

# Cache-Aware GPU Memory Scheduling Scheme for CT Back-Projection

Ziyi Zheng and Klaus Mueller *Senior Member, IEEE*

**Abstract**—Graphic process units (GPUs) are well suited to computing-intensive tasks and are among the fastest solutions to perform Computed Tomography (CT) reconstruction. As previous research shows, the bottleneck of GPU-implementation is not the computational power, but the memory bandwidth. We propose a cache-aware memory-scheduling scheme for the back-projection, which can ensure a better load-balancing between GPU processors and the GPU memory. The proposed reshuffling method can be directly applied on existing GPU-accelerated CT reconstruction pipelines. The experimental results show that our optimization can achieve speedup ranging from 1.18-1.48. Our cache-optimization method is particular effective for low-resolution volumes with high resolution projections.

## I. INTRODUCTION

Computed Tomography has been widely used in medical society. Recent researches focus on how to perform the computation on parallel computing devices, to yield the speed satisfying the clinical need. GPU implementation is a feasible high-performance solution along with other choice, such as Cell processors and FPGAs. For the analytical reconstruction algorithm, such as FDK, GPU-based solutions can match the speed of data generation by X-ray scanners [1], [2]. Nevertheless, for low-dose CT which typically requires iterative reconstructions, GPU-based reconstructions have a much longer running time [3].

Many GPU-acceleration works [4], [5] identified that the memory bandwidth was the bottleneck of the GPU-based back-projection problem. This problem roots in the limited amount of cache and insufficient bandwidth of device memory in the GPU architectures. Previous works [4], [5] increased the memory bandwidth by reordering-loops, unrolling loops and multithreading. In this paper, we focus on how to improve the cache hit rate specifically. We propose a new volume shuffling method ensuring better cache behavior depending on different projection angles. Our method can be directly added into the CUDA cone-beam CT frameworks [2-4], [6].

The rest of the paper is organized as follows. We begin in Section II by giving a brief introduction of GPU architecture. Section III discusses cone-beam back-projection. Section IV shows a summary of existing methods and Section V describes our cache-aware optimization. Section VI presents experimental results. Finally, Section VII concludes the paper.

---

Manuscript received Mar 4, 2010. This work was supported by NSF grant EAGER 1050477.

Ziyi Zheng is with the Center of Visual Computing, Computer Science Department at Stony Brook University, Stony Brook, NY 11794-4400 USA (telephone: 631-327-5940, e-mail: zizhen@cs.sunysb.edu).

Klaus Mueller is with the Center of Visual Computing, Computer Science Department at Stony Brook University, Stony Brook, NY 11794-4400 USA (telephone: 631-632-1524, e-mail: mueller@cs.sunysb.edu).

## II. GPU ARCHITECTURE

All Current GPUs are based on single instruction multiple data (SIMD) design. Here we take a recently released GPU – NVIDIA GeForce GTX 480 as an example to discuss the GPU architecture.

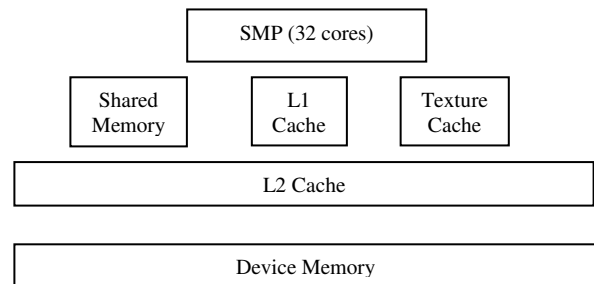


Fig. 1. The GPU processor-memory configuration for one SMP

A GTX 480 GPU card contains 480 processors and has theoretical computational power reaching 1.3 TFLOPS in single floating-point precision. It contains 15 *streaming multi-processor* (SMP) which can perform tasks independently from each other. A group of 32 processors forms a SMP, as shown in Fig. 1. These processors within one SMP can share an L1 cache and access a fast on-chip user-controllable cache — shared memory. One shared memory access costs 2 clock cycles in the best case, compared to hundreds of clock cycles for off-chip memory (also called device/global memory) access. The amount of shared memory and L1 cache in one SMP is user-configurable (16KB + 48KB or 48KB + 16 KB). All 15 SMPs will share a single chunk of L2 cache (768KB). Bilinear filtering is not done in SMPs but in texture filtering units. The peak filtering rate for the GTX 480 is 42G texels/s. There is a separate texture cache dedicated for filtering (average 12KB per SMP) while it still uses the same L2 cache and device memory.

