Microprocessors and Microsystems xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Correlation ratio based volume image registration on GPUs

Ang Li [a,b,*], Akash Kumar [b], Yajun Ha [c], Henk Corporaal [a]

[a] Eindhoven University of Technology, Eindhoven, The Netherlands
[b] National University of Singapore, Singapore
[c] Institute for Infocomm Research, Singapore

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Image registration
Histogram
GPU
Correlation ratio
Conflict-free

ABSTRACT

Volume image registration remains one of the best candidates for Graphics Processing Unit (GPU) acceleration because of its enormous computation time and plentiful data-level parallelism. However, an efficient GPU implementation for image registration is still challenging due to the heavy utilization of expensive atomic operations for similarity calculations. In this paper, we first propose five GPU-friendly Correlation Ratio (CR) based methods to accelerate the process of image registration. Compared to widely used Mutual Information (MI) based methods, the CR-based approaches require less resource for shadow histograms, a faster storage, such as the on-chip scratchpad memory, therefore can be fully exploited to achieve better performance. Second, we make design space exploration of the CR-based methods, and study the trade-off of introducing shadow histograms on different storage (shared memory, global memory) by computation units of different granularity (thread, warp, thread block). Third, we exhaustively test the proposed designs on GPUs of different generations (Fermi, Kepler and Maxwell) so that performance variations due to hardware migration are addressed. Finally, we evaluate the performance impact corresponding to the tuning of concurrency, algorithm settings as well as overheads incurred by preprocessing, smoothing and workload unbalancing. We highlight our last CR approach which completely avoids updating conflicts of histogram calculation, leading to substantial performance improvements (up to 55× speedup over naive CPU implementation). It reduces the registration time from 145 s to 2.6 s for two typical 256 × 256 × 160 volume images on a Kepler GPU.

1. Introduction

Volume image registration (VIR), the process of generating a transformation that maximizes the similarity between two volume images [1] (see Fig. 1), is one of the fundamental components frequently encountered in many medical image processing applications [2]. Among various medical registration frameworks, FMRIB's Linear Image Registration Tool (FLIRT) [3,4] is reported to be effective and robust [5]. Several similarity functions are exploited in FLIRT, the default one, however, is Correlation Ratio (CR) [6]. Based on information theory, CR exhibits comparative robustness and stability as the Mutual Information (MI) methods [4,7]. It is also reported that CR is more accurate and easier to compute than MI [4], which is confirmed by this paper as well.

VIR traditionally requires enormous computation time (e.g. registering two 256 × 256 × 160 images spends 145 s). The calculation of the similarity function, however, is the most dominant component which takes over 98% of the registration time. Meanwhile, the similarity function is inherently data parallel [8] as voxels of the volume images can be processed independently. Therefore, ever since Nvidia published Compute Unified Device Architecture (CUDA) [9], people are seeking to accelerate VIR as well as the similarity function calculations via GPU. However, an efficient GPU implementation for VIR is still challenging due to heavy utilization of expensive atomic operations for similarity calculations, which frequently turn into a performance bottleneck [10]. Although several approaches are proposed [10–14], most of them are specifically targeted for MI and still fail to resolve the bottleneck very effectively.

In this paper, we show that, compared to MI, the CR-based similarity functions are more suitable for the GPU platform. We thus explore the design space of CR and propose five CR-based similarity function implementations. The FLIRT registration framework is implemented to embed these similarity functions to construct a complete registration procedure. We show the trade-off between benefits and overheads of mapping local sub-histograms (or shadow histograms) to different storage (shared memory, global

* Corresponding author is currently at: Eindhoven University of Technology, Eindhoven, The Netherlands.

E-mail addresses: ang.li@tue.nl (A. Li), akash@nus.edu.sg (A. Kumar), ha-y@i2r.a-star.edu.sg (Y. Ha), h.corporaal@tue.nl (H. Corporaal).

http://dx.doi.org/10.1016/j.micpro.2015.04.002
0141-9331/

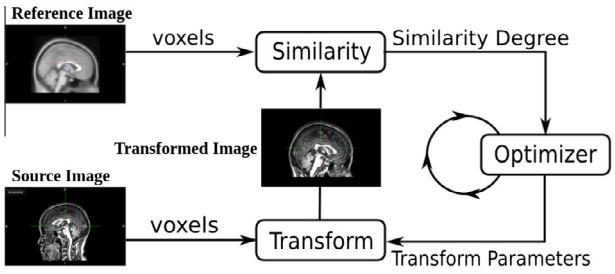


Fig. 1. Image registration. In the example, the source image is a raw MRI image while the reference image is a template. The registration framework measures the similarity between the transformed image and the reference image and tunes the transform matrices accordingly based on the searching strategies. After registration, the raw image is supposed to be aligned with the template when applying the obtained transform.

memory) by execution units of different granularity (thread, warp, thread block). The proposed designs are exhaustively tested on GPUs of different generations (Fermi, Kepler and Maxwell) so that performance variations due to hardware migrations are addressed. Further, the performance impact corresponding to the tuning of concurrency, algorithm settings (such as the number of bins) as well as overheads induced by preprocessing, smoothing and workload unbalancing are also evaluated. It is highlighted that, in the last proposed scheme, the updating conflicts of histogram calculation are completely avoided, leading to substantial performance improvements. Our best scheme achieves over 55× speedup compared to the original FLIRT version on CPU, which reduces the registration time from 145 s to 2.6 s for typical $256 \times 256 \times 160$ 3D images on a Kepler platform. Hence, the contributions of this paper are:

- Five CR based registration implement schemes for GPU. To the best of our knowledge, this is the first time the CR method is reported to be employed for image registration on GPUs. Experimental results show that CR outperforms MI, both on speed and accuracy.
- A novel design that completely eliminates the updating conflicts. This highlights the significant advantage of CR over MI on the GPU platforms.
- The trade-off between benefits of exploiting shadow histograms and its concomitant overhead based on comparisons among different schemes.
- An exhaustive and detailed evaluation of the schemes for different generations of GPUs. In this way, we address the stability and portability of the proposed designs while acquiring more details about the hardware capabilities.

The rest of the paper is organized as follows. Section 2 introduces the background of image registration, FLIRT framework and histogram calculation. Section 3 presents the proposed schemes to implement the CR similarity function. Section 4 validates these schemes on hardware. Section 5 discusses the related performance considerations. Section 6 reviews related works. Finally, Section 7 draws the conclusion.

2. Background

In this section, we first briefly describe the meaning of image registration, the process of FLIRT framework and the definition of Correlation Ratio. We then present histogram calculation and explain why conflicts exist.

2.1. Image registration

Image registration is the process of determining a transformation that maps points from one image (source image) to their homologous points in another image (reference image). It is generally formalized as a cost optimization problem. Its cost function measures the similarity degree between two images. Therefore, the optimization process is attributed as the search for a transform that minimizes the cost function (i.e. maximizes similarity):

$$\begin{aligned} &\textit{Calculate } \textbf{\textit{Transform}} \\ &\textit{such that } \textbf{\textit{similarity}}(\boldsymbol{A},\ \boldsymbol{B}) \textbf{ \textit{is maximized}} \\ &\textit{where } A = \textit{reference\_image}, \\ &B = \textit{Transform}(\textit{source\_image}) \end{aligned}$$

Fig. 1 illustrates the process of image registration. The *Transformed Image* is produced by applying the transform function on the *Source Image*. The similarity between the *Transformed Image* and the *Reference Image* is then calculated, which is returned to the optimizer. Based on the similarity, the optimizer iteratively tunes the transform function until finally the *Transformed Image* and the *Reference Image* show the best similarity.

In order to tune the transform function, we need to parameterize it. In this paper, *affine registration* is considered, so the transform is *affine transform*, which can be expressed as:

$$\textit{transformed\_image} = M \times \textit{source\_image} + \vec{b}$$

where M is a 3×3 matrix; $\vec{b}$ is a vector. The 3×4 matrix $[M\,b]$ is labeled as a *transform matrix* that uniquely defines a transform function. Therefore, the transform parameter shown in Fig. 1 is in fact a transform matrix.

During the search process various searching strategies are employed to enhance the possibility of obtaining an optimal transform, while reducing search time. These strategies comprise a searching framework.

2.2. FLIRT framework

FLIRT algorithm [3,4] is one of such searching frameworks. It is composed of four stages – each stage focuses on a specific resolution, from 8 mm, 4 mm, 2 mm to 1 mm progressively. A stage contains a series of local searches in which four spaces are traversed: *rotation, translation, scale* and *skew*. Each space is three dimensional (X, Y, Z), so if one dimension is represented by one degree of freedom (DOF), at maximum a 12-DOF search can be performed.

The primary 8 mm searching stage first executes a rotation space searching with a stride of 60 degrees, thus $6 \times 6 \times 6$ times to cover the whole space (360 degrees for all three dimensions). For each checkpoint, a 4-DOF (i.e. rotation and global scale) local search is done. Then another rotation space search with a finer stride of 18 degrees is executed. This time, $(360/18)^3 = 8000$ trials are required. However, unlike the coarse grain search, for every checkpoint, we only evaluate that specific spot instead of initiating a complete local search. Afterwards, three transformation matrices that generate the minimum cost are selected to execute a 7-DOF (i.e. rotation, translation and global scale) full search. The obtained matrices are marked as candidates for the next stage.

In the second 4 mm stage with 4 mm resolution, a 7-DOF (i.e. rotation, translation and global scale) search is applied to the three candidates together with their 30 neighbors (for each candidate, two perturbations on each rotation dimension with 9 degree deviation, four perturbations on scaling with zoom in and zoom out by a factor of 0.1 and 0.2). The best transformation is found out as input for the next stage.

