Evaluating the Fission/fusion Transformation of an Iterative Multiple 3D-stencil on GPUs

Siham Tabik*
Dept of Computer Architecture, University of Malaga
29071 Malaga
Spain
siham@uma.es

Alin Murarasu
Dept of Informatics, Technische Universitat Munchen
Munchen,Germany
Murarasu@in.tum.de

Luis F. Romero
Dept of Computer Architecture, University of Malaga
29071 Malaga
Spain
felipe@uma.es

ABSTRACT

This paper focuses on problems that can be expressed as iterative multiple 3D-stencils and find-out ways to optimize them for GPUs. We selected for this study a representative example, the Anisotropic Nonlinear Diffusion (AND) algorithm, which is one of the most powerful methods to denoise multidimensional data. When optimizing such algorithms for cache-based architectures, emphasis is placed on fusing all the involved stencils to increase data reutilization. However, on local-store based architectures (GPUs) the challenge is finding the optimal fission-fusion level, which implies exploring a combinatorial number of the stencils, thus making the process non-trivial. We found experimentally that the fission of 3D-stencils with high registers and shared memory pressure and fusion of 3D-stencils with high and similar concurrencies provide the best compromise reutilization, concurrency and overheads. The optimal fission-fusion combination is 1.5× faster than the case in which we fully decompose our stencil on the NVIDIA GPUs.

Keywords

3D-stencils, fission, fusion, GPUs

1. INTRODUCTION

The most powerful methods to model and manipulate images, surfaces and volumetric data sets in image processing, computer graphics and visualization are those based on solving PDE equations [15]. In general, these algorithms are often expressed as an iterative "big" multiple 3D-stencil, where each stencil has a different arithmetic intensity. As a concrete example, we study the case of the Anisotropic Non-linear Diffusion (AND) algorithm which is widely used in signal processing [15, 4]. Our goal is to develop an efficient implementation of AND for high-throughput processors such as Nvidia GPUs.

A typical optimization strategy for this kind of stencils aims at improving the locality and is based on partitioning the volume data into blocks that fit the fast on-chip memories, e.g. cache. Then, the idea is to keep each block in fast memory until it is completely denoised. Although this might seem at first a reasonable solution, in practice its performance is strongly influenced by the different computational stages (smaller stencils) of the denoising process. As the stages typically have different processing characteristics, they require updated optimizations when making the transition from one stage to another. We refer to the decomposition into stages as *kernel fission*. This is in contrast with the initial method which is more rigid, i.e. the fitting to the fast memory does not adapt to the requirements of different stages.

Nvidia GPUs are massively parallel processors that provide an advantageous ratio between GFlop/s rate and power consumption. They are characterized by a large number of cores (e.g. 16) called Streaming Multi-Processors (SM), wide SIMD units (e.g. 32 lanes), and a complex memory hierarchy including scratchpad memory (called shared memory) explicitly controllable by the programmer, a 2-level cache hierarchy, constant and texture memory, and global memory (RAM). Since they are in-order processors, GPUs hide pipeline latencies by interleaving the execution of thousands of GPU threads per core. To support a fast context switch for such a large number of concurrent threads, each SM is equipped with 32K 32-bit registers. GPUs are programmed using CUDA, a thread based programming model which offers only a subset of the synchronization options available in other thread based models for CPUs, more precisely barriers within a group of threads scheduled on one SM and atomic operations. For complex stencils, two GPU aspects are of major importance: (1) improving data reuse at register level, shared memory and L1-cache, and (2) ensuring high concurrency. The question that we try to answer is how much should we decompose a complex stencil on GPUs.

This paper brings the following two contributions to the existing work on stencil computations on GPUs:

- We express the optimization of a widely used denoising algorithm called AND as a multi-fission problem, i.e. we split the computation into four computational stages (four stencils) and we explore different combi-

*The corresponding author.

HiStencils 2014
First International Workshop on High-Performance Stencil Computations
January 21, 2014, Vienna, Austria
In conjunction with HiPEAC 2014.

http://www.exastencils.org/histencils/2014/

nations. Our approach could be applied to other applications based on complex stencils.

- We present performance results obtained on Nvidia GPUs that demonstrate the difficulty of addressing the fission problem. More precisely, the best multi-fission scheme is not obvious: The computation should be divided in 2 stages (2 stencils). This version is 1.52× faster than the four stencils version and 7× faster than the worst 2 stencils version.

