Random Permutation on a GPU – Is Your Algorithm Unbiased for n ≠ 2^m ?

Michael Waechter, Kay Hamacher, Franziska Hoffgaard, Sven Widmer, Michael Goesele

TECHNISCHE UNIVERSITÄT DARMSTADT

Motivation

Parallel random permutation generation can, e.g., be used

- **to perturb the input of a subsequent algorithm in order to make worst-case behavior unlikely, or**
- **to perform computations only on a small sampled subset of the whole problem space in order to draw conclusions about the whole problem space, e.g.,**
  - **in statistical science and modeling, or**
  - **in bioinformatical phylogenetic reconstruction**

Especially bioinformatical problems tend to be computationally expensive, which justifies the need for massive parallelization using GPU-based processing.

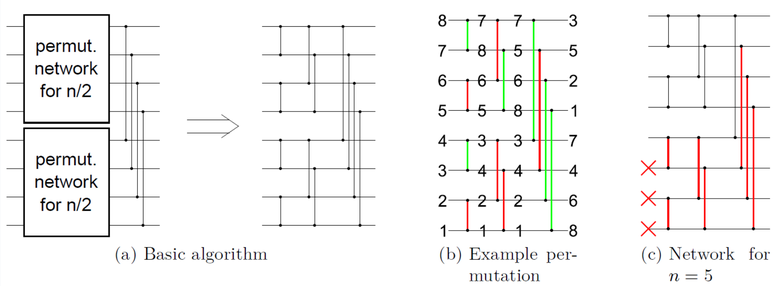
Requirements for GPU-based permutation algorithms are:

- **scalability to a number of processing elements close to the problem size**
- **in-placeness**
- **few thread synchronizations, little contention resolving**
- **little communication among threads**

Most existing random permutation algorithms belong to one of five categories (see Cong and Bader [1]):

- **Rand_Sort, Rand_Dart, Rand_Shuffle, Rand_Dist**
- **Permutation Networks**

Permutation networks meet the requirements for our GPU setting best, e.g., a butterfly network with a time complexity of O(n log n) as shown beneath:



(a) Basic algorithm (b) Example permutation (c) Network for $n = 5$

Such networks are typically defined for sizes of n = 2^m .For non-powers of two (e.g., n = 5) the trivial generalization would be to simply leave out exchanges that involve non-existing array elements. But this generalization does not generate all possible permutations with equal likelihood yielding a bias.

Bias

A permutation algorithm's bias can be described using stochastic permutation matrices:

$$M_n = \begin{bmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{bmatrix}$$

p_{ij} is the probability of element i of the initial array being permuted into position j of the final array. For the butterfly network and n = 5 this becomes

$$M_5 = \begin{bmatrix} 1/8 & 1/8 & 1/8 & 1/8 & 1/2 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \\ 1/8 & 1/8 & 1/8 & 1/8 & 1/2 \end{bmatrix}$$

which is clearly non-uniform. We define a bias measure as

$$B(M_n) = \frac{1}{n^2} \sum_{i,j} \frac{\left| M_n(i,j) - 1/n \right|}{1/n} = \frac{1}{n} \sum_{i,j} \left| M_n(i,j) - 1/n \right|$$

The green curve in Figure 1 (k = 1) shows the bias $B(M_n)$ of the butterfly network for various input data sizes n. k gives the number of iterations of the permutation applied to the input array (see Bias Reduction).

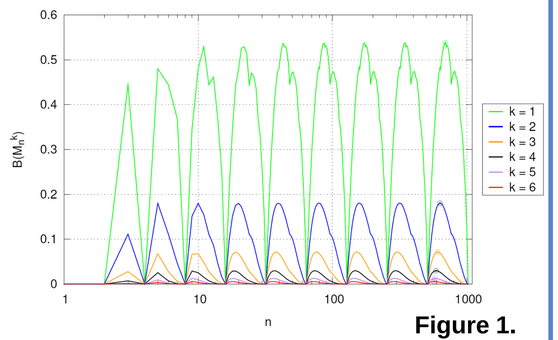


Figure 1.

We note that the bias observed in the butterfly network is not a pathological example. The need for n to be a power of two results naturally from the divide and conquer paradigm and the abstraction to arbitrary n is far from trivial. In fact, we also obtained a similar bias for the permutation algorithm described by Waksman [2].

References

[1] Cong, G., Bader, D.A.: An empirical analysis of parallel random permutation algorithms on SMPs. ISCA PDCS 2005
[2] Waksman, A.: A permutation network. J. ACM 15, 1968
[3] Waechter, M. et al.: Is Your Permutation Algorithm Unbiased for n ≠ 2^m ?. To be presented at PPAM 2011.
www.gris.tu-darmstadt.de/research/captreal/projects

Bias Reduction

For all biased permutation algorithms with positive permutation matrices an iterative application of the algorithm reduces the bias. In our paper [3], we prove that the bias converges exponentially against 0 as illustrated in Figure 1 for k = 1 to 6 iterations.

The bias can further be reduced for butterfly networks by cyclic shifting of the array content between the algorithm iterations. However, the shifting offsets need to be selected carefully, since otherwise shifting might increase the bias.

Experimental Results

An implementation of the butterfly network on an NVIDIA GeForce GTX 480 compared to a Rand_Sort based on the CUDPP Radix Sort implementation:

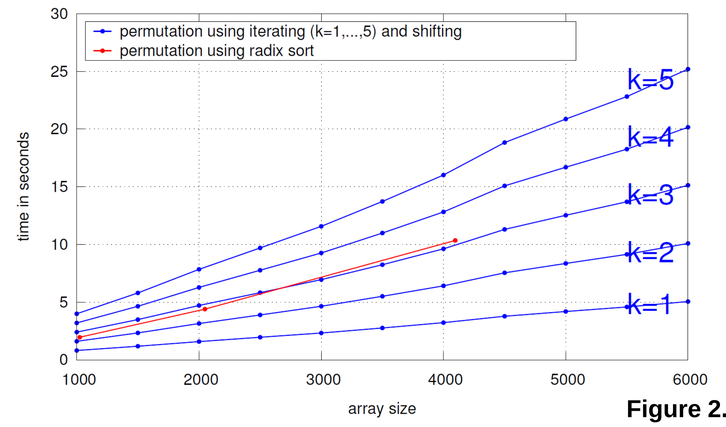


Figure 2.

Up to three bias reduction iterations the speed is competitive width the optimized CUDPP code. Rand_Sort stores, however, additional keys and does therefore not work in-place.

Conclusions

- **Permutation networks are typically well-suited for a GPU implementation.**
- **Bias is an issue for many permutation algorithms.**
- **Bias can be reduced at an exponential rate by repeatedly applying the permutation algorithm.**
- **Cyclic shifting with correct offset yields a further improvement for the butterfly network.**
- **Bias may be tolerable in a practical application (e.g., phylogenetic reconstruction) and can be traded off against computation speed (i.e., the number of iterations).**