In the 2 mm stage, a 7-DOF (i.e. rotation, translation and global scale), 9-DOF (i.e. rotation, translation and scale) and 12-DOF (i.e. rotation, translation, scale and skew) local search are performed alternately, further approaching the global optimal.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

Finally, in the 1 mm stage, the expected global optimal is obtained after going through a complete 12-DOF (i.e. rotation, translation, scale and skew) local search. This transform matrix generates the maximum value for the similarity function.

2.3. Correlation ratio

The Correlation Ratio (CR) [6] of two variables X and Y is a measurement of functional dependence between them, defined as:

$$\eta(Y|X) = \frac{Var[E(Y|X)]}{Var(Y)} = 1 - \frac{Var[Y - E(Y|X)]}{Var(Y)} \tag{1}$$

which can be measured as:

$$\eta(Y|X) = 1 - \frac{1}{N\sigma^2}\sum_i N_i\sigma_i^2 \tag{2}$$

where

$$\sigma^2 = \frac{1}{N}\sum_{\omega\in\Omega} Y(\omega)^2 - m^2, \quad m = \frac{1}{N}\sum_{\omega\in\Omega} Y(\omega) \tag{3}$$

$$\sigma_i^2 = \frac{1}{N_i}\sum_{\omega\in\Omega_i} Y(\omega)^2 - m_i^2, \quad m_i = \frac{1}{N_i}\sum_{\omega\in\Omega_i} Y(\omega) \tag{4}$$

Here Ω denotes the overlapping region of the two images; N is the number of voxels in Ω. Consider the distribution of X, if a histogram operation is performed on X to sample the distribution, then Ω_i denotes the voxels belong to the ith column of the histogram (i.e. $\Omega_i = \{\omega \in BIN(i)\}$); N_i is the number of voxels in Ω_i.

The value of CR is between 0 (no functional dependence or the two images are completely different) and 1 (fully deterministic dependence or the two images are exactly the same). Defined as a ratio, CR is invariant to the scaling of Y or $Y(\omega)$. Besides, CR is asymmetrical by definition, indicating that in general $\eta(Y|X) \neq \eta(X|Y)$.

Compared to MI, the computation of CR does not require 2D-histogram calculation, which makes it more suitable for GPUs which has very limited on-chip memory. Moreover, the computation complexity of CR is $O(n_x)$, better than $O(n_x n_y)$ for MI. Further, CR can generate comparatively accurate result while showing better robustness at lower resolutions [6] and less sensitive on sub-sampling [15]. These features are especially beneficial to a multi-resolution framework such as FLIRT.

2.4. Histogram calculation

The calculation of CR requires the values of $N_i, \sum_{\omega\in\Omega_i} Y(\omega)^2$ and $\sum_{\omega\in\Omega_i} Y(\omega)$ for each i, which contains histogram calculations. The CR histogram routine is shown in Listing 1.

For a single thread, histogram calculation is straightforward. As illustrated in Fig. 2, the thread simply goes through all the elements of an array, updating the target counters accordingly. However, to run it on a multithreaded machine like GPU, several threads may attempt to update the same counter simultaneously, leading to inconsistent results. Therefore, atomic operations are utilized to sequentialize these access. This solution preserves correctness, but significantly enlarges the updating latency and usually aggravates to be the major bottleneck of the application. The notion of *conflicts* is to describe the scenario that multiple threads update the same memory location, as shown in Fig. 2.

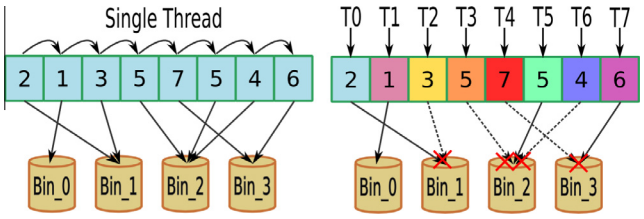


Fig. 2. Update conflicts for histogram calculation on multithreaded platform: If T0 and T2 update Bin 1 simultaneously, one may overwrite the other, leading to incorrect results.

Histogram conflicts are generally tackled by histogram replication [16], which is to allocate local copies of the shared histogram counters so as to reduce the conflict degree on those counters. We label such local copies as *shadow histogram* in this paper.

3. Registration algorithm

In this section, we describe the registration algorithm. We first present the skeleton for the whole algorithm and then introduce the proposed CR schemes, which are summarized at last.

3.1. Skeleton

The skeleton of our GPU-based registration algorithm is shown in Fig. 3. As can be seen, for every FLIRT stage, a half-sampling procedure is performed to convert the images into the operating resolution. An additional normalization is required provided the source image and reference image are not initially in the same resolution. After that, some preparation work are done in the "init volume" phase, mainly the allocation and configuration for GPU processing. For example, copying the images to texture memory and allocating histogram counters. Extra preprocessing follows if necessary. For each local search, depending on the search logic, the transform matrix is tuned before transferring to the GPU constant memory. Then histogram calculation is executed and the similarity degree – measured by CR is computed. The similarity calculation can take over 98% of the execution time for the registration process.

Since the transform is affine transform, it is *linear* or say, *invertible*: we can either map the source image to the reference image with transform f, or we can map the reference image to the source image with f^{-1}, which is just the inverse matrix of the transform matrix described in Section 2.1. In this work, we are choosing to map the reference image to the source image, i.e. $\eta(Source\ image|Reference\ image)$ is the CR similarity measure, because:

```
1  void histogram(int *bins, float *image){
2    for(int idx=tid; idx<imageSize; idx++){
3      float bin = calcBin(image[idx]);
4      float val = calcVal(idx,image);
5      h[bin]++; y1[bin]+=val; y2[bin]+=val*val;
6  } }
```

Listing 1. Histogram calculation for CR.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

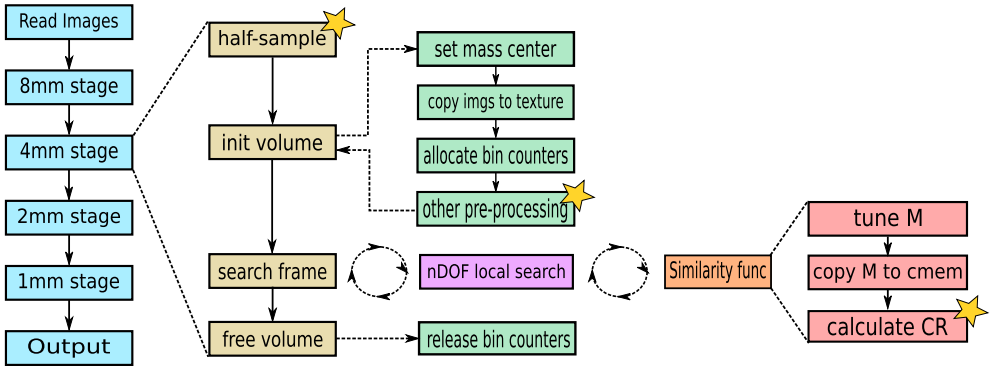


Fig. 3. Algorithm framework. The vertical arrows indicate module execution sequences. Dashed horizontal arrows implies module hierarchy. The dashed circle means the right-hand modules are called by the left-hand module repeatedly. The yellow star means the current module contains GPU kernel functions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

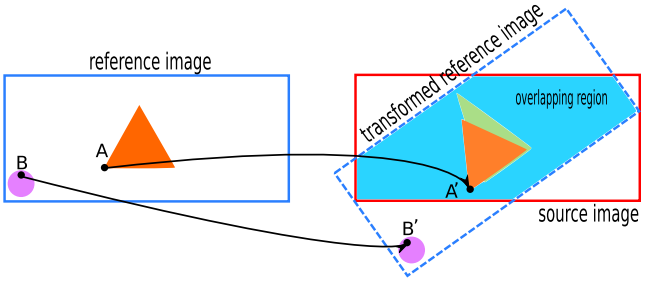


Fig. 4. Voxel mapping. Voxel $A(x,y,z)$ from the reference image is mapped to $A'(x',y',z')$ in the *transformed reference image* by multiplying the transform matrix M^{-1} to (x,y,z). A and A' have identical intensity value but different coordinates. Then we measure the intensity difference between A' and the point from source image with the same coordinate (x',y',z'). Note that only the voxels within the overlapping region are considered. Therefore, B is neglected in this scenario.

- The reference image size is generally smaller than or equal to the source image size, as the example shown in Fig. 1. In such cases, choosing the reference image as X in the evaluation of CR can reduce the number of voxels that has to be processed during histogram.
- The reference image is generally fixed (e.g. a template image). Thus, by taking reference image as X, it is possible to reuse the preprocessing outcome from the initialization phase for new source images.

The calculation procedure is shown in Fig. 4. The reference image is transformed to a *transformed reference image* by multiplying the transform matrix (M^{-1}) to all of its voxels. Then, the similarity between source image and the *transformed reference image* is computed.

In the remaining part, we present the proposed schemes alternately.