The structure of the paper is the following: We present first the related works where we emphasize on the fact that although fission / fusion is an old technique applied especially to for-loops, it has not been fully studied in the scope of complex stencils on GPUs to the best of our knowledge. In Sec. 3, the mathematics behind the complex stencil of the AND algorithm is described. Our approach to fission is explained in Sec. 4. Sec. 5 describes all the GPU-implementations evaluated in this work. Sec. 6 shows our experiments and indicates the best combination of smaller stencils. Sec. 6 provides conclusions and future works.

2. BACKGROUND AND MOTIVATION

2.1 Anisotropic nonlinear diffusion (AND)

The Anisotropic nonlinear diffusion PDE-based technique is a filtering method that reduces noise and enhances local structure. It has been widely used in many fields especially in computational vision, biomedicine and differential geometry because it shows a superior performance to other filtering methods [15]. It was first introduced in visualizing electron tomograms in [5, 6]. In this work we used AND for filtering 3D-grids [4].

2.2 AND on multicore versus GPUs

Multicore systems are characterized by larger caches and a small number of cores. One thread per core can use up to 64Kb L1, 4Mb L2 and 18 Mb L3. Whereas, the GPU has a larger number of cores/SM that share a small area of shared memory (shmem), 48 Kb. Thus, a high shmem utilization per thread may penalize concurrency.

When optimizing AND for multicore architectures all emphasis is placed in splitting (or fissioning) data into blocks that fits the caches [14]. Halos are needed only in the frontiers between neighbor threads (or cores). The communication of halos between cores is performed via the shared cache levels, typically shared L2 or shared L3. Re-computation of halos, when needed, is considered only between cores from different sockets or nodes.

On the GPU, one has to consider not only a much finer splitting of data but also fission of the calculation, known as kernel fission. For the PDE-based applications that can be expressed as iterative multiple 3D-stencil, increasing the number of kernels increases accesses to global memory and decreases stress on shmem and registers. Decreasing the number of kernels increases pressure on registers and shmem since it implies a larger overhead due to copying a larger volume of halos and also re-computation in some cases. This means that the performance of this kind of problem is a trade-off between 1) on-chip memory utilization, 2)concurrency and 3)global memory accesses. Evaluating kernel fission in AND on the GPU needs exploring a large number of combinations, thus making the process non-trivial. To reduce the space of the combinations we consider only the ones that involve the dominant stencil.

2.3 Stencils

Stencils are characterized by a fixed data access pattern and number of arithmetical operations. Simple stencils, typically N-points stencils (N=7,15, ..) that perform single Jacobi iteration have been widely analyzed and tuned for multicore processors and GPUs [9, 10, 17, 12]. However, most of these studies consider only out-of-place sweeps (one grid for read and one grid for write) and do not extend their analysis to the boundary problem. When optimizing complex stencils that involve multiple simpler stencils, the boundary problem becomes an issue from a programming point of view since it requires a large volume of halos and sometimes re-computation.

Our own work is complementary to this line since it analyzes realistic applications that involve more than two 3D-stencils, more than two grids and includes boundary conditions for each stencil. Our objective is to find out the most efficient ways to optimize this kind of stencils for GPUs.

2.4 Fission / fusion optimization

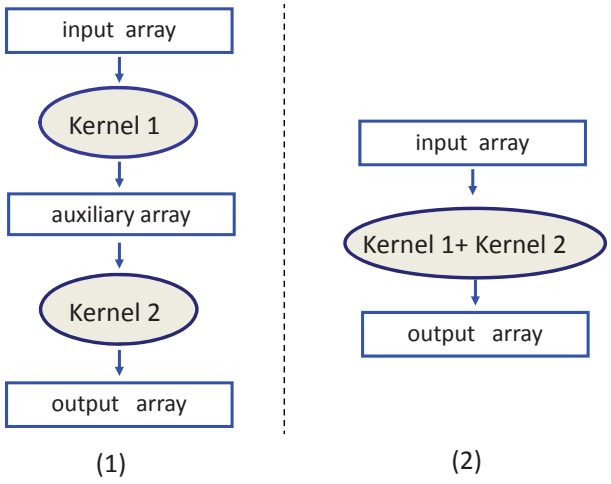

Figure 1: The type of kernel fission/fusion analyzed in this work for multiple 3D-stencils. The arrays in rectangle, array 1, intermediate array and array 2 are read and write from/to global memory in the GPU.

