Large-Scale Genome-Wide Association Studies on a GPU Cluster Using a CUDA-Accelerated PGAS Programming Model

Jorge González-Domínguez*, Jan C. Kässens, Lars Wienbrandt, Bertil Schmidt

Abstract Detecting epistasis, such as 2-SNP interactions, in Genome-Wide Association Studies (GWAS) is an important but time consuming operation. Consequently, GPUs have already been used to accelerate these studies, reducing the runtime for moderately-sized datasets to less than one hour. However, single-GPU approaches cannot perform large-scale GWAS in reasonable time. In this work we present multiEpistSearch, a tool to detect epistasis that works on GPU clusters. While CUDA is used for parallelization within each GPU, the workload distribution among GPUs is performed with Unified Parallel C++ (UPC++), a novel extension of C++ that follows the Partitioned Global Address Space (PGAS) model. multiEpistSearch is able to analyze large-scale datasets with 5M SNPs from 10K individuals in less than 3 hours using 24 NVIDIA GTX Titans.

Keywords PGAS, CUDA, GPU, UPC++, Bioinformatics

1 Introduction

Modern high-throughput technologies are able to gather information of millions of SNPs from thousands of individuals for GWAS. The most common phenotype classification is a binary trait, i.e. the presence (case) or absence (control) of an associated disease. 2-SNPs analyses try to find pairs of SNPs whose joint genotype frequencies show a statistically significant difference between cases and controls which potentially explains the effect of the genetic variation leading to disease (epistasis). Computing epistasis is highly time-consuming due to the large number of pairwise tests to be calculated. Targeting this problem with High Performance Computing (HPC) architectures can help to speedup the process. Parallel codes exist to detect epistasis [1, 2]. The best of these approaches are able to reduce the time to analyze a moderately-sized dataset from several days on a traditional CPU [3] to around one hour. Nevertheless, up to our knowledge, none of them is able to perform the analysis of datasets that contain millions of SNPs in acceptable time.

In this paper we present multiEpistSearch, a tool to detect epistasis on a GPU cluster. Our approach employs a hybrid implementation with CUDA for intra-GPU parallelism and Unified Parallel C++ (UPC++) [4] to distribute data among different GPUs. UPC++ is a novel extension of C++ that combines the advantages of both PGAS and Object Oriented paradigms. Up to our knowledge, this is the first hybrid implementation of an application using CUDA and UPC++, as most of scientific codes for GPU clusters employ MPI for inter-GPU parallelism.

*Corresponding author:
Jorge González-Domínguez, Parallel and Distributed Architectures Group, Johannes Gutenberg University, Staudingerweg 9 55128 Mainz, Germany. Email: j.gonzalez@uni-mainz.de

2 Hybrid Implementation

Our GWAS tool works with datasets containing information about a large number of biallelic genetic markers from many individuals. For each marker (SNP) there are three genotypes numerically represented as {0,1,2}. Each individual is further characterized as case or control, depending on the presence or absence of an associated disease. Two SNPs present epistasis or interaction if their combination discriminates between cases and controls significantly better than discrimination using each SNP individually. The number of SNPs and individuals is denoted as M and N, respectively.

Following the same approach as in [2], three filters (KSASA, KSA and log-linear) are applied to identify which of the $\frac{M(M-1)}{2}$ SNP-pairs present interaction. All of them use a 3x3x2 contingency table per pair that needs to be calculated from the N individual genotype pair values before the filters are applied. Our parallel implementation creates one UPC++ process per GPU and distributes the workload among them. Each GPU analyzes whether different SNP-pairs present epistasis through a CUDA kernel. In order to balance the memory usage per node, the biallelic information of the SNPs is distributed in blocks in a round-robin way, where the user can specify the number of blocks per process. This information is stored in shared memory so it can be directly accessed by all processes. A similar data distribution has been used in our UPC++ implementation for multicore clusters (see [5] for more details). When adapting this distribution to the hybrid CUDA&UPC++ implementation the GPUs may need information stored in remote memory to analyze the SNP-pairs. In this scenario the UPC++ process associated to that GPU must access the corresponding

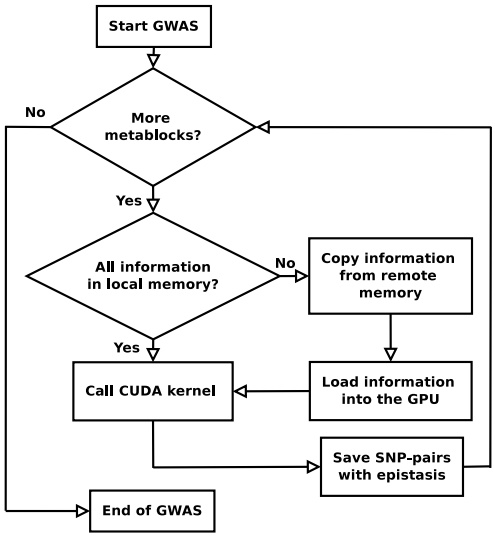

Figure 1: Procedure within each UPC++ process.

remote data and load it into the GPU before calling the kernel.