3.2. First scheme

The code segment is shown in Listing 2. Three counter arrays $h, y1, y2$ are allocated on global memory, corresponding to $N, \sum_{\omega\in\Omega} Y(\omega)^2$ and $\sum_{\omega\in\Omega} Y(\omega)$, respectively. Each counter accounts for a bin. During execution, from the current position in the reference image, a thread applies the transformation defined by the transform matrix and obtains the homologous coordinate in the

```
1  __global__ static void calcCRKernel(float *y1, float *y2, float *h){
2    const int threads=blockDim.x*gridDim.x;
3    const int tid=blockIdx.x*blockDim.x+threadIdx.x;
4
5    for(int idx=tid; idx<refSize; idx+=threads){
6      float x = idx % refsize.x;
7      float y = (idx / refsize.x) % refsize.y;
8      float z = (idx / refsize.x) / refsize.y;
9      float3 srcCorr = transform(x,y,z);
10
11     if(insideOverlappingRegion(srcCorr)){
12       float srcVox = tex3D(srcImg,srcCorr);
13       int bin = floor(refImg[idx])/binWidth;
14       float weight = calWeight();
15       atomicAdd( &h[bin],weight);
16       if(srcVox>0){
17         atomicAdd(&y1[bin],weight*srcVox);
18         atomicAdd(&y2[bin],weight*srcVox*srcVox);
19       }
20     }
21   }
22 }
```

Listing 2. Kernel code segment of the *global_atomic* scheme.

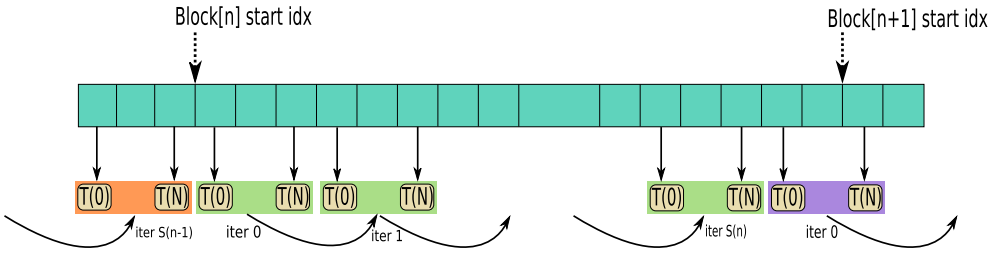


Fig. 5. Execution Procedure for *Conflict_free* Scheme. Thread block n traverses from position marked by Block[n] start index. After several iterations, it terminates at the primary element owned by the next block. During this term, each thread handles one element and calculates the corresponding $N_i, Y(\omega_i)^2$ and $Y(\omega_i)$. After all the rotations are accomplished, the threads submit their local accumulated values to shared memory followed by a block-wise reduction.

transformed reference image. Using this coordinate, the voxel is fetched from the source image. The thread then calculates the bin this voxel belongs to and updates the counters accordingly.

Note that the statement in line 17 improves the performance noticeably since for this scheme, atomic updates are the bottleneck and most of the intensities from the image background are zero. Meanwhile, the reference image voxels are only retrieved if the mapped coordinates fall in the overlapping region. This trick does not violate the coalesced memory access [9] but potentially save L2 bandwidth if L1 cache is disabled. It also reduces the number of memory requests provided all threads in a warp falling outside the overlapping region. The weight function in line 14 is used for smoothing which is discussed in Section 5.3.

The first scheme is a direct implementation of the histogram calculation using the hardware-based atomic primitives for GPU global memory. We label this scheme as *global_atomic*.

3.3. Second scheme

The bottleneck for the first scheme is the conflicts of atomic updates. Intuitively, we can allocate extra counters to mitigate the conflict degree. Imagine if every thread is equipped with a private copy of the histogram counters, there will be no conflicts at all. This is our second scheme: every thread updates its local shadow counters which are aggregated afterwards. We label this scheme as *global_merge*.

This scheme consumes huge space, therefore cannot fit into the shared memory. Some existing proposals attempted to circumvent this problem by assigning less storage per bin via bit-shifting [17], but these schemes strictly limit the number of bins and the bits allocated per bin. In the computation of CR, however, each bin requires three floating-point counters (12 bytes, see List 1), hence the overall consumption is far overloaded for the small on-chip shared memory.

The shadow bins avoid the possible updating collisions. However, due to enormous global memory traffic, especially most of them are non-coalesced, performance of the data cache and TLB are extremely low. Meanwhile, the final aggregation phase merges many zero counters which are redundant. This also induces overhead.

Table 1
Proposed schemes.

Scheme	Name	Characteristics
1	*global_atomic*	Atomic updating on global memory
2	*global_merge*	Per-thread shadow histogram on global memory
3a	*shared_atomic*	Intra-block atomic updating on shared memory; then inter-block atomic updating on global memory
3b	*shared_merge*	Intra-block atomic updating on shared memory; then inter-block merging on global memory
4	*shared_warp*	Warpwise atomic updating on shared memory
5	*conflict_free*	Conflict free scheme

Table 2
Platform configuration.

	System_1	System_2	
CPU	Intel Q8300	Intel i7-4770	
gcc	gcc-4.4.7	gcc-4.4.7	
GPU	GTX-570	GTX-TITAN	GTX-750 Ti
Architecture	Fermi	Kepler	Maxwell
Compute capacity	2.0	3.5	5.0
CUDA cores	15(SM) × 32	14(SM) × 192	5(SM) × 128
GPU frequency	1464 MHz	876 MHz	1137 MHz
Memory throughput	152 GB/s	288 GB/s	86.4 GB/s
Driver/runtime	6.5/4.0	6.5/6.5	

Table 3
OASIS image information.

Type	MR1
Subjects	OA_0001 to 22 (except 08)
dim 1–3	256x256x160
pixdim 1–3	(1,1,1) mm
datatype	4 Bytes
filetype	ANALYZE-7.5

3.4. Third scheme

Shared memory is exploited here concerning its fast speed for both raw and atomic memory access [9]. We allocate one shadow histogram for each thread block on the shared memory and integrate them afterwards. Consequently, the possible conflicts are separated into local intra-block conflicts and global inter-block conflicts.

Intra-block conflicts occur among threads of the same block, which are resolved by shared memory atomic operations. Inter-block conflicts, on the other hand, take place between thread blocks on the global memory. Two approaches can be deployed to resolve the inter-block conflicts. One is through global memory atomic operations, marked as *shared_atomic*; the other is via merging on global memory, marked as *shared_merge*. We will compare these two alternative schemes in the experiments.

3.5. Fourth scheme

The fourth scheme also exploits the shared memory. Concerning the updating conflicts are still significant for threads in a thread block, we allocate shadow histogram for each warp instead of block so that inter-thread conflicts can be further mitigated. However, the drawback is more shared memory usage that can limit the number of thread block in a streaming multiprocessor. A merging phase is still necessary and requires even more efforts since there are more shadow histograms. This scheme is labeled as *shared_warp*.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

```
__global__ static void calcCRKernel(float *y1, float *y2, float *h,
                const int* blkpos, const int* iref, const int* blk2bin){
  const int blksize=blockDim.x;
  const int tid=threadIdx.x; const int bid=blockIdx.x;
  const int startpos=blkpos[bid]; const int endpos=blkpos[bid+1];
  float ly1=0,ly2=0,lh=0;

  for(int idx=startpos+tid;idx<endpos; idx+=blksize){
    float x = idx % refsize.x;
    float y = (idx / refsize.x) % refsize.y;
    float z = (idx / refsize.x) / refsize.y;
    float3 srcCorr = transform(x,y,z);
    if(insideRegion(srcCorr)){
      float weight = calWeight();
      float srcVox = tex3D(srcImg,srcCorr);
      lh  += weight;
      ly1 += weight*srcVox;
      ly2 += weight*srcVox*srcVox;
    }
  }
  sh[tid]=lh; sy1[tid]=ly1; sy2[tid]=ly2;
  __syncthreads();
  //... merging on sh,sy1 and sy2...
  //... submit to h,y1,y2 ...
}
```

Listing 3. Kernel code segment of *Conflict_free* scheme.

3.6. Fifth scheme

Previous schemes explore design space from various aspects. However, none of them entirely resolve the conflicts problem despite trying different ways to mitigate the degree (actually there is no conflict in *global_merge*, however at the expense of enrolling enormous global memory access). Motivated by this, we propose the last design – *conflict_free* scheme. The basic idea is that, *since the cause of the conflict is the interleaved access to the shared histogram counters, why not sorting the voxels first so that each thread block can work on voxels belonging to the same histogram bin.*

3.6.1. Pre-processing

As aforementioned, in this scheme, a pre-sorting on the reference image is required, which takes place in the module of *other pre-processing* in Fig. 3. An array V_{index} is constructed with sequential series (i.e. 0, 1, 2, ...) as the initial values. Then, V_{index} is sorted, using the reference image intensities I_{ref} as the sorting key. After that, I_{ref} is in order. For an arbitrary element of I_{ref} with index i, we can still obtain its original unsorted index $V_{index}(i)$ so as its primary coordinate. In this way, we gather voxels with the same intensity together and make them continuously distributed but still conserving the original coordinate information.

We sort in the GPU so as to reduce data transfer. Effective sorting algorithms in GPU are radix sort [18] or bitonic sort [19]. In this work, we use the thrust library [20] for simplicity and efficiency. The associated overhead for sorting is measured in Section 5.2. After sorting, I_{ref} is returned to the host part and a routine is employed to traverse I_{ref} to mark the starting and ending positions for the workload of each thread block.