Kernel fission which consists of splitting one kernel into two or more kernels and its orthogonal optimization, kernel fusion, are both inspired from the well known loop fission/fusion technique. Fissioning one multiple 3D-stencil kernel implies more accesses to global memory. Figure 1 illustrates this optimization.

A large number of works show that fusion is good for performance. On multicore processors, [7, 11] showed that fusing multiple dependent kernels improves cache reuse, reduces the overheads of runtime systems and contention in work-queues. On GPUs, there is in addition several attempts to automate this optimization for the kernels of some domain specific applications. For example, the authors of [16] state

that using fusion and fission together in warehousing applications reaches a significant improvement of performance especially on large volumes of data. Our work is similar only in philosophy with the existing fission / fussion optimizations. To the best of our knowledge, the technique fission/fusion has not been covered for real-world iterative multiple 3D-stencils on GPUs.

Implementing this technique is not straightforward as we are required to first identify the stencils implied in the initial AND-kernel, analyze dependencies between them and subsequently decide how to split. The tradeoffs that characterize GPUs, e.g. between locality and concurrency, complicate the fission decision. Therefore, multiple fission variants are implemented and later compared based on their respective performance numbers measured on GPUs. We made decisions based on the weight of the smaller stencils to limit the evaluation space to the most interesting variants.

3. THE MATHEMATICAL BASIS OF AND

The most sophisticated and powerful denoising algorithms are those that solve Partial Differential Equations (PDE) [1, 2, 8, 4]. In this work we focus on a representative one of these methods, the Anisotropic Nonlinear Diffusion. AND accomplishes a sophisticated edge-preserving denoising that takes into account the structures at local scales. AND tunes the strength of the smoothing along different directions based on the local structure estimated at every point of the multidimensional image. This section presents local structure determination via structure tensors, the concept of diffusion, a diffusion approach commonly used in image processing and, finally, details of the numerical implementation.

3.1 Estimation of structure tensor

The *structure tensor* is the mathematical tool that allows us to estimate the local structure in a multidimensional image. Let $I(\mathbf{x})$ denote a 3D image, where $\mathbf{x} = (x, y, z)$ is the coordinate vector. The structure tensor of I is a symmetric positive semi-definite matrix given by:

$$\mathbf{J}(\nabla I) = \nabla I \cdot \nabla I^T = \begin{bmatrix} I_x^2 & I_x I_y & I_x I_z \\ I_x I_y & I_y^2 & I_y I_z \\ I_x I_z & I_y I_z & I_z^2 \end{bmatrix} \quad (1)$$

where $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_z = \frac{\partial I}{\partial z}$ are the derivatives of the image with respect to x, y and z, respectively.

The eigen-analysis of the structure tensor allows determination of the local structural features in the image [15]:

$$\mathbf{J}(\nabla I) = [\mathbf{v_1}\ \mathbf{v_2}\ \mathbf{v_3}] \cdot \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix} \cdot [\mathbf{v_1}\ \mathbf{v_2}\ \mathbf{v_3}]^T \quad (2)$$

The orthogonal eigenvectors $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$ provide the preferred local orientations, and the corresponding eigenvalues μ_1, μ_2, μ_3 provide the average contrast along these directions.

3.2 Diffusion in image processing

Diffusion is a physical process that equilibrates concentration differences as a function of time, without creating or destroying mass. In image processing, density values play the role of concentration. This observation is expressed by the *diffusion equation* [15]:

$$I_t = \mathrm{div}(\mathbf{D} \cdot \nabla I) \quad (3)$$

where $I_t = \frac{\partial I}{\partial t}$ denotes the derivative of the image I with respect to the time t, ∇I is the gradient vector, $\mathbf{D}$ is a square matrix called *diffusion tensor* and div is the *divergence* operator:

$$\mathrm{div}(\mathbf{f}) = \frac{\partial f_x}{\partial x} + \frac{\partial f_y}{\partial y} + \frac{\partial f_z}{\partial z}$$

In AND the smoothing depends on both the strength of the gradient and its direction measured at a local scale. The diffusion tensor $\mathbf{D}$ is therefore defined as a function of the structure tensor J:

$$\mathbf{D} = [\mathbf{v_1}\ \mathbf{v_2}\ \mathbf{v_3}] \cdot \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot [\mathbf{v_1}\ \mathbf{v_2}\ \mathbf{v_3}]^T \quad (4)$$

where $\mathbf{v_i}$ denotes the eigenvectors of the structure tensor. The values of the eigenvalues λ_i define the strength of the smoothing along the direction of the corresponding eigenvector $\mathbf{v_i}$. The values of λ_i rank from 0 (no smoothing) to 1 (strong smoothing). Therefore, this approach allows smoothing to take place anisotropically according to the eigenvectors determined from the local structure of the image. Consequently, AND allows smoothing on the edges: Smoothing runs along the edges so that they are not only preserved but smoothed. AND has turned out, by far, the most effective denoising method by its capabilities for structure preservation and feature enhancement [15, 5, 3, 4].

