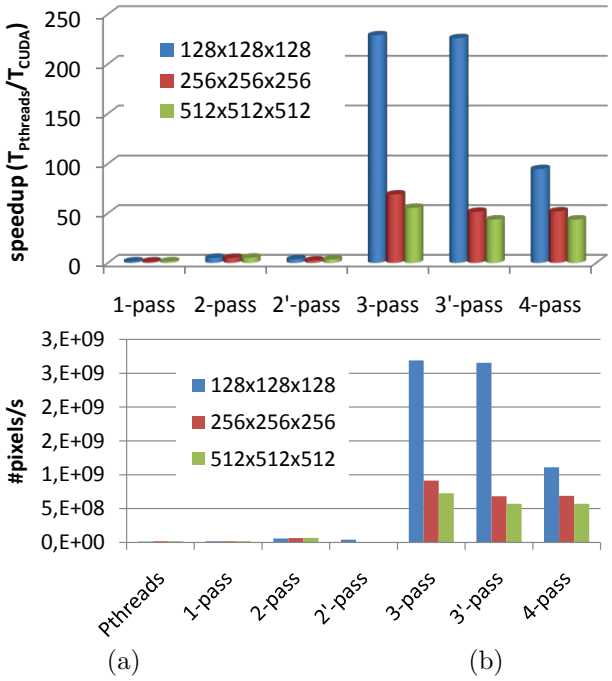

Fig. 3. (a) Speedup of the 1-, 2-, 2'- 3-, 3'- and 4- passes CUDA-versions (on Tesla C5070) with respect to the Pthreads-version (on a 8-core Intel X7550 Beckton) for three 3d-image sizes, $128 \times 128 \times 128$, $256 \times 256 \times 256$ and, $512 \times 512 \times 512$. (b) The number of processed pixels per second by the Pthreads- and CUDA-, 1-, 2-, 2'- 3-, 3'- and 4- passes versions respectively.

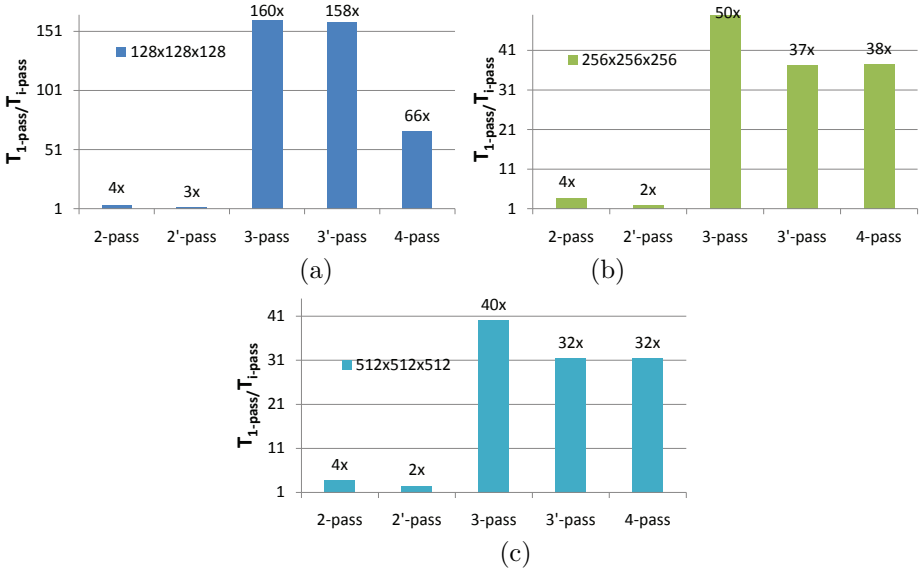

Fig. 4. (a) Speedup of 2-, 3-, 3'- and 4- passes versions with respect to the 1-pass implementation, on Tesla C5070, for three 3d-image sizes, (a) $128 \times 128 \times 128$, (b) $256 \times 256 \times 256$ and, (c)$512 \times 512 \times 512$.