In the ideal case, the number of thread blocks is identical to the preestablished bin counts. However, due the severe workload unbalancing (For a typical MRI image, because of the large

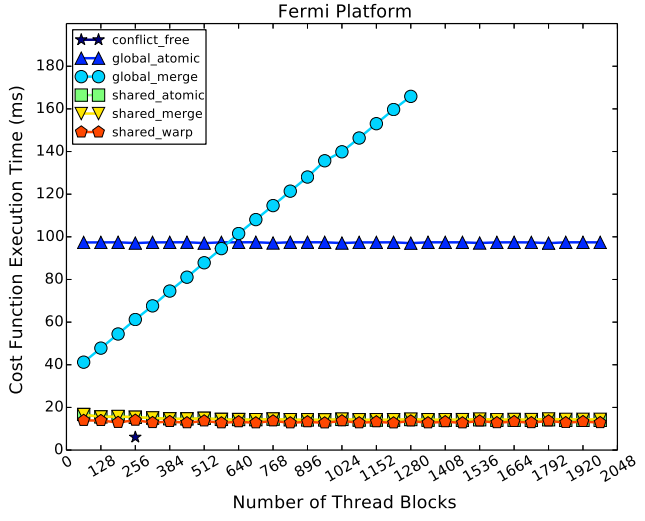


Fig. 6. Execution time for different thread blocks in Fermi GPU. *global_merge* breaks halfway is due to the shortage of global memory.

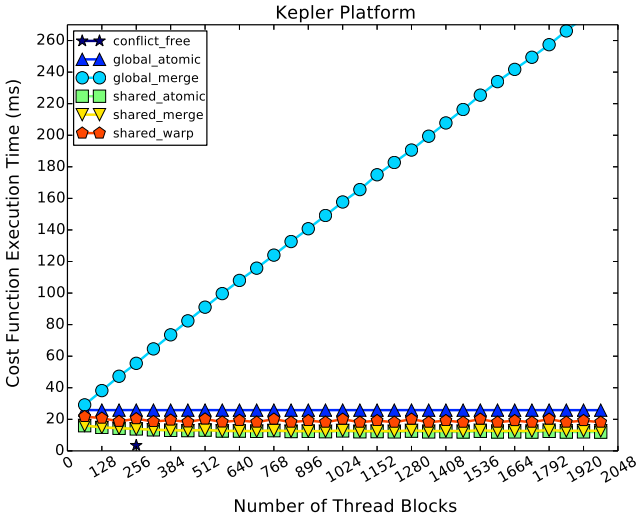


Fig. 7. Execution time for different thread blocks in Kepler GPU.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

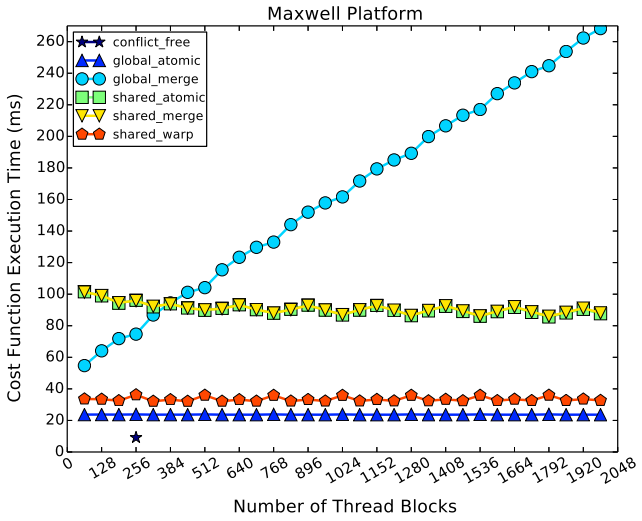


Fig. 8. Execution time for different thread blocks in Maxwell GPU.

background area, the first bin contains as much as 77% of all voxels.), we need to settle a threshold to limit the maximum workload (see Section 5.1 for detailed evaluation).

It is important to note that, unlike other conflict-free schemes with pre-sorting such as [12], the preprocessing and sorting stage in our scheme is executed **only once** for each resolution stage provided that the bin size is unchanged. Yet, the sorted data can be reused by the cost function for thousands of times within that stage. Further, if the reference image is invariant, the sorting results can be further reused for registration of other source images. This is a good property for the scenarios of multiple image registration.

3.6.2. Cost function

In the preprocessing phase, we know the starting and ending positions for every thread block. Inside the cost function kernel, a thread block goes through the partition of I_{ref} it is in charge of. As all the elements for the whole thread block belong to a unique bin, three per-thread registers are sufficient for the counters. The counters are accumulated locally in thread-scope registers first and then merged on the block-scope shared memory after a block-wise synchronization. This procedure and the corresponding code segment are illustrated in Fig. 5 and Listing 3.

Compared to former schemes, the advantages of scheme_5 are:

- During the whole life, a thread only works on a unique bin. Therefore, it has a very low demand for storage. In fact, 12 bytes are sufficient, which can be easily fulfilled by the fast registers.
- All threads from a thread block target the same bin so they can be rapidly aggregated through parallel reduction operations in the shared memory, after accomplishing their own tasks. Moreover, one thread only consumes 4 bytes of shared memory. Compared to *shared_atomic*, *shared_merge* and *shared_warp* schemes, it is more likely that a streaming processor can accommodate extra thread blocks when shared memory size is the limitation.
- All the conflicts are resolved through shadow counters in registers and parallel merging in the shared memory. This is very different from *global_merge* in which conflicts are resolved in the global memory at the expense of excessive global memory access.

3.7. Scheme summary

As a summary, all the proposed schemes are listed in Table 1.

4. Experiments

In this section, we compare the proposed schemes with the variation of three configuration factors and evaluate the performance for the entire application on different GPU platforms. Several observations are made and discussed through the comparisons.

4.1. Environment settings

The experiments are performed on three Nvidia GPUs of different hardware generations. Their configurations and the environment are listed in Table 2. The compiler optimization level is *-O3*. The L1 cache is disabled by setting the compiler option *-ptxas -dlcm=cg* since histogram calculation is generally cache-unfriendly. The interpolation method is nearest-point for stage 1

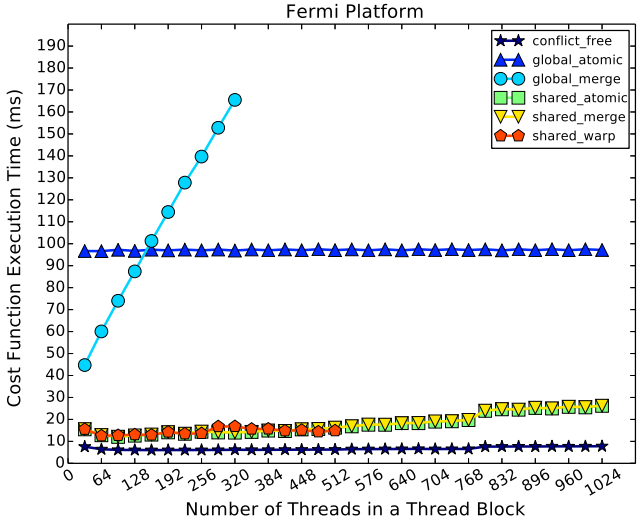


Fig. 9. Execution time for different threads per block in Fermi GPU. *global_merge* and *shared_warp* break halfway are due to the shortage of global memory and shared memory.

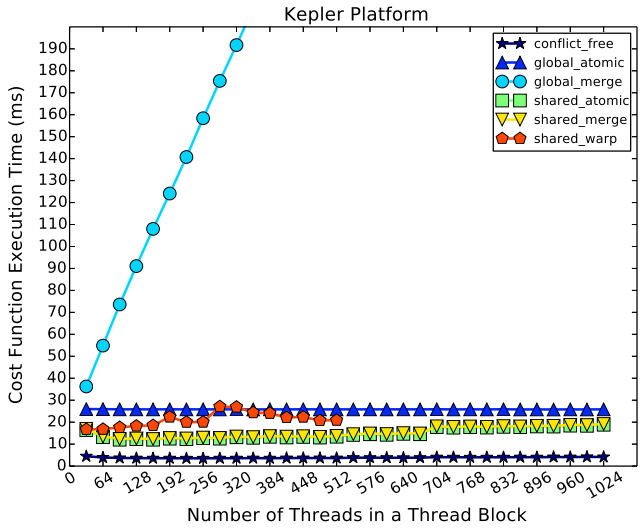


Fig. 10. Execution time for different threads per block in Kepler GPU. *shared_warp* breaks halfway is due to the shortage of shared memory.

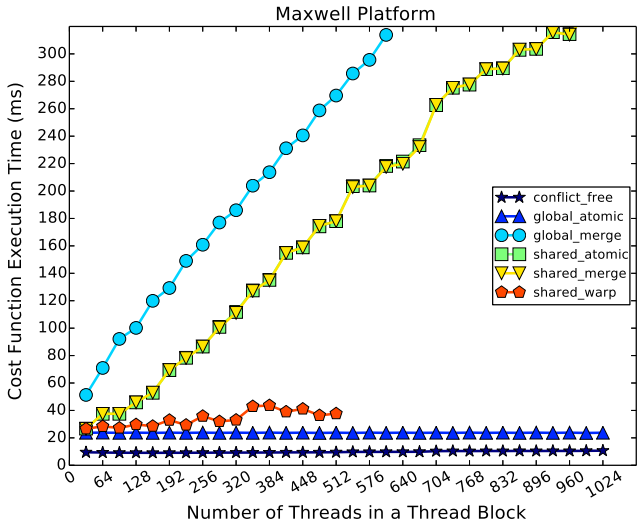


Fig. 11. Execution time for different threads per block in Maxwell GPU. *global_merge* and *shared_warp* break halfway are due to the shortage of global memory and shared memory.

and stage 2, but trilinear for the remaining stages. The texture addressing model is wrapping for all the three spatial axes.

The used dataset is obtained from the OASIS database which is public available [21]. We use 11 MR1 images indexed from OA_0001 to OA_0012 except OA_0008 (due to unavailability) for registration. The test plan is that we register the last 10 images to the first one (the information is listed in Table 3) and calculate the average value.