3.3 Gaussian smoothing

One of the most common ways of setting up the diffusion tensor $\mathbf{D}$ gives rise to the so-called Edge Enhancing Diffusion (EED) approach [15]. The primary effects of EED are edge preservation and enhancement. Here strong smoothing is applied along the preferred directions of the local structure, (the second and third eigenvectors, $\mathbf{v_2}$ and $\mathbf{v_3}$). The strength of the smoothing along the normal of the structure, i.e. the eigenvector $\mathbf{v_1}$, depends on the gradient: the higher the value is, the lower the smoothing strength is. Consequently, λ_i are then set up as:

$$\begin{cases} \lambda_1 = g(|\nabla I|) \\ \lambda_2 = 1 \\ \lambda_3 = 1 \end{cases} \quad (5)$$

with g being a monotonically decreasing function, such as $g(x) = 1/\sqrt{(1 + x^2/K^2)}$, where $K > 0$ acts as a contrast parameter [15]; Structures with $|\nabla I| > K$ are regarded as edges, otherwise they are considered to belong to the interior of a region. Therefore, smoothing along edges is preferred over smoothing across them, hence edges are preserved and enhanced.

3.4 Solving PDE

The diffusion equation, Eq. (3), can be numerically solved using finite differences. The term $I_t = \frac{\partial I}{\partial t}$ can be replaced by an Euler forward difference approximation. The resulting explicit scheme allows calculation of subsequent versions of the image iteratively:

$$\begin{aligned} &I^{s+1} = I^s + \\ &\tau \cdot (\tfrac{\partial}{\partial x}(D_{11} I_x) + \tfrac{\partial}{\partial x}(D_{12} I_y) + \tfrac{\partial}{\partial x}(D_{13} I_z) + \\ &\tfrac{\partial}{\partial y}(D_{21} I_x) + \tfrac{\partial}{\partial y}(D_{22} I_y) + \tfrac{\partial}{\partial y}(D_{23} I_z) + \\ &\tfrac{\partial}{\partial z}(D_{31} I_x) + \tfrac{\partial}{\partial z}(D_{32} I_y) + \tfrac{\partial}{\partial z}(D_{33} I_z)) \end{aligned} \quad (6)$$

where s is the iteration index, τ denotes the time step size, I^s denotes the image at time $t_s = s\tau$, the terms I_x, I_y,

I_z are the derivatives of the image I^s with respect to x, y and z, respectively. Finally, the D_{mn} terms represent the components of the diffusion tensor $\mathbf{D}^s$. The standard scheme to approximate the spatial derivatives ($\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$ and $\frac{\partial}{\partial z}$) is based on central differences.

In this traditional explicit scheme for solving the partial differential equation Eq. (3), the stability is an issue [15]. The maximum time step used in the experiments carried out in this work is $\tau = 0.1$. Typically the total number of iterations ranges from 60 to 100 in 3D problems [15, 3, 4] with that value of τ.

4. AND:AN ITERATIVE MULTIPLE 3D STENCIL

The algorithm for solving the PDE in Eq. (6) using the discretization of the temporal and spatial derivatives described above can be expressed as an iterative multiple four 3D-stencils as simplified in Figure 2. The first stencil is a 7-point gauss computation, the second is a 7-point stencil that computes the structure tensor, a one-point stencil that solves an 3×3 eigenvalue system and the last stencil is a multiple stencil that solves the PDE equation.

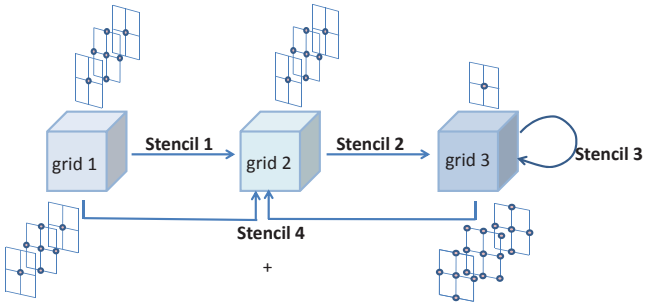

Figure 2: The four 3D-stencils involved in AND and their respective data dependence.

