Optimizing CUDA Shared Memory Usage

Shuang Gao, Gregory D. Peterson

Electrical Engineering and Computer Science, University of Tennessee

Abstract

CUDA shared memory is fast, on-chip storage. However, the bank conflict issue could cause a performance bottleneck. Current NVIDIA Tesla GPUs support memory bank accesses with configurable bit-widths. While this feature provides an efficient bank mapping scheme for 32-bit and 64-bit data types, it becomes trickier to solve the bank conflict problem through manual code tuning. This paper presents a framework for automatic bank conflict analysis and optimization. Given static array access information, we calculate the conflict degree, and then provide optimized data access patterns. Basically, by searching among different combinations of inter- and intra- array padding, along with bank access bit-width configurations, we can efficiently reduce or eliminate bank conflicts. From RODINIA and the CUDA SDK we selected 13 kernels with bottlenecks due to shared memory bank conflicts. After using our approach, these benchmarks achieve 5%-35% improvement in runtime.

Background

Given an array ***A*** of 4-byte elements, by setting the bank access width as 4-bytes or 8-bytes, the array data are mapped in row-major or column-major way. The proposed approach optimizes bank access efficiency by inter-padding, intra-padding, and bank bit-width configuration. Figure 1 describes how these three schemes impact the conflict degree. Since manually exploring the large potential solution search space is tedious and time consuming, our approach supports automated optimization.

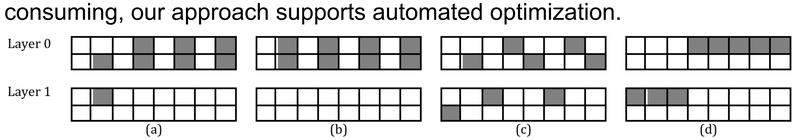


Figure 1: Using inter-padding, intra-padding, and changing bank access bit-width to eliminate bank conflicts: (a) original problem has 2-way conflict, (b) change access offset, (c) change access stride, and (d) use 8-byte bank access bit-width.

Single Warp bank conflict analysis

Row-major bank mapping has no conflict when the stride is odd. For even strides, we divide them into two categories: (1) stride is power-of-two, and (2) other even valued strides. For the **column-major mapping**, both even and odd strides could cause bank conflict. Table 1 describes methods we use to analyze different cases.

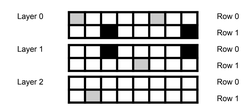
Row-major Bank Mapping		Column-major Bank Mapping
Odd Stride		**Odd Stride**
No Conflict.		Calculate the conflict caused by sites visited in different rows. An example: stride=5. 
Stride=2^e		
stride>=2^{b+r}	conflict degree is the warp size	
2^b <= *stride* < 2^{b+r}	conflict degree is *ceil(GCD x 2^{s-b}/r)*	
stride < 2^b	conflict degree is *ceil(GCD/r)*	
Other even strides		**Even strides**
For *stride=s x 2^e*, the warp visited sites can be divided into 2^e groups each has *s* rows with no conflict. Then we calculate the conflict among the ith rows of all groups.		the problem can be transformed either to an odd stride problem, or to a conventional bank access problem which can be solved by *GCD*.

Table 1: Single warp bank conflict analysis

Single Expression Bank Analysis

Warps share same access pattern and their offset is periodic. Following steps are used to get the conflict number:

- Calculate the number of distinct offsets
- Calculate the conflict for each offset
- Calculate the warp numbers that share the same offset.
- Compute the total conflict number.

Parameter Optimization

For each bit-width configuration, it looks for the best intra-padding solution for all arrays. Choose the bit-width configuration that has minimum conflict. If conflict number is not 0, use inter-padding to reduce conflicts.

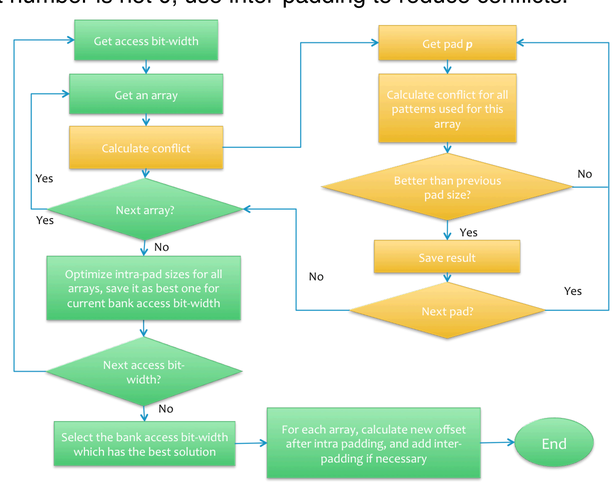


Figure 2: parameter optimization strategy

Results

6 benchmarks (13 kernels) from RODINIA and the CUDA SDK are used to test the approach. The test platform includes a Tesla 20c with CUDA 5.0. The results in figure 3 show that this approach greatly reduces the bank access replay number and results in 5%-35% execution time improvement for these benchmarks.

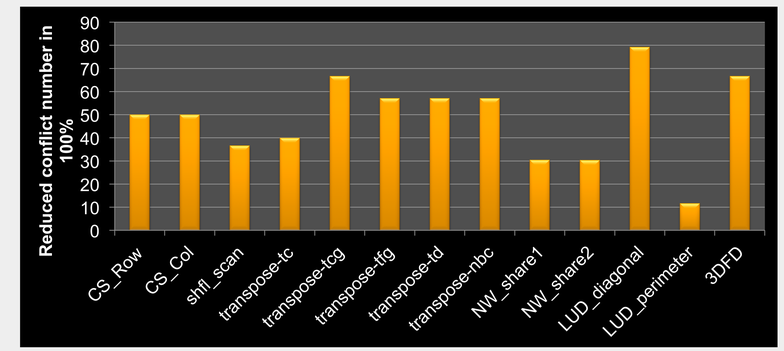


Figure 3: Reduced bank access replay in 100%

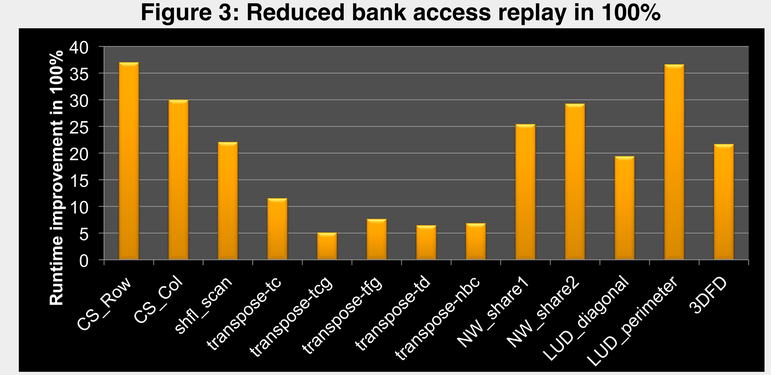


Figure 4: Runtime improvement in 100%

Conclusion

1. Using this approach, we obtain an average 19% improvement for a set of benchmark applications.
2. Static bank conflict analysis is an efficient method since warps commonly share same array access patterns.

References

[1] Baskaran, M.M., et al. *A compiler framework for optimization of affine loop nests for GPGPUs.* in *Proceedings of the 22nd annual international conference on Supercomputing*. 2008. ACM.

[2] Fetterman, M., et al., *Dynamic bank mode addressing for memory access*. 2012, Google Patents.

This work is partially funded by the Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476