4.2. Cost function evaluation

First, we concentrate on the CR cost function and evaluate three factors that may influence the performance:

- **Number of thread blocks.** If fewer thread blocks are allocated, the remaining ones have to finish more jobs. This parameter may make a difference on resource usage but will not affect conflict degree.
- **Number of threads per block.** This factor may influence concurrency, resource usage and conflict degree.
- **Number of histogram bins.** This factor impacts resource usage and conflict degree.

Since the execution time for the cost function is very short, to avoid distortion, we run the cost function 10 times for each subject (i.e. one source image and the reference image) and obtain the average. We then do the test on all the 10 subjects and calculate their average value to be the result for a particular testing point.

4.2.1. Influence due to the variation of the number of thread blocks

Figs. 6–8 illustrate the variation of execution time with the number of thread blocks from 64 to 2048 for the Fermi, Kepler and Maxwell platforms. *conflict_free* has only one point because the number of thread blocks for this approach depends on the number of bins, which is fixed to 256 in this experiment. The block size is 256 threads.

As can be seen, inside each figure, only the *global_merge* curve is strongly correlated with the variation of thread blocks. This is reasonable as the *global_merge* scheme allocates shadow histograms for all threads: the more the thread blocks allocated, the more irregular global memory access, the lower the performance. For this scheme, clearly the bottleneck is the global memory throughput. Among the different platforms, Kepler shows the best performance as its global memory bandwidth is the highest (288 GB/s compared to 152 GB/s in Fermi and 86.4 GB/s in Maxwell).

Regarding the *global_atomic* scheme, it is observed that the efficiency of performing atomic operations on global memory has improved quite significantly – the execution time reduces from 95 ms in Fermi to 22 ms in Kepler. This substantial hardware improvement has been declared in the Kepler whitepaper. Although from Kepler to Maxwell the time reduction is not obvious, considering that there are fewer cores in Maxwell, such "performance unchanging" is still a step forward (Maxwell has a larger L2 cache thus a larger buffer for atomic operations on global memory). Therefore, it is suggested to use global memory atomic operations for the latest GPUs. Note that *global_atomic* has already becomes the best choice among these four tested schemes in Maxwell platform.

For the three shared memory related schemes, it is also observed that from Kepler to Maxwell, the efficiency of performing atomic operations on the shared memory has dropped significantly, especially in the case of inter-warp conflicts. We see the corresponding execution time for *shared_merge* and *shared_atomic* increase from about 18 ms to more than 80 ms while at the same

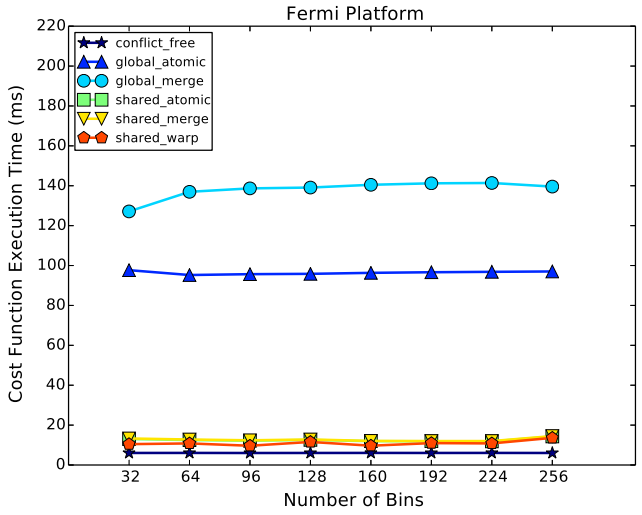


Fig. 12. Execution time for different histogram bins in Fermi GPU.

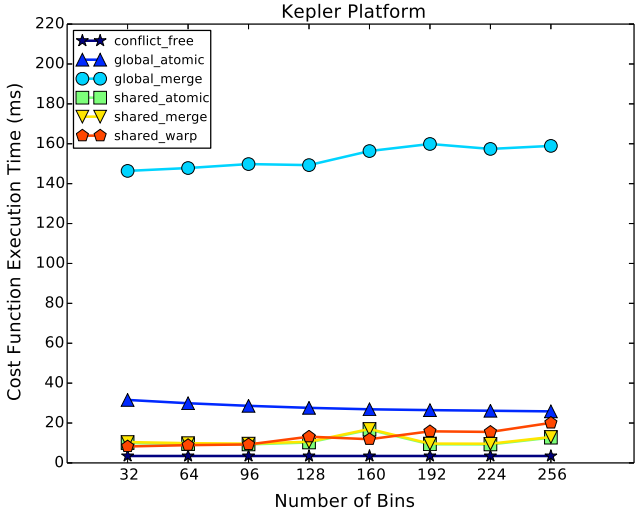


Fig. 13. Execution time for different histogram bins in Kepler GPU.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

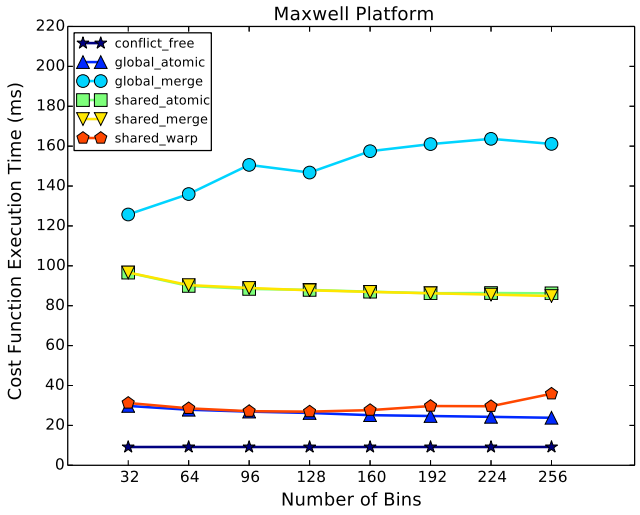


Fig. 14. Execution time for different histogram bins in Maxwell GPU.

time, the delay of *shared_warp* scheme has only increased slightly. In Maxwell Tuning Guide [22], Nvidia states that the *lock-update-unlock* pattern for realizing atomic operation in Fermi and Kepler has been replaced by a native implementation in Maxwell. This might explain the performance degradation here.

4.2.2. Influence due to the variation of the number of threads per thread block

Figs. 9–11 depict the variation of execution time with the number of threads in a thread block from 32 to 1024 for all schemes in Fermi, Kepler and Maxwell platforms, respectively. The volume of thread blocks is 1024 and the number of bins is 256.

Still, the *global_merge* curves are linearly correlated with the block size as more threads are initiated, which means more shadow histograms have to be allocated in the global memory. The *global_atomic* curves are irrelevant to the variation of thread block size as the overall conflict degree in global memory can be hardly influenced by this factor.

The remaining four schemes show similar variation pattern: the curve drops slightly at the beginning and then rises slowly but steady after a minimum point. This is because for the initial part, more threads leads to more parallelism. Although some extra shared memory conflicts are introduced, the benefit is larger. However, beyond the inflection point, the conflict overhead becomes the major factor hence we see the continuous performance degradation. It should be noted that in Maxwell, such degradation is quite significant. When comparing with the *shared_warp* curve, it is obvious that the bottleneck is the inter-warp conflicts (rather than the intra-warp conflicts). Therefore, it is not suggested to leverage the atomic operations in the shared memory of Maxwell GPUs.

Table 4
Application test configuration.

	Fermi		Kepler		Maxwell	
	Threads	Blocks	Threads	Blocks	Threads	Blocks
global_atomic	256	15	1024	14	1024	5
global_merge	192	15	256	14	256	5
shared_atomic	96	120	128	224	32	160
shared_merge	96	120	128	224	32	160
shared_warp	256	90	256	112	64	160
conflict_free	256		128		128	

4.2.3. Influence due to the variation of the number of bins

Figs. 12–14 illustrate how the execution time varies with the number of bins set by the algorithm. The number of thread blocks is 1024 while thread block size is 256.

As can be seen, most of the schemes are insensitive to the variation of bin size. The *global_merge* curves rise slightly, but not in a degree as sharp as the conditions in Section 4.1.1 and 4.1.2. It is because the number of threads allocated is not changed, so is the number of memory transactions. However, more bins imply a larger memory space for the messy access during histogram, therefore the poorer L2 cache efficiency (L1 cache is disabled). The *global_atomic* curves decrease marginally in Kepler and Maxwell platforms. The reason is that in general, more histogram bins imply lower average conflict degree, thus lower execution time. Similar effect are also observed for other shared memory relevant schemes at their beginning phases. However, since the shared memory is very small and has to be distributed among the concurrent thread blocks, more bins leads to higher shared memory consumption. Consequently, at the ending phases, the parallelism is damaged and the performance is affected. This is especially the case for the *shared_warp* scheme.

To summarize, we have the following observations:

- The *conflict_free* scheme demonstrates the best performance in all experiments. Besides, it is very stable and insensitive to the variation of the three factors: the number of thread blocks, the number of threads per block and the number of bins.
- The *global_merge* scheme behaves the worst in all experiments meaning that avoiding some conflicts at the expense of excessive irregular global memory access is not worthwhile.
- From Fermi to Kepler to Maxwell, the efficiency of global memory atomic access has improved significantly. So it is more advisable to adopt *global_atomic* schemes in Maxwell GPUs.
- From Kepler to Maxwell, the overhead of resolving inter-warp conflicts in shared memory has degraded considerably. So *shared_atomic* and *shared_merge* schemes are not good choices anymore for Maxwell platform.
- Except the *conflict_free* scheme, *shared_warp* shows the best performance. However, it is highly sensitive to the shared memory usage.