An efficient parallel implementation of AND for single and multicore processors consists of dividing the volume into subvolumes that fit entirely in the cache hierarchy and assign for denoising each subvolume to one core. This implementation has shown to have a high cache reutilization and low memory footprint as demonstrated in [14, 13].

AND can be seen as an iterative complex stencil where each iteration can be expressed using four 3D-stencils that operate on three 3D-arrays as plotted in Figure 2. The 3D-grids, grid 1 and grid 2, of dimension $N_x * N_y * N_z$ represent the initial noisy volume and the final denoised volume after one iteration. For the next iteration, the content of grid 2 becomes the initial volume to be denoised. grid 3 of dimension $6 \times N_x * N_y * N_z$ is used to store the structure tensor of grid 2. Table 1 sumarizes the characteristics of each one of the four stencils.

From here on stencil 1, 2, 3 and 4 will be labeled as $GAUSS$, ST, DT and PDE respectively.

- Stencil 1 ($GAUSS$), is a 7-points gauss smoothing stencil, it reads the input noisy 3D-grid of size $N_x * N_y * N_z$ and generates a different 3D-grid of the same size.

Table 1: A simplified summary of the stencils involved in AND.

Stencils	Flops/ stencil	Reads/ stencil	writes/ stencil	# of in and output grids
1	8	7	1	1 in 1 out
2	12	6	1	1 in 1 out
3	27	6	1	1 for in and out
4	69	51	1	2 in 1 out

- Stencil 2 (ST), is a 6-points stencil that calculates the structure tensor of size $6 * N_x * N_y * N_z$ from the 3D-grid obtained in stencil 1.
- Stencil 3 (DT), is a in-place 1-point stencil, where each point is in fact a 3×3 symmetric matrix thus only 6 elements are stored. This stencil uses the jacobi iterative method to solve an eigen-value eigen-vector problem of the 3×3-matrix at each point of the 3D-grid.
- Stencil 4 (PDE), is a multiple, 7-points and 16-points stencils. It reads both the input 3D-grid and the diffusion tensor obtained from stencil 3 to solve the Partial Differential Equation of the discritized problem.

Fusing all these stencils together decreases the accesses to main memory and also the used space in the main memory. Theoretically, this implies an increase of the theoretical peak performance of AND but it omits factors such as recomputation of the borders which may penalize concurrency and also the synchronization needed to update the halos and borders of each stencil.

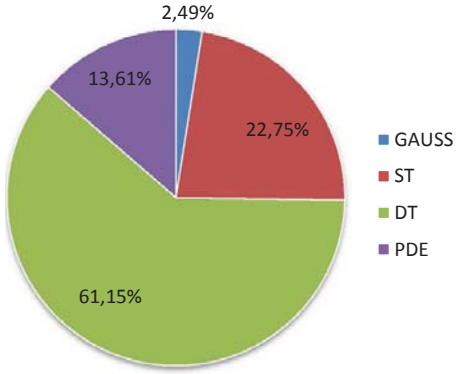

Figure 3: The weight of each stencil of AND in the total runtime.

To guide our analysis we first measured the weight of each stencil in the total runtime as plotted in Figure 3. The runtime of each stencil includes its corresponding border conditions problem. As it can be seen from this Figure, DT is the heaviest stencil, therefore, we implemented and later evaluated all the fission/fusion combinations around DT, namely, $G|ST+DT+PDE$, $G+ST+DT|PDE$, $G+ST|DT+PDE$, $G|ST+DT|PDE$, $G+ST|DT|PDE$ and $G|ST|DT|PDE$. The symbols | and + refers to fission and fusion respectively.

5. GPU IMPLEMENTATIONS

The kernels, or stencils, of all the evaluated combinations follow the same scheme. The 3D-grids are processed by tiles of dimension $DIMX \times DIMY$ along the z-axis. First, data and halos are read from global memory and copied into a tile in shared memory, then the corresponding computation is carried out and finally the output 3D-grids are updated. The main differences between each implementation are the number of accesses to main memory, the used space in main memory, the used number of registers and space of shmem.