V. Conclusions

This paper characterizes PDE-denoising images to make their CUDA-parallel implementation as efficient as possible with less programming efforts. We showed that 1) implementing register limited stencils into one single pass, 2) merging stencils that use the same data structure and data access into one pass and 3) implementing time consuming stencils with high concurrency into one pass improves substantially the performance of complex stencils that are initially bounded by registers and shared mem-

TABLE II

One(Gauss+ST+DT+PDE), two(Gauss+ST+DT PDE), three'(Gauss+ST, DT, PDE) and (Gauss, ST+DT, PDE) and, four-passes(Gauss, ST, DT, PDE) for a 256×256×256 3D-image and a fix thread-block 16×16

	time (%)	active warps/ active cycles	L1 gld hit rate (%)	IPC	inst/ byte	L2 glb mem read throughput (Gb/s)	L2 glb mem write throughput (Gb/s)
1-pass:							
Gauss+ST+DT+PDE	83,71	7,98	85,28	0,47	3,52	16,4	16,47
2-passess:							
Gauss+ST+DT	**44,64**	**16,02**	**46,24**	**0,84**	**1,34**	**67,69**	**84,3**
PDE	7,95	16,46	86,39	1,06	4,51	47,16	9,79
2'-passess:							
Gauss+ST	2,56	16,79	66,18	0,36	1,53	9,20	52,78
DT+PDE	**80,11**	**4,21**	**84,36**	**0,55**	**13,81**	**5,24**	**5,25**
3-passes:							
Gauss+ST	9,8	23,74	69,82	0,37	1,52	9,18	53,42
DT	**47,3**	**22,04**	**72,46**	**0,67**	**1,23**	**70,36**	**67,75**
PDE	6.08	14,89	86,42	1	4,65	45,61	9,81
3'-passes:							
Gauss	1,44	31,81	69,47	1,15	2,06	84,11	52,87
ST+DT	**43,76**	**15,92**	**40,92**	**0,83**	**1,33**	**68,17**	**84,66**
PDE	7,76	15,58	86,26	1,05	4,64	45,74	9,85
4-passes:							
Gauss	1,06	30,34	68,29	1,13	2,03	83,60	52,66
ST	9,49	23,70	67,26	0,33	1,43	8,91	51,87
DT	**44,5**	**22,59**	**72,59**	**0,67**	**1,22**	**70,54**	**67,81**
PDE	5,7	16,51	85,69	1,13	4,89	45,64	9,83

ory. We also proposed a way of systematizing the application of this optimization to diminish the programming time and effort.

References

[1] Dongarra J. Keyes D. Abdelfattah, A. and H. Ltaief. Optimizing memory-bound numerical kernels on gpu hardware accelerators, 10th international meeting on high-performance computing for computational science (vecpar 2012), riken advanced institute for computational science (aics), kobe, japan, july 17th-20th. 2012.

[2] D. Barash. Fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2002.

[3] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, 2008.

[4] J.J. Fernandez. Tomobflow: feature-preserving noise filtering for electron tomography. doi:10.1186/1471-2105-10-178. 2009.

[5] J. Holewinski, L. Pouchet, and P. P. Sadayappan. High-performance code generation for stencil computations on gpu architectures. In *Proceedings of the 26th ACM international conference on Supercomputing*, ICS '12, pages 311–320. ACM, 2012.

[6] S. Li J.J. Fernandez. Anisotropic nonlinear filtering of cellular structures in cryo-electron tomography. *Computing in Science & Engineering*, 7(5):5461, 2005.

[7] J. Meng and K. Skadron. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 256–265, New York, NY, USA, 2009. ACM.

[8] P. Micikevicius. 3d finite difference computation on gpus using cuda. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 79–84, 2009.

[9] G. Rivera and Chau-Wen Tseng. Tiling optimizations for 3d scientific computations. In *Supercomputing, ACM/IEEE 2000 Conference*, page 32, 2000.

[10] S. Tabik, E. M. Garzón, I. García, and J. J. Fernández. High performance noise reduction for biomedical multidimensional data. *Digit. Signal Process.*, 17(4):724–736, July 2007.