Device memory is an off-chip memory that stores the input data and receives the output from the processors. The GTX 480 has 1.5GB DDR5 device memory with peak bandwidth 177.4 GB/s. The maximum GPU global bandwidth can only be achieved by memory coalescing, which essentially translates into 1 memory instruction per 128 bytes. This implies 32 neighbouring threads (a warp) read/write within a 128-byte-aligned segment. With proper alignment, sequential mapping of threads to memory address will yield a coalesced memory access pattern.

NVIDIA GPUs can be programmed via a C-like API — CUDA. CUDA is a general purpose API which exposes more control over how a task is computed on the GPU hardware, as compared to graphics-based APIs (CG, GLSL). The task-

hardware mapping is enabled by introducing the concept of “block”. Each *block* is mapped to an SMP.

CUDA kernels are executed in terms of thread *blocks*, whereas the total task is called *grid*. On the hardware level, each *block* is mapped to a single SMP. In the back-projection stage of the CT reconstruction, SMPs are assigned to different regions of the resulting volume sequentially. This enables a mapping where the *grid-block* decomposition in CUDA corresponds to the volumetric reconstructed 3D dataset.

### III. CONE-BEAM BACK-PROJECTION

In the back-projection computation, the commonly used method is the voxel-driven method. For example, slabs in the volume (Fig. 2) are mapped to thread *blocks* in CUDA. This volume-to-slabs decomposition has been used in [2-4]. Another type of mapping is based on decomposing the volume into horizontal tiles and each tile is mapped to a CUDA thread *block*, as used in [5], [6]. While the former use 2D square tiles, the latter uses 1D linear tiles. We take the slab-based approach as an example to show how to optimize the hardware mapping to improve cache hit rate.

Each 3D slab of the reconstructed-volume is assigned to a SMP. In the depth dimension, the brick extends through the reconstructed volume. The slab is further divided into vertical tiles. Here, the product of the tile’s width  $w$  and height  $h$  needs to be below the maximum number of threads per *block*  $L$ . In the NVIDIA Fermi GPU,  $L = 1024$ . The tile’s width should be a multiple of 32 to ensure a coalesced writing pattern.

When performing the back-projection inside each slab, the order of execution of this back-projection follows along the set of 2D vertical tiles arranged in depth order. After we finish back-projecting, we incrementally store the resulting slices back in global memory.

The mapping of the kernel to the projection geometry is depicted in Fig. 2. The concurrent execution of threads inside an SMP is limited by the maximum number of threads that can be run there. The NVIDIA 480 has 15 SMPs. A set of projections  $i_a$  to  $i_b$  is processed at the same time. Then the total projected area processed by the GPU, arising from projection  $i_a$  to projection  $i_b$  is:

$$\sum_{i=i_a}^{i_b} A_i = 15l^2 \left( \sum_{i=i_a}^{i_b} \max\{thread\} |\cos(\theta_i)| \right) \quad (1)$$

In Eq. 1,  $\max\{thread\}$  is the maximum resident threads for each SMP,  $\theta_i$  is the projection angle for  $i^{th}$  projection and  $l$  is the ratio of source-object distance over source-detector-distance. Assume  $C_{cache}$  is the constant representing the amount of L2 caches, the projected area should be smaller than the cache limit (Eq. 2) to ensure better cache hit-rate.

$$\sum_{i=i_a}^{i_b} A_i \leq C_{cache} \quad (2)$$

Back-projection implementations [1-6] all use texture memory for projection data storage to deal with irregular memory fetching pattern. Using texture memory has several advantages. First, the trilinear/bilinear interpolation is supported at hardware level, which can reduce the computation cost of GPU processors. Second, hardware-

scheduled cache can benefit data reading with locality. Last, except locality, texture memory has no other constraint such as coalescing or confliction.