Table 2: Global memory accesses for read and update in/out 3D-grids

implementation	global memory accesses per implementation per kernel
$G\|ST+DT+PDE$	g1 g2 | g1 g2
$G+ST+DT\|PDE$	g1 g3 | g1 g3 g2
$G+ST\|DT+PDE$	g1 g3 | g1 g3 g2
$G\|ST+DT\|PDE$	g1 g2 | g2 g3 | g1 g3 g2
$G+ST\|DT\|PDE$	g1 g3 | g3 g3| g1 g3 g2
$G\|ST\|DT\|PDE$	g1 g2 | g2 g3 | g3 g3 | g1 g3 g2

g1, g2, g3 refers to the input 3D-grid (labeled as grid 1 in Figure 2), output 3D-grid (labeled as grid 2 in Figure 2) and the structure tensor(labeled as grid 3 in Figure 2) used in the implementations. tile1, tile2 and tile3 are the auxiliary arrays used in shmem to store tiles from grid 1, grid 2 and grid 3 respectively.

Table 3: Shared memory usage per implementation per kernel

implementation	shmem usage per implementation per kernel
$G\|ST+DT+PDE$	tile1 | 4*tile1 12*tile3
$G+ST+DT\|PDE$	4*tile1 3*tile2 | 3*tile1 6*tile3
$G+ST\|DT+PDE$	4*tile1 3*tile2 | 3*tile1 6*tile3
$G\|ST+DT\|PDE$	tile1 | tile2 | 3tile1 6*tile3
$G+ST\|DT\|PDE$	4*tile1 | | 3*tile1 6*tile3
$G\|ST\|DT\|PDE$	tile1 | tile2 | | 3*tile1 6*tile3

6. EXPERIMENTAL ANALYSIS OF STENCILS FISSION/FUSION IN AND

6.1 Additional optimizations

A performance tuning was used before measuring the impact of fission/fusion transformation on the performance of AND. In particular, register blocking with an appropriate loop unrolling was used in the kernels when it is feasible and when it improves the performance. No padding is required since all the used volume sizes are multiple of 128. Experimentally, we found that synchronizing the threads at one block level before and after solving the eigen-value eigenvector problem in stencil 3 improves drastically its performance since it conducts to coalescing accesses to memory. We also used the mathematical functions implemented by CUDA, (e.g., fabsf and sqrtf) by using the compilation option -use_fast_math

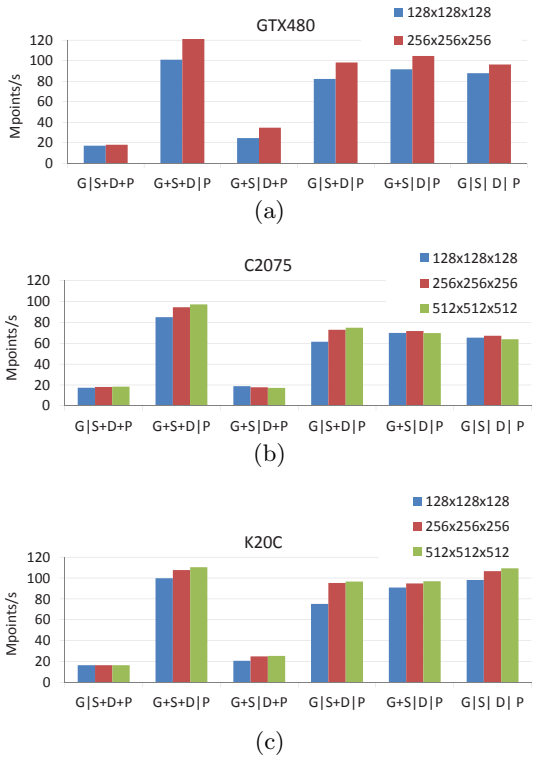

Figure 4: The throughput in Mpoints/s of all the fussion/fusion combinations, $G|ST+DT+PDE$, $G+ST+DT|PDE$, $G+ST|DT+PDE$, $G|ST+DT|PDE$ and $G|ST|DT|PDE$ on GTX489 , C2075 and K20C.

6.2 Fission/fusion transformation impact

This section provides and analyzes the performance evaluations of the 6 combinations, $G|ST+DT+PDE$, $G+ST+DT|PDE$, $G+ST|DT+PDE$, $G|ST+DT|PDE$, $G+ST|DT|PDE$ and $G|ST|DT|PDE$. Recall that we consider all the combinations except the full fused version $G+ST+DT+PDE$. In fact, the experimental results shown in this section confirm that fusing the three heaviest stencils together leads to a very poor performance and thus fusing GAUSS with this version will leads to similar performance. The experimental setup is summarized in Table 4.

