MADALGO – CENTER FOR MASSIVE DATA ALGORITHMICS

Nodari Sitchinava
KIT

Karlsruhe Institute of Technology

Volker Weichert
Goethe University

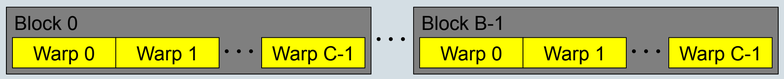
GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Danmarks Grundforskningsfond
Danish National Research Foundation

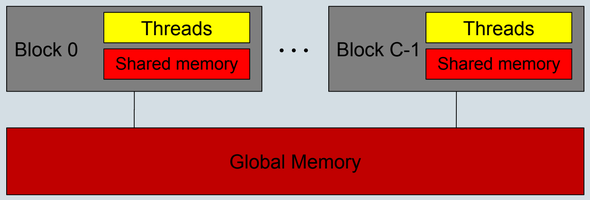
Provably Efficient GPU Algorithms

GPU Programming

GPU kernels run on **C** blocks (CTAs) of **W** *warps*. Each warp is a group of **w**=32 threads, which are executed in SIMD fashion.



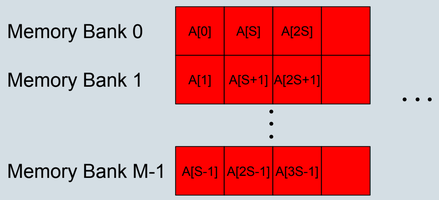
Each CTA has a small but fast private cache, that is shared among its threads (*shared memory*). All CTAs can access the large global memory, which has a higher latency than shared memory (ca. 2 orders of magnitude).



While intra-CTA synchronization is basically free, GPU programs are divided into **R** *kernels* to facilitate inter-CTA synchronization via CPU.

Bank Conflicts

Shared memory is organized in **S** memory banks that can be accessed simultaneously without penalty. When data is stored, the memory banks are used cyclically. On modern GPUs, **S=w.**



Bank conflicts happen, when multiple threads access different words on the same shared memory bank at the same time. If that happens, the memory access are performed sequentially, which makes runtimes hard to predict in case of random or data dependent memory accesses.

Other Relevant Observations

- We observe that accesses to global memory can achieve maximum bandwidth if performed in blocks of size **w** (takes care of modelling coalesced memory accesses).
- We limit the CTA size to one warp. Since some resources are shared among CTAs, there is a hardware limit to the number of CTAs, that can be accommodated at the same time. While possible, it is not beneficial to use more CTAs.

Cost of a GPU Algorithm

- Our model [1] is based on the Parallel External Memory (PEM) model by Arge et al. [2]
- There is unlimited global memory, while shared memory is of size **M**.
- We can view each CTA as a virtual processor with **w** cores that work in SIMD fashion.
- Generic kernel layout:
  1) Load data from global memory to shared memory.
  2) Work on the data in shared memory.
  3) Write output data to global memory.
- There is unlimited global memory, while shared memory is of size **M**.
- In most applications, **n** >> **M**, so each CTA has to work on multiple *pages* of input data.

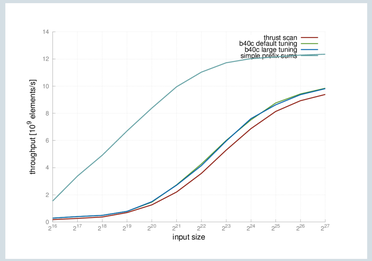
The total runtime of a GPU algorithm on input size **n**, consisting of **R(n)** kernels and executed on **p** processors is

$$T(n) = O\left(\sum_{k=1}^{R(n)} \left(t_k(M)\frac{n}{Mp} + \lambda q_k(M)\frac{n}{Mp} + \sigma\right)\right)$$

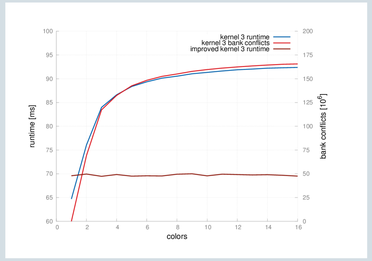
where $\mathbf{t_k}$ is the amount of parallel computation performed in kernel **k**, **λ** is the latency for accesses to global memory, $\mathbf{q_k}$ is the amount of parallel data transfer (*I/O*) and **σ** is the time needed for a global synchronization.

Experiments

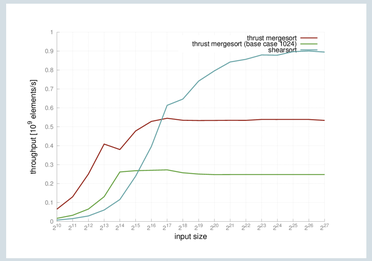
We implement a bank-conflict free algorithm for prefix sums using out design principles and compare against thrust [2] and back40computing [3]:



We extend our implementation to colored prefix sums. The design process shows the importance of bank conflicts for the kernel runtime:



We also implement the first sorting algorithm for small inputs that causes no bank conflicts. We replace the base case of thrust mergesort with our shearsort based implementation:



References

[1] N. Sitchinava and V. Weichert. *Provably Efficient GPU Algorithms*. Submitted to SPAA 2013.
[2] L. Arge, M. Goodrich, M. Nelson and N. Sitchinava. *Fundamental parallel algorithms for private-cache chip multiprocessors*. SPAA 2008.
[3] http://code.google.com/p/thrust/
[4] http://code.google.com/p/back40computing/