The flowchart in Figure 1 summarizes the behavior of each UPC++ process. The term "metablock" describes each block of SNP-pairs that must be computed; i.e. all possible combinations with the blocks of SNPs. In an example with 3 processes where the biallelic data is distributed using 2 blocks per process (6 blocks in total) there are 21 "metablocks" of SNP-pairs. For each associated metablock, if necessary, the process initially reads the biallelic information from remote memory and loads it into the GPU memory. Then, it launches the CUDA kernel so that the GPU searches for epistasis in all the SNP-pairs within the metablock. Thanks to the shared global memory space and the one-sided communications available in PGAS, remote copies can be performed without synchronization with the owner. Moreover, the execution of the CUDA kernel and the copy of the information needed to compute the next metablock are overlapped using asynchronous communication.

We have developed two approaches to distribute the metablocks among the processes and GPUs. First, a static distribution where the metablocks are initially associated to processes. The workload is balanced (by analyzing similar number of metablocks per GPU). The number of copies from remote memory is also minimized. Our distribution guarantees that metablocks are computed by processes that already have at least one of the necessary information blocks in their local memory. Second, an on-demand distribution has been implemented, where metablocks are not initially assigned to certain UPC++ processes. A table of UPC++ locks, with one open lock per metablock, is created in global memory (accessible by all processes). Every time one kernel call finishes, the process accesses this shared table to know which metablocks have not been or are not being computed at the same time by other GPUs; i.e. those metablocks whose lock is still open. Once one idle process finds a metablock to compute, it closes the associated lock.

To increase the efficiency of the UPC++ code, a set of optimization techniques has been applied, mostly focused on minimizing the communication cost. The utilized CUDA kernel is similar to the one explained in [2], but it includes two optimizations: 1) a reorder of the biallelic information before loading it to the GPU to improve coalescence; 2) exploitation of fast shared memory by the threads of the same thread block.

3 Experimental evaluation

Two different test systems have been employed in the experimental evaluation. (1) 8 nodes of the `Mogon` supercomputer, installed at the JGU Mainz, connected through a QDR InfiniBand network. Each node contains 3 NVIDIA GTX Titan GPUs (i.e. 24 GPUs in total). (2) 12 nodes of `Pluton`, installed at the University of A Coruña, connected through a Gigabit Ethernet network. 8 of the nodes contain one NVIDIA Tesla K20m and 4 nodes contain two NVIDIA Tesla 2050.

We have used a real-world dataset obtained from the Wellcome Trust Case-Control Consortium (WTCCC) [6] consisting of 3,004 controls and 2,005 cases genotyped at 500,568 SNPs. In order to provide a fair comparison, the influence of the number of blocks per process is removed by always showing the result for the best block size for each scenario. multiEpistSearch provides an autotuning option in order to automatically identify the best value for this parameter. The autotuning process uses information about the runtime of previous executions.

Figure 2 compares the runtime of the distributions described in Section 2 for a varying number of GPUs. The top graph shows that static distribution is better for clusters with the same type of GPUs, with parallel efficiency over 95%. When all the GPUs are similar, the on-demand version tends to equally distribute the workload. Thus, the workload distribution is the same for both approaches but the accesses to the locks present in the on-demand version lead to a performance overhead that makes it less efficient. However, results for the on-demand distribution are better on the cluster with heterogeneous GPUs, even though it includes the overhead of working with locks; e.g. it is 15% better when using all the GPUs. While in the static distribution all GPUs analyze a similar number of SNP-pairs, the on-demand approach distributes the workload so that the faster GPUs (Tesla K20m) compute more pairs than the slower ones (Tesla 2050).

Table 1 gathers the runtime of multiEpistSearch and other approaches when analyzing the WTCC dataset.

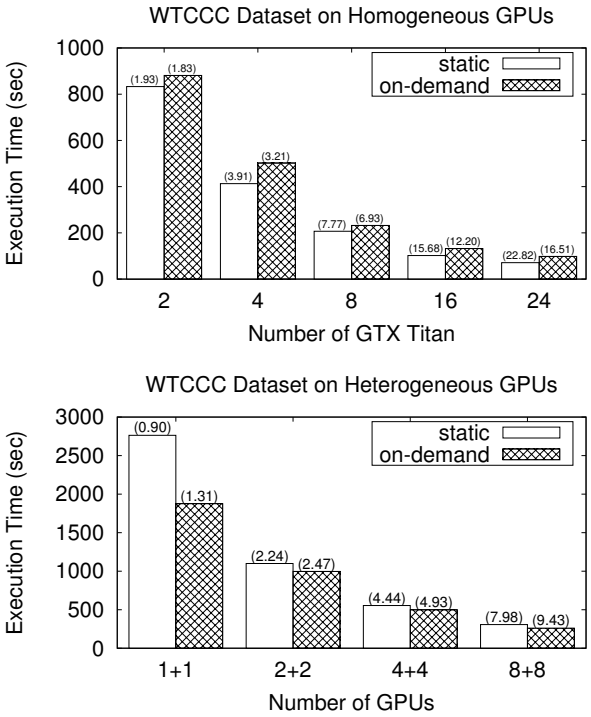

Figure 2: Performance comparison of the two multiEpistSearch workload distribution schemes. The labels indicate the speedup compared to multiEpistSearch executed on a single GTX Titan (top) and on one Tesla K20m (bottom).