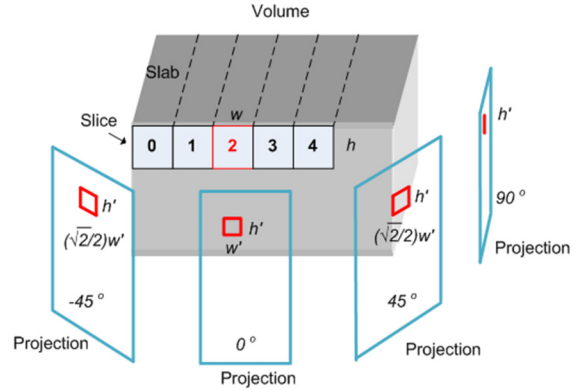


Fig. 2. Illustration of bricks, slices, slice’ projections and corresponding SMPs. In the volume, different numbers represent the different SMPs. The red color indicates the input and output regions involved in SMP number 2.

### IV. BACK-PROJECTION ORDERING

According to different orderings of back-projection loops, the published methods can be classified into three categories.

#### A. Single Projection Method

```

FOR each projection
  FOR each (slab) slice along the depth dim
    Accumulate projected values in the current slice
    Write results to current slice
  END
END

```

The idea behind this scheme is to obtain a maximum reuse of the SMPs’ L2 cache holding the projection. The locality of the texture reading is increased since the amount of overspreading of the reading location is restrained. Then for each slice there is a cache-miss at the beginning but not likely thereafter. The disadvantage of this method is memory-writing overhead.  $N$  projections will require reading and writing the computed results  $N$  times. Also, when we perform incremental writing on the global memory, the L2 cache will also cache the output slices. This undesirable cache behavior will reduce the effective capacity of L2 cache. Keck et. al. [3] used this method in CUDA-based SART. This method is better when cache-miss penalty is larger than memory writing overhead.

#### B. Multiple Projection Method

```

FOR each (slab) slice
  FOR each projection
    Accumulate projected values in the current slice
  END
  Write results to current slice
END

```

Xu and Muller [1] used this method but their work was based on the classic graphics pipeline. This method does not have writing output overhead ( $N$  projection only need to write the computed result once). The cache performance will be the bottleneck of this approach. The downside of this method is that cache for the input projection will be depleted very fast.

Noël et. al. [6] also used this method in their tile-based decomposition CUDA kernel. This method is better when cache-miss penalty is less than memory writing overhead.

### C. Hybrid Ordering Method

```

FOR each set of N/J projections
  FOR each (slab) slice
    FOR each J projections
      Accumulate projected values in the current slice
    END
  Accumulate results to current slice
END
END

```

Okitsu et. al. [5] used this method in their tile-based decomposition kernel. This method requires the trade-off between better cache and less output. The hybrid ordering improves the cache hit rate. The downside of this method is that the innermost loop for  $J$  projections is usually unrolled to avoid conditional branches. But it will require more registers and will result in register spilling which will reduce concurrency in the GPU.

This method helps explore the balance between single-projection and multiple-projections. The innermost loop number  $J$  needs to be small enough to not exceed the L2 cache size (Eq. 2). On the other hand,  $J$  needs to be large enough such that the number of slice updates in global memory is minimized.

## V. CACHE-AWARE METHOD

### A. 3D Reshuffling

As for projection angles, there can be two cases:

1. Back-Projection within  $[45,135]$ ,  $[225, 315]$  degrees.

A 2D slice with area  $w \times h$  is projected into a region with area within  $[0, (\sqrt{2}/2)w' \times (\sqrt{2}/2)h']$ , the cache problem is not very severe. Problem only arise when projections are concentrated on  $45+90k$  degrees, where  $k \in \mathbb{N}$ .

2. Back-Projection around  $[135,225]$ ,  $[-45, 45]$  degree.

A 2D slice with area  $w \times h$  is projected into a region with area within  $[(\sqrt{2}/2)w' \times (\sqrt{2}/2)h', w' \times h']$ . Then the SMP will be depleted of cache frequently.

We propose a method using a 3D transpose to improve the cache-hit rate. Note that rotating the volume by 90 degree will change case 2 into case 1, but writing the results directly into global memory will cause the coalescing problem. The reason can be explained for the 2D case since we essentially switch the row-major storage into column-major, which is not sequentially stored in memory any more but scattered into different memory segments. We use shared memory as a buffer to perform the matrix transpose inside each SMP before writing to the device memory, to preserve the coalescing pattern of the output. Also, we avoid shared memory bank conflicts by using a  $33 \times 4$  byte pitch. This will facilitate 32 parallel memory-conflict free threads. Another copy of the volume need to be allocated in the device since the 3D transpose is not in-place.