4.3. Application evaluation

We proceed to the experiments for the whole registration process. The configurations for similarity function are listed in Table 4, which are extracted from the experiment results described in Section 4.2. For those values that are constant (e.g. thread blocks), we choose the value that is just sufficient to reach the highest theoretical occupancy for that platform [23]. The number of bins is varied depending on the present stage, say 64, 128, 256, 256 for stage 1, 2, 3, 4, respectively.

Meanwhile, since no CR based GPU acceleration scheme has been published before, to make a solid demonstration, the proposed CR-based schemes are also compared with several existing MI-based approaches (see Section 6): Shams' per-warp method [12], sort-and-count method [24], Chens' method [13], Vetters' method [10] and the native CPU implementation (denoted as *MI_Sham*, *MI_sort*, *MI_Chen*, *MI_Vetter* and *MI_cpu*, respectively). In fact, Sham et al. also propose a per-thread scheme in [12]. However, this method is similar to the *global_merge* scheme and the acceleration technique is only feasible for small numbers of bins; it is therefore excluded from the results. Note we replace the software simulated atomic operations with their hardware-based counterparts in these approaches since some of them are based on the older generation GPUs. The figures for *MI_cpu* and *CR_cpu* are obtained in System 1 (see Table 2).

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

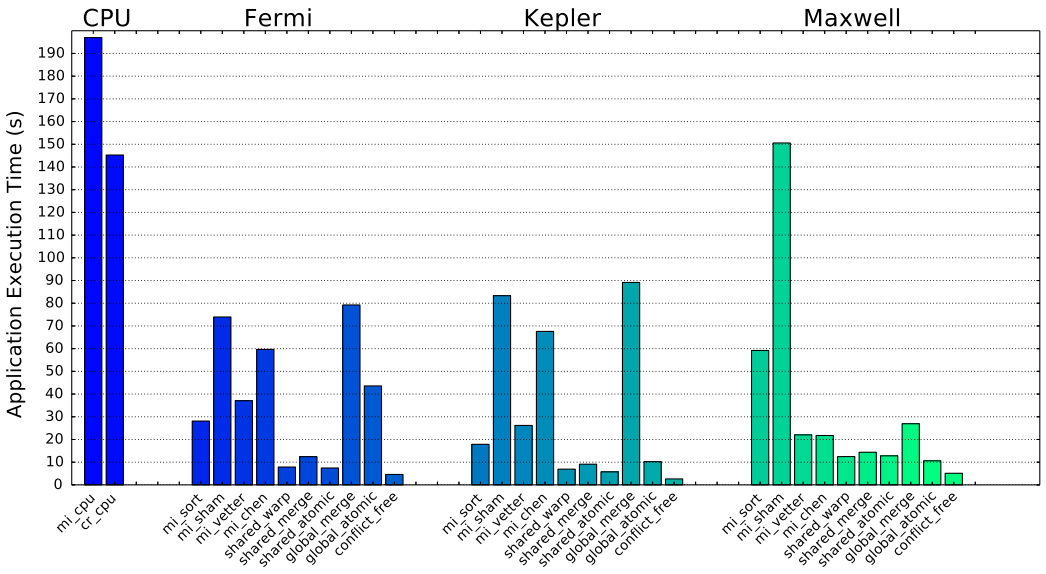


Fig. 15. Application execution time for different schemes on different platforms. As can be seen, *conflict_free* always shows the best performance, especially on Kepler GPU.

Table 5
Average speedup over native version.

	gb_atom	gb_merg	sh_atom	sh_merg	sh_warp	cf_free
Fermi	3.3×	1.8×	19.5×	11.7×	18.5×	31.6×
Kepler	14.2×	1.6×	25.3×	15.9×	20.9×	55.3×
Maxwell	13.7×	5.4×	11.3×	10.1×	11.7×	28.4×

Fig. 15 illustrates the average application execution time across the 10 different source images (from 10 patients). As can be seen, in general the CR based methods run faster than the MI based methods in both CPU and GPUs except *global_merge*. Particularly, the *conflict_free* achieves the best performance among all the approaches in all GPU platforms. The speedup of the proposed schemes over *CR_cpu* is listed in Table 5.

Fig. 16 shows the RMS error between output images (i.e. registered source images) and reference image. The measurement of error is not straightforward due to the lack of a standard baseline. So we use the intensity standard deviation (RMS) between the output images and the reference image as the metric:

$$RMS(X,Y) = \sqrt{\frac{1}{N}\sum_{\omega\in\Omega}(X(\omega)-Y(\omega))^2} \tag{5}$$

where Ω is the overlapping region, N is the number of voxels in Ω.

As can be seen, both MI and CR show consistent figures for different schemes. The CR in GPUs is slightly better than in CPU while MI is worse. However, overall the CR based methods show a lower RMS than MI based methods, in both CPU and GPUs, which highlights the great advantage of CR.

Meanwhile, except *global_merge*, all the other proposed schemes achieve good speedup (see Table 5). In particular, the *conflicts_free* scheme reaches a speedup of 55x over the baseline on Kepler with however, less RMS. The execution time is 2.63s compared with 145.3s in for *CR_cpu*.

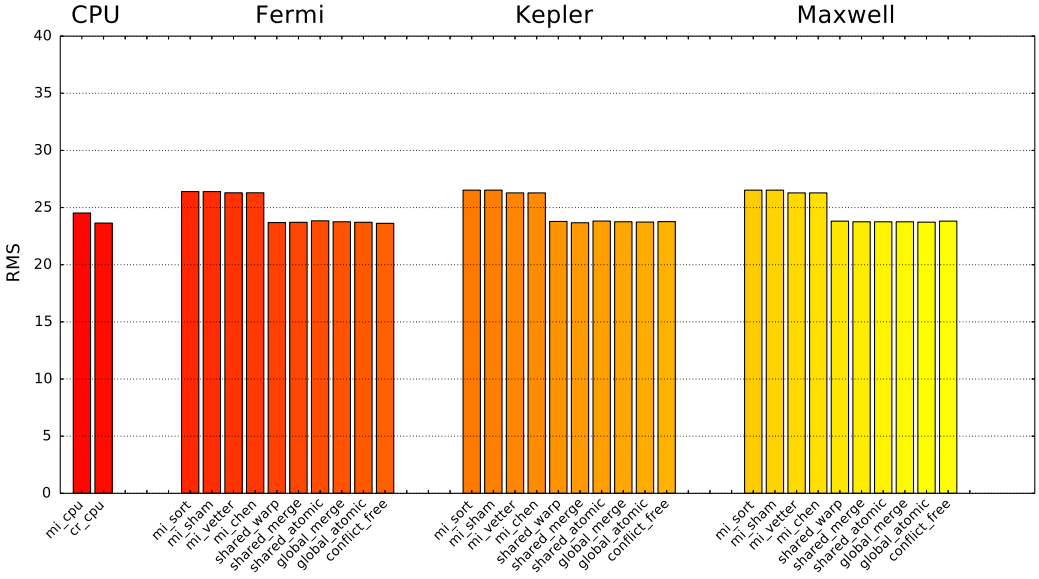


Fig. 16. RMS error for the registration application on different platforms. Clearly, CR achieves lower error than MI on CPU and GPUs. The slightly differences among differnt MI or CR schemes on GPUs are due to the accumulated rounding error via different reduction approaches.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

Table 6
Histogram result for 8 bins.

Bin	1	2	3	4	5	6	7	8
Voxels	8,065,900	1,073,218	742,400	369,655	113,187	90,263	29,056	2081
%	76.92	10.24	7.08	3.53	1.08	0.86	0.28	0.02

```
1  for(int i=0; i<refSize; i++){
2    if(i-preI >= MAX_BLK_WORKLOAD || (val=iref[i]/binWidth)!=preVal){
3      blkpos[blkId]=i;
4      blk2bin[blkId]=val;
5      preVal=val;
6      preI=i;
7      blkId++;
8    }
9  }
```

Listing 4. Scheme_5 post-sorting process.

The low performance of *global_merge* suggests that the overhead of excessive global memory access is much higher than some atomic operations when referring to *global_atomic*, not to mention that the efficiency of performing global atomic operations has improved considerably from Fermi to Kepler. Meanwhile, comparing between *global_atomic* and *shared_atomic*, it is obvious that atomic operations on shared memory are more efficient in the old hardware (i.e. Fermi), but probably not the case for the new hardware anymore (i.e. Maxwell). Further, the comparison between *shared_atomic* and *shared_merge* indicates if the updating conflicts are not severe on global memory, the overhead of merging is often larger than that of atomic operations. For *shared_atomic*, after intra-block conflicts are resolved on shared memory, inter-block conflicts are much lighter and thus can be easily hidden by multi-threading since thread blocks are independent and progress in their own contexts. Finally, the performance of *shared_atomic* being similar to *shared_warp* indicates that the conflicts among threads of the same thread block are not very severe that allocating shadow histograms at the warp level is not beneficial.

5. Discussion

In this section, we analyze three topics about the conflict free scheme and their impact on performance.

5.1. Per-thread-block workload unbalancing

GPU distributes thread blocks among streaming processors following a round robin fashion based on the assumption that the workload for each thread block is roughly identical. However, this is not the case for *conflict_free* scheme if each thread block accounts for an entire histogram bin. Table 6 shows an example of one histogram calculation between OA_0002 and OA_0001 with 8 bins.

As can be seen, the proportion of voxels belonging to Bin 1 is about 77% where Bin 8 is less than 0.02%. This is extremely unbalanced. So if 8 thread blocks are allocated, the 8th streaming processor will be idle for more than 99.97% of the total execution time. Further, if there are many thread blocks and a heavy-loaded thread block is dispatched at the last time of the execution, the processor utilization would be even lower since all the other streaming processors are idle at that time. So there should be a threshold to limit the maximum workload for a thread block. Though some overhead may be induced, it is well worth: the total execution time has dropped by a factor of over 3 in our test. The code segment for this post-sorting procedure is presented in Listing 4.