Regarding the related work, we have executed the fastest publicly available tool (GBOOST [1]) utilizing only one GPU, as it does not support computation for multiple GPUs. Finally, we have estimated the execution time for BOOST [3] on an Intel Core i7 by analyzing a smaller simulated dataset and asssuming quadratic increase of time with the number of SNPs. The experimental results show that our CUDA implementation is 2.78 times faster than GBOOST using the same hardware (one GTX Titan). An additional advantage of our tool is the multi-GPU support, which allows us to obtain a speedup of 54.93 using 24 GTX Titans. According to the estimation of the sequential CPU-based BOOST, multiEpistSearch obtains speedups of more than 373 and 8,500 against a 3GHz CPU using 1 and 24 GTX Titans, respectively. Finally, thanks to the on-demand approach, the runtime on `Pluton` is reduced from 5 minutes and 40 seconds using only 8 Tesla K20m to 4 minutes and 20 seconds on the whole cluster, even though the Tesla 2050 is around 1.5 times slower than the Tesla K20m.

A simulated dataset with 5M SNPs and 10K samples has also been analyzed by multiEpistSearch in 2 hours and 45 minutes using 24 GTX Titans. A similar execution using only 1 GPU would need several days, which demonstrates the need of multi-GPU GWAS tools in order to perform large-scale GWAS analyses in reasonable time. Such a large dataset could not be analyzed by GBOOST due to out-of-bounds problems in the internal arrays.

Table 1: Runtime of different designs when looking for epistasis in the WTCCC dataset. The results for CPU BOOST were estimated from a smaller dataset (*)

Design	Architecture	Runtime
multiEpistSearch	24 GTX Titan	1 m 11 s
multiEpistSearch	8 Tesla K20m + 8 2050	4 m 20 s
multiEpistSearch	8 Tesla K20m	5 m 40 s
multiEpistSearch	8 Tesla 2050	10 m 12 s
multiEpistSearch	1 GTX Titan	27 m
GBOOST	1 GTX Titan	1 h 15 m
BOOST*	Intel Core i7	7 d

4 Conclusions

We have presented multiEpistSearch, a tool that exhaustively measures the interaction of all SNP-pairs of a GWAS dataset on GPU clusters using CUDA and UPC++, a new PGAS extension of C++. Static and on-demand approaches have been implemented for the workload distribution among GPUs. They have been tested on two different GPU clusters. The static distribution, that initially assigns metablocks to GPUs, obtains the best performance on `Mogon`, a cluster with 24 GTX Titans. The on-demand distribution, where the workload is dynamically assigned to the GPUs during execution time, is more suitable for clusters such as `Pluton`, where there are different types of GPUs. Our evaluation shows that multiEpistSearch is able to exhaustively look for epistasis in large-scale SNP datasets in feasible time. For instance, it takes less than 3 hours to analyze a dataset with 5M SNPs and 10K individuals on `Mogon`.

References

[1] Yung LS, Yang C, Wan X, et al. GBOOST: A GPU-Based Tool for Detecting Gene-Gene Interactions in Genome-Wide Case Control Studies. *Bioinform.*, 27(9):1309–1310, 2011.

[2] González-Domínguez J, Schmidt B, Wienbrandt L, et al. Hybrid CPU/GPU Acceleration of Detection of 2-SNP Epistatic Interactions in GWAS. In *Proc. 15th Intl. Eur. Conf. on Par. and Dist. Comp. (Euro-Par'14)*, 2014.

[3] Wan X, Yang C, Yang Q, et al. BOOST: a Fast Approach to Detecting Gene-Gene Interactions in Genome-Wide Case-Control Studies. *Am. J. Hum. Genet.*, 87(3):325–340, 2010.

[4] Zheng Y, Kamil A, Driscoll M, et al. UPC++: a PGAS Extension for C++. In *Proc. 28th IEEE Intl. Par. and Dist. Proc. Symp. (IPDPS'14)*, 2014.

[5] Kässens JC, González-Domínguez J, Wienbrandt L, Schmidt B. UPC++ for Bioinformatics: A Case Study Using Genome-Wide Association Studies. In *Proc. 15th IEEE Intl. Conf. on Cluster Comp. (Cluster'14)*, 2014.

[6] The Wellcome Trust Case Control Consortium. Genome-Wide Association Study of 14,000 Cases of Seven Common Diseases and 3,000 Shared Controls. *Nature*, 447(7145):661–78, 2007.