Table 4: Brief summary of the GPUs, configuration and compilers used in the experiments. *The bandwidth Nvidia benchmark was used.

	C2070	K20C	GTX 480
Peak GFlops (single precision)	1288	3520	1816
Bandwidth* (GB/s)	84.8	143.9	117.7
# SMs	14	13	8
# Cores/SM	32	192	32
Clock Speed (GHz)	1.15	0.71	1.50
shmem/block (Kb)	48	48	48
# Reg/block	32768	65536	32768
Global memory (Gb)	5.24	4.68	1.45
ECC enabled	Yes	Yes	Yes
Used CUDA Compiler	4.0	5.0	4.0

All the CUDA-implementations are written in CUDA C and deal with single precision float datatype.

Figure 4(a), (b) and (c) show the throughput expressed in Mpoints/s, calculated as $N_x * N_y * N_z/time(s)$, of the combinations $G|ST + DT + PDE$, $G + ST + DT|PDE$, $G + ST|DT + PDE$, $G|ST + DT|PDE$, $G + ST|DT|PDE$ and $G|ST|DT|PDE$ on the three GPUs described in Table 1, and using three 3D-arrays of sizes $128 \times 128 \times 128$, $256 \times 256 \times 256$ and, $512 \times 512 \times 512$. The used thread blocks are of size 16×16. From the figures, one can observe that the combination $G + ST + DT|PDE$ provides the best performance over all the combinations and on all the GPUs especially on GTX and C2075. $G + ST + DT|PDE$ is up to $6\times$ better than $G|ST + DT + PDE$, and up to $1.52\times$ better than the fully fission version $G|ST|DT|PDE$.

In particular, $G + ST + DT|PDE$ and $G|ST|DT|PDE$ are both versions that provide near and sometimes competitive performance. $G + ST + DT|PDE$ is $1.3\times$, $1.4\times$ and $1.52\times$ faster than $G|ST|DT|PDE$ for the 3D-grids of sizes $128 \times 128 \times 128$, $256 \times 256 \times 256$, and $512 \times 512 \times 512$ respectively on C2070. $G + ST + DT|PDE$ is $1.14\times$ and $1.28\times$ faster than $G|ST|DT|PDE$ for the 3D-grids of sizes $128\times128\times128$, $256\times256\times256$ respectively on GTX 480. The implementation $G|ST|DT|PDE$ could not be evaluated on GTX 480 for the volume size equals $512\times512\times512$ because of memory space limitations. On Kepler, the implementations $G+ST+DT|PDE$ and $G|ST|DT|PDE$ offer very competitive throughput for the evaluated volumes. This fact can be explained by the large bandwidth and high number of registers in K20C.

To further understand these differences we measured the overhead due to updating halos and borders and also to the necessary re-computation of halos in versions $G|ST + DT + PDE$ and $G + ST + DT|PDE$. Figure 5 shows this overheads and their weights versus the weight of real useful work. Although, the combination $G|ST + DT + PDE$ has a large data reutilization and few accesses to global memory (only to read and write input and output volume), its performance is strongly penalized by the large overhead which means a lower occupancy. Whereas, version $G + ST + DT|PDE$ presents a good balance between useful work and overhead. Figure 5 represent the overhead and useful work for the volume of size $256 \times 256 \times 256$ on C2070. The plots of the rest of the volumes on all the GPUs are very similar.

In addition, we carried out a profiling of DT and PDE stencils as the combination of this two stage is the reason behind the poor performance of the implementations $G|ST+DT+PDE$ and $G+ST|DT+PDE$. We found that PDE is register bound and shows a very low active warps per active cycle, of the order of 15, while DT shows a much better concurrency level, around 25 active warps per active cycle for volume size equals $256 \times 256 \times 256$. As result, combining a stencil with poor concurrency with a stencil with higher concurrency penalizes the performance of the last one.

In summary, applying the multi-fission transformation by implementing 3D-stencils with a high registers and shemm pressure into one kernel and fusing kernels with similar concurrency provides the best compromise reutilization, concurrency and overheads.