The question arises what the optimal threshold is. Since more thread blocks introduce extra overhead (initialization, merging, dispatch etc.), this threshold is in fact a compromise between workload balance and the overhead due to extra thread blocks. Fig. 17 illustrates the variation of execution time with the workload for a thread block in different platforms. In the left part of the inflexion point, the overhead is the major factor while in the right part, the effect of workload unbalancing becomes more evident. From the figure, we learn that 1280, 512, 768 (vox/block) are the optimal thresholds for Fermi, Kepler and Maxwell, which means each thread has to process 5, 2, 3 voxels at maximum. The corresponding execution time are 4.6 s, 2.6 s and 5.1 s, respectively.

5.2. Overhead of presorting

The sorting process was presented in Section 3.6.1. Here we evaluate its overhead. Since the stages have distinct resolutions and different image sizes, we list the time expense and throughput for the sorting kernels and the delay for the entire sorting phase (including GPU memory allocation and copy) on different platforms for each stage in Tables 7–9 respectively. It can be seen that the sorting kernels achieve higher throughput for larger datasets.

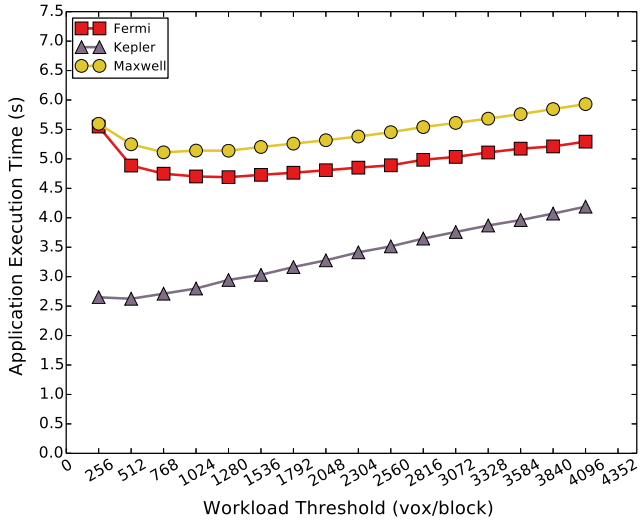


Fig. 17. Average execution time over the maximum per-block workload. The curve drops drastically at the beginning and increases thereafter indicating that after the flex point, the impact of load unbalancing appears to be prominent.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002

Table 7
Fermi sorting overhead for stages.

Stage	Vox size	Kernel time (μs)	Throughput (vox/μs)	Overall delay (μs)
1	20,480	678	30.2	766
2	163,840	1532	106.9	2166
3	1,310,720	3122	419.8	5814
4	10,485,760	16,249	645.3	57,312

Table 8
Kepler sorting overhead for stages.

Stage	Vox size	Kernel time (μs)	Throughput (vox/μs)	Overall delay (μs)
1	20,480	1086	18.9	1141
2	163,840	1411	116.1	1722
3	1,310,720	2815	465.6	3932
4	10,485,760	12,766	821.4	26,195

Table 9
Maxwell sorting overhead for stages.

Stage	Vox size	Kernel time (μs)	Throughput (vox/μs)	Overall delay (μs)
1	20,480	1435	14.3	1554
2	163,840	2068	79.2	2863
3	1,310,720	4677	280.2	8459
4	10,485,760	27,346	383.4	55,371

The phase of sorting is only beneficial if its outcome could be reused sufficiently. From this perspective, the more searches inside a stage, the higher benefit we can expect from this preprocessing. For other schemes, the scale of reference image dictates the capacity of workload, thus the degree of conflicts. Therefore, a larger volume reference image will benefit more from the sorting, further showing the advantage of the *conflict_free* scheme.

In fact, the *conflict_free* scheme does not really need a complete sorting but a process to aggregate voxels with the same intensity values together. This is also reflected in Listing 4. Further, if a series of registration jobs share the same reference image, the sorting results can be recycled among them. Note we do not apply this optimization technique in the experiments.

5.3. Smoothing of the border region

The FLIRT algorithm deploys a weighting method to eliminate the discontinuity of the similarity function caused by the simultaneous variation of the amount of the overlapping region and the voxel intensities inside (so both the numerator and denominator in Formula 4 change, leading to the discontinuity and local optimum of CR) during registration. The weighting method gives a lower influence coefficient to the voxels near the edges of the overlapping region (see [3] for details). This apodization approach is also implemented in the proposed CR implement (i.e. *calWeight()* in Listing 2). In the previous experiments, we disabled this function by returning 1. Here we evaluate its performance impact. The results are presented in Figs. 18 and 19.

As weighting distance increases, more points all into the boundary region hence more extra process and computation are needed. However, from the figures, we can observe that the execution time is almost unchanged, showing that these additional computations can be fully hidden through multi-threading and the long latency of texture fetching. Besides, the RMS curve fitting into a convex figure indicates that small weighting distance helps to reach global optimal, but if the weighting distance becomes too large, the accuracy of the similarity function is affected. For this dataset, 34 is shown to be the optimal weighting distance.

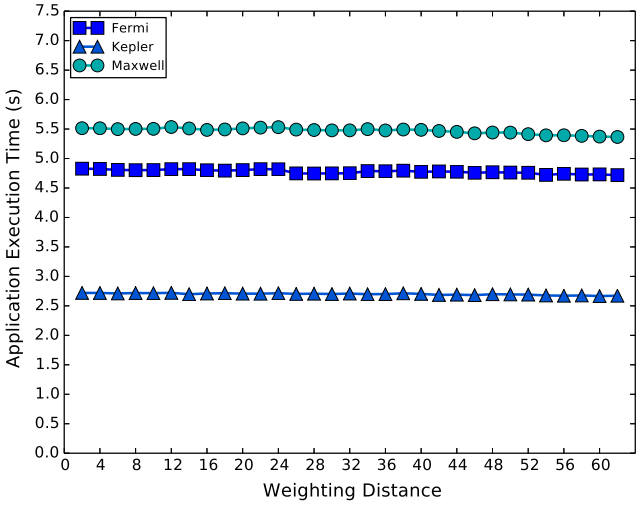


Fig. 18. Average execution time over weighting distance. The execution time seems irrelevant to the size of weighting distance indicating that these extra computation overhead are completely hidden by multithreading and the long latency texture fetching.

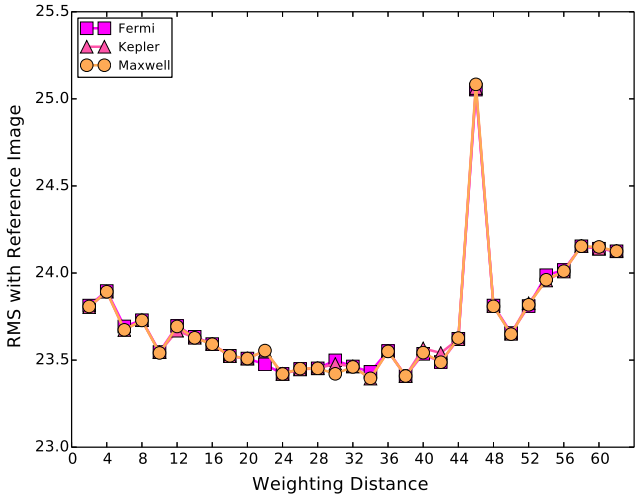


Fig. 19. RMS error with the reference image over weighting distance. The curve decreases at the beginning showing that weighting the border region does help to improve global accuracy but if the weighting distance is too large, significant bias are suffered.

6. Related work

Ever since CUDA has been published [9], several works are proposed about realizing the image registration algorithms on GPUs [10–13,25]. However, the interleaved and concurrent writing access to a limited memory region by massive threads make this migration a difficult task because atomic operations are employed to preserve updating correctness meanwhile introducing serious overheads, especially when numerous threads are competing for the same bin location. Shams et al. [11,12] maintain a number of shadow histograms in global memory (when shared memory is out of range) and aggregate afterwards to alleviate the degree of conflicts, or keep partial number of shadow histograms in shared memory but traverse the image several times to cover the entire bin range. Chen et al. [13] sort the reference volume beforehand

to restrict the shadow histograms' range when counting the joint entropy. Vetter et al. [10] also perform a pre-sorting to narrow the space required to fit into shared memory and guarantee coalescing access. They further mitigate the collisions by allocating more counters for the "fat bin" that appears to incur more conflicts based on an intensity distribution figure generated from a pre-profiling. All of these methods, however, are mutual information based and conflicts unavoidable. Though Shams et al. proposed a delegate sort-and-count algorithm to achieve atomic operations free in [24], this sort-and-count phase has to be performed by each thread block every time the cost function is invoked, thus leads to additional performance overhead.

7. Conclusion

In this paper, we presented five Correlation Ratio based image registration schemes that are optimized for GPU acceleration. Through comparisons among different schemes on different GPU platforms, we observed that: (1) while atomic operations on shared memory are much faster than on global memory for old GPUs, it is not the case for new GPUs; (2) The overhead of massive access is larger than that of atomic operations on global memory; (3) If conflicts are not too severe, atomic operations are a better choice than merging. Particularly, we proposed a scheme that totally avoided conflicts and achieved more than 55 times' speedup compared to native implementation with lower RMS error and was very stable. This design is based on the algorithmic characteristics of CR, showing its great advantage on GPUs when compared with the MI based approaches. Additionally, we evaluated the impact of workload balancing, sorting and smoothing on performance and accuracy of the conflict-free scheme, which suggested that: (1) workload balancing is crucial for delivering performance; (2) the sorting overhead can be amortized on larger dataset and via sufficient reuse of the sorted results; (3) proper smoothing can reduce registration error without impairing performance.