The reshuffling from XYZ to ZYX is listed below:

1. Load a 2D XZ slices into shared memory

2. 2D transpose to ZX slices
3. Write to ZYX in device memory

## VI. RESULTS AND DISCUSSION

The implementation was based on NVIDIA's CUDA 3.2 and on the GTX 480. All CUDA Kernels had the same configuration:  $32 \times 8$  threads per block. The inputs were  $364 \times 1024 \times 768$  projections on a half circular trajectory. We reconstructed a 3D volume within the FOV with two different resolution,  $256^3$  and  $512^3$ . We tested the different running times for the Single projection method (S), Multiple projection method (M) and Hybrid ordering method (H). We then added the comparison to our optimized 3D transpose methods (T). As shown in Table I, our transpose method had better performance regardless of the back-projection loop-ordering.

TABLE I. RUNNING TIME FOR 364 PROJECTIONS IN MS

	S	M	H	S+T	M+T	H+T
$256^3$	1110	1018	1181	785	688	785
$512^3$	4814	4880	5001	4360	4263	4060

The 3D Transpose method used  $16 \times 16$  CUDA blocks. Although it wasted some bandwidth, as opposed to the  $32 \times 32$  CUDA block, since the memory accesses were not fully aligned, the  $16 \times 16$  configuration was experimentally faster than  $32 \times 32$  and it achieves occupancy 1. Our 3D transpose method took 2ms for a  $256$  volume and 20ms for a  $512^3$  Volume.

TABLE II. FINAL RESULTS AND SPEEDUP

	Our (ms)	Projection/sec	Speedup
$256^3$	688	529	1.48
$512^3$	4060	90	1.18

The final back-projection performance is shown in Table II. For a  $256^3$  volume, our remapping method with shared-memory-enabled transpose achieved a total speedup of 1.48. For a  $512^3$  volume, our remapping method achieved a total speedup of 1.18.

Besides NVIDIA, other major GPU manufactures are ATI and Intel. In this paper, we focused on NVIDIA's GPUs to evaluate different acceleration techniques. Although the specification of GPUs may change, the concept and algorithm introduced in the paper can generally port to a different GPUs by using a cross-vendor programming API – OPENCL.

## VII. CONCLUSIONS

We present a volume reshuffling scheme optimizing the locality of the memory reading pattern in GPU-based back-projection. Our scheme can yield better cache hit rate and can be added to variety of existing methods with different back-projection ordering [2-4], [6], except for those implementations that use a square-tile in block-level mapping [5]. The results show our method is particularly useful for

reconstructions with low-resolution volume with high-resolution projections.

#### REFERENCES

- [1] F. Xu, and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology*, vol. 52, pp. 3405-3419, 2007.
- [2] L. Hillebrand, R. Lapp, Y. Kyriakou, and W. Kalender. "Interactive GPU-accelerated image reconstruction in cone-beam CT", In *Proceedings of SPIE*, pp. 72582A-72582A-8, 2009.
- [3] B. Keck, H. Hofmann, H. Scherl, M. Kowarschik, and J. Hornegger, "GPU-Accelerated SART Reconstruction Using the CUDA Programming Environment", In *Proceedings of SPIE 7258-72582B*, 2009.
- [4] H. Scherl, B. Keck, M. Kowarschik, J. Hornegger, "Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA)", Nuclear Science Symposium, Medical Imaging Conference, Vol. 6, pp. 4464-4466, 2007.
- [5] Y. Okitsu, F. Ino, K. Hagihara, "High-performance cone beam reconstruction using CUDA compatible GPUs", *Parallel Computing*, vol. 36, no. 2-3, pp. 129-141. 2010.
- [6] P. Noël, A. Walczak, J. Xu, J. Corso, K. Hoffmann, S. Schafer, " GPU-based cone beam computed tomography ", *Computer Methods and Programs in Biomedicine*, vol. 98, no. 3, pp. 271-277, 2010.