7. CONCLUSIONS

This paper evaluates the impact of a multi-fission transformation on the performance of an iterative multiple 3D-stencil on Nvidia GPUs. In particular, we analyzed the fission / fusion transformation of AND, a representative method of PDE-based algorithms used in signal processing. Our results validate that fission improves the performance of our application on the GPU. A remarkable result is that the fission of 3D-stencils with lower concurrency and fusion of 3D-stencils with high and similar concurrencies provide the best trade-off between locality and concurrency on GPUs, especially on Fermi. For the case of AND, the two-kernels combination that fuses the three single 3D-stencils into one kernel and the multiple stencil into one kernel is $1.52\times$ faster than full fission (four kernels).

We have identified two main directions for future work: (1) indicating the best decomposition of complex stencils at design time, and (2) including our fission approach in a more general-purpose framework for tuning stencil codes on GPUs.

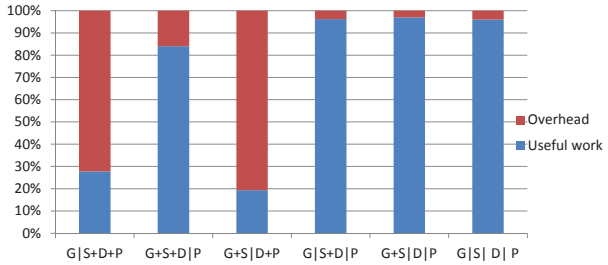

Figure 5: The overhead due to synchronization and memory accesses necessary to copy halos and borders for the six combinations $G|ST + DT + PDE$, $G+ST+DT|PDE$, $G+ST|DT+PDE$, $G|ST+DT|PDE$, $G + ST|DT|PDE$ and $G|ST|DT|PDE$ on GTX480, C2075 and K20C. The overheads of combinations $G|ST+DT+PDE$ and $G+ST+DT|PDE$ include also re-computation of some halos.

8. REFERENCES

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

[2] J.-J. Fernandez. Tomobflow: feature-preserving noise filtering for electron tomography. *BMC bioinformatics*, 10(1):178, 2009.

[3] J.-J. Fernández and S. Li. An improved algorithm for anisotropic nonlinear diffusion for denoising cryo-tomograms. *Journal of structural biology*, 144(1):152–161, 2003.

[4] J.-J. Fernandez and L. Sam. Anisotropic nonlinear filtering of cellular structures in cryoelectron tomography. *Computing in science & engineering*, 7(5):54–61, 2005.

[5] A. S. Frangakis and R. Hegerl. Noise reduction in electron tomographic reconstructions using nonlinear anisotropic diffusion. *Journal of structural biology*, 135(3):239–250, 2001.

[6] A. S. Frangakis, A. Stoschek, and R. Hegerl. Wavelet transform filtering and nonlinear anisotropic diffusion assessed for signal reconstruction performance on

multidimensional biomedical data. *Biomedical Engineering, IEEE Transactions on*, 48(2):213–222, 2001.

[7] A. Haidar, H. Ltaief, P. Luszczek, and J. Dongarra. A comprehensive study of task coalescing for selecting parallelism granularity in a two-stage bidiagonal reduction. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 25–35, May.

[8] J. M. J. Fehrenbach. Small non-negative stencils for anisotropic diffusion. *arXiv:1301.3925, Numerical Analysis*, 2013.

[9] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams. An auto-tuning framework for parallel multicore stencil computations. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

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

[11] A. G. Navarro, R. Asenjo, S. Tabik, and C. Cascaval. Analytical modeling of pipeline parallelism. In *PACT*, pages 281–290, 2009.

[12] M. Rumpf and R. Strzodka. *Nonlinear diffusion in graphics hardware*. Springer, 2001.

[13] S. Tabik, E. M. Garzón, I. García, and J.-J. Fernández. Implementation of anisotropic nonlinear diffusion for filtering 3d images in structural biology on smp clusters. In *Proc. Int. Conf. Parallel Computing: Current & Future Issues of High-End Computing, ParCo*, volume 33, pages 727–734, 2005.

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

[15] J. Weickert. *Anisotropic diffusion in image processing*. Teubner Stuttgart, 1998.

[16] H. Wu, G. Diamos, J. Wang, S. Cadambi, S. Yalamanchili, and S. Chakradhar. Optimizing data warehousing applications for gpus using kernel fusion/fission. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 2433–2442. IEEE, 2012.

[17] Y. Zhao. Lattice boltzmann based pde solver on the gpu. *The Visual Computer*, 24(5):323–333, 2008.