Acknowledgements

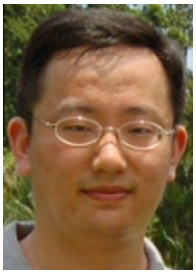
Part of this work has been done in the Department of Imaging and Interventional Radiology, The Chinese University of Hong Kong (CUHK). Special thanks are owed to Dr. Wang Defeng and Dr. Shi Lin from CUHK for their advices on the initial design of the proposal. Thanks to Dr. Wu Qiang from Hunan University (HNU) for his comments on the draft of the paper. We also thank all the reviewers for their insightful suggestions and feedbacks.

References

[1] B. Zitova, J. Flusser, Image registration methods: a survey, Image Vis. Comput. 21 (11) (2003) 977–1000.
[2] J. Maintz, M.A. Viergever, A survey of medical image registration, Med. Image Anal. 2 (1) (1998) 1–36.
[3] M. Jenkinson, P. Bannister, M. Brady, S. Smith, Improved optimization for the robust and accurate linear registration and motion correction of brain images, Neuroimage 17 (2) (2002) 825–841.
[4] M. Jenkinson, S. Smith, A global optimisation method for robust affine registration of brain images, Med. Image Anal. 5 (2) (2001) 143–156.
[5] A. Klein, J. Andersson, B.A. Ardekani, J. Ashburner, B. Avants, M.-C. Chiang, G.E. Christensen, D.L. Collins, J. Gee, P. Hellier, et al., Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration, Neuroimage 46 (3) (2009) 786–802.
[6] A. Roche, G. Malandain, X. Pennec, N. Ayache, The correlation ratio as a new similarity measure for multimodal image registration, in: Medical Image Computing and Computer-Assisted Interventation, MICCAI, Springer, 1998, pp. 1115–1124.
[7] J.P. Pluim, J.A. Maintz, M.A. Viergever, Registration of medical images: a survey, IEEE Trans. Med. Imaging 22 (8) (2003) 986–1004.
[8] R. Shams, P. Sadeghi, R. Kennedy, R. Hartley, A survey of medical image registration on multicore and the GPU, IEEE Signal Process. Mag. 27 (2) (2010) 50–60.
[9] Nvidia, Programming Guide, 2008. <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>.
[10] C. Vetter, R. Westermann, Optimized GPU histograms for multi-modal registration, in: IEEE International Symposium on Biomedical Imaging: From Nano to Macro, IEEE, 2011, pp. 1227–1230.
[11] R. Shams, N. Barnes, Speeding up mutual information computation using Nvidia CUDA hardware, in: 9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications, IEEE, 2007, pp. 555–560.
[12] R. Shams, R. Kennedy, Efficient histogram algorithms for Nvidia CUDA compatible devices, in: Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS), 2007, pp. 418–422.
[13] S. Chen, J. Qin, Y. Xie, W.-M. Pang, P.-A. Heng, CUDA-based acceleration and algorithm refinement for volume image registration, in: International Conference on Future BioMedical Information Engineering, FBIE, IEEE, 2009, pp. 544–547.
[14] K. Ikeda, F. Ino, K. Hagihara, Efficient acceleration of mutual information computation for nonrigid registration using cuda, IEEE J. Biomed. Health Inform. 18 (3) (2014) 956–968.
[15] A. Roche, G. Malandain, N. Ayache, X. Pennec, et al., Multimodal image registration by maximization of the correlation ratio, 1998.
[16] J. Gomez-Luna, J.M. Gonzalez-Linares, J.I.B. Benitez, N.G. Mata, Performance modeling of atomic additions on GPU scratchpad memory, IEEE Trans. Parallel Distrib. Syst. 24 (11) (2013) 2273–2282.
[17] V. Podlozhnyuk, Histogram calculation in CUDA, Nvidia Corporation, White Paper, 2007.
[18] N. Satish, M. Harris, M. Garland, Designing efficient sorting algorithms for manycore GPUs, in: IEEE International Symposium on Parallel & Distributed Processing, IPDPS, IEEE, 2009, pp. 1–10.
[19] H. Peters, O. Schulz-Hildebrandt, N. Luttenberger, Fast in-place sorting with CUDA based on bitonic sort, in: Parallel Processing and Applied Mathematics, Springer, 2010, pp. 403–410.
[20] N. Bell, J. Hoberock, Thrust: A 2 6, GPU Computing Gems Jade Edition, 2011, p. 359.
[21] D.S. Marcus, T.H. Wang, J. Parker, J.G. Csernansky, J.C. Morris, R.L. Buckner, Open access series of imaging studies (OASIS): cross-sectional MRI data in Young, and demented older adults, J. Cognit. Neurosci. 19 (9) (2007) 1498–1507.
[22] Nvidia, Maxwell Tuning Guide, 2015. <http://docs.nvidia.com/cuda/maxwell-tuning-guide/>.
[23] Nvidia, Occupancy Calculator, 2009. <http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls>.
[24] R. Shams, P. Sadeghi, R. Kennedy, R. Hartley, Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images, Comput. Methods Prog. Biomed. 99 (2) (2010) 133–146.
[25] P. Muyan-Ozcelik, J.D. Owens, J. Xia, S.S. Samant, Fast deformable registration on the GPU: a CUDA implementation of demons, in: International Conference on Computational Sciences and Its Applications, ICCSA'08, IEEE, 2008, pp. 223–233.

Ang Li received the B.S. degree in software engineering from Zhejiang University (ZJU), China, in 2010. From 2010 to 2012, he worked as a software engineer on GPU technology in the industry in Shanghai and Hong Kong, China. Since 2012, he started to purse the Ph.D degree in the joint Ph.D program between National University of Singapore (NUS), Singapore and Eindhoven University of Technology (TUe), Eindhoven, The Netherlands. His research interests include performance modeling and optimizations for GPUs.

Akash Kumar received the B.Eng. degree in computer engineering from the National University of Singapore (NUS), Singapore, in 2002. He received the joint Master of Technological Design degree in embedded systems from NUS and the Eindhoven University of Technology (TUe), Eindhoven, The Netherlands, in 2004, and received the joint Ph.D. degree in electrical engineering in the area of embedded systems from TUe and NUS, in 2009. In 2004, he was with Philips Research Labs, Eindhoven, The Netherlands, where he worked on Reed Solomon codes as a Research Intern. From 2005 to 2009, he was with TUe as a Ph.D. student under project PreMaDoNA. Since 2009, he has been with the Department of Electrical and Computer Engineering, NUS. Currently, he is working as an Assistant Professor in the department. He has published over 60 papers in leading international electronic design automation journals and conferences. His current research interests include analysis, design methodologies, and resource management of embedded multi-processor systems.



Yajun Ha received the B.S. degree from Zhejiang University, China, in 1996, the M.Eng. degree from National University of Singapore, Singapore, in 1999, and the Ph.D. degree from Katholieke Universiteit Leuven (KULeuven), Leuven, Belgium, in 2004, all in electrical engineering. He is currently the Deputy Director & Research Scientist of I2R-BYD Joint Lab at Institute for Infocomm Research, Singapore. Before this, he was an Assistant Professor with the Dept. of Electrical and Computer Engineering, National University of Singapore. From January 1999 to February 2004, he worked as a researcher with the Inter-University MicroElectronics Center (IMEC), Leuven, Belgium. His research interests include the general area of embedded computing (VLSI) architectures, circuits and design methodologies, with the focus on reconfigurable computing and smart and autonomous electrical car applications. He has published around 80 internationally peer-reviewed journal/conference papers on these topics. Dr. Ha has served a number of positions in the professional communities. He serves as the Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS (2011–2013), the Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS (2013–2014), and the Journal of Low Power Electronics(since 2009). He serves as the General Co-Chair of ASP-DAC 2014; Program Co-Chair for FPT 2010 and FPT 2013; Chair of the Singapore Chapter of the IEEE Circuits and Systems (CAS) Society (2011 and 2012); Member of ASP-DAC Steering Committee; and Member of IEEE CAS VLSI and Applications Technical Committee. He is the Program Committee Member for a number of well-known conferences in the fields of embedded systems and FPGAs, such as DAC, DATE, ASP-DAC, FPGA, FPL and FPT.

Henk Corporaal received the M.S. degree in theoretical physics from the University of Groningen, Groningen, The Netherlands, and the Ph.D. degree in electrical engineering, in the area of computer architecture, from the Delft University of Technology, Delft,The Netherlands. He has been teaching at several schools for higher education. He has been an Associate Professor with the Delft University of Technology in the field of computer architecture and code generation. He was a Joint Professor with the National University of Singapore, Singapore, and was the Scientific Director of the joint NUS-TUE Design Technology Institute. He was also the Department Head and Chief Scientist with the Design Technology for Integrated Information and Communication Systems Division, IMEC, Leuven, Belgium. Currently, he is a Professor of embedded system architectures with the Eindhoven University of Technology, Eindhoven, The Netherlands. He has co-authored over 250 journal and conference papers in the (multi)processor architecture and embedded system design area. Furthermore, he invented a new class of very long instruction word architectures, the Transport Triggered Architectures, which is used in several commercial products and by many research groups. His current research interests include single and multiprocessor architectures and the predictable design of soft and hard real-time embedded systems.

Please cite this article in press as: A. Li et al., Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.04.002