Brief Announcement: MIC++: Accelerating Maximal Information Coefficient Calculation with GPUs and FPGAs

Chao Wang, Xi Li, Aili Wang, and Xuehai Zhou
School of Computer Science, University of Science and Technology of China, Hefei, China
School of Software Engineering, University of Science and Technology of China, Suzhou, China
{cswang,llxx,wangal,xhzhou}@ustc.edu.cn

ABSTRACT

To discover relationships and associations between pairs of variables in large data sets have become one of the most significant challenges for bioinformatics scientists. To tackle this problem, maximal information coefficient (MIC) is widely applied as a measure of the linear or non-linear association between two variables. To improve the performance of MIC calculation, in this work we present MIC++, a parallel approach based on the heterogeneous accelerators including Graphic Processing Unit (GPU) and Field Programmable Gate Array (FPGA) engines, focusing on both coarse-grained and fine-grained parallelism. As the evaluation of MIC++, we have demonstrated the performance on the state-of-the-art GPU accelerators and the FPGA-based accelerators. Preliminary estimated results show that the proposed parallel implementation can significantly achieve more than 6X-14X speedup using GPU, and 4X-13X using FPGA-based accelerators.

Keywords

MIC; Accelerator; GPU; FPGA

1. INTRODUCTION AND CONTRIBUTION

Large scale data analysis is posing significant challenges to state-of-the-art bioinformatics computing machines and technologies. The problem of identifying whether two or more variables are independent or have some kinds of associations (functional or non-functional) is called the *Detecting Association Problem* [1]. This problem is becoming g increasingly important in different fields growing with the explosive data scale. To tackle this issue, David N.Reshef [2] has established the generality of maximal information coefficient (MIC) calculation method, showing its equitability on functional relationships through simulations, and observe the intuitively equitable behavior on more general associations. However, this MIC computation algorithm is entirely computational and data intensive so that it is not feasible to employ traditional machine on large data sets. Billions of samples and thousands of variables make the computing impossible and time-consuming. To our best knowledge, although there has been some preliminary research on the acceleration of MIC calculation using cloud-based distributed system [4] or GPU-based accelerators [3], so far no hardware accelerating techniques like using FPGA-based platforms have been implemented.

To solve this problem, in this paper, we propose MIC++, an accelerator framework based on heterogeneous GPU and FPGA hardware accelerators. We extended the current sequential algorithm which is based on dynamic programming, and can achieve higher performance without any correctness penalty and insignificant overheads. We claim following contributions of this work:

- In this work, we analyze the behavior of the MIC calculation with four steps and propose a parallelized implementation of the MIC calculation with hierarchical coarse-grained and fine-grained parallelisms, named MIC++. We can use parallelized MIC++ algorithm to different accelerators like GPU and FPGA based architectures.
- We implement the parallel MIC++ approach on NVIDIA GTX 750 GPU, and Xilinx Virtex-7 NetFPGA SUME real hardware platforms. Experimental results demonstrate the MIC++ accelerator can achieve 6X-14X speedup with ignorable hardware cost. Finally, we believe the MIC accelerator can be utilized in a wider range such as data processing in machine learning areas.

2. MIC PARALLELIZATION

Fig.1 illustrates the parallel MIC++ architecture. In particular, Fig.1 (a) demonstrates the coarse-grained parallelism on accelerator based cluster architectures, while Fig.1 (b) and Fig.1 (c) illustrate the fine-grained parallelism on accelerating threads.

2.1 Coarse-grained parallelism

Regarding the MIC calculation process, Each element of MICM (MIC matrix) is defined as $I(x, y)$, which stands for the maximum entropy when partitioning all the two-variable pairs in x rows and y columns. The computing process of MICM starts by sorting all pairs in ascendant order by the Y value. Thus, each column of MICM uses the same initial partition. Then, we use accelerator based architecture to compute MICM in parallel. In the coarse-grained parallelism step, we only focus on the sorting of the input data set D, while the fine-grained parallelism step handles actual accelerating for computing MICM. When processing large data sets, sorting steps (step 1 and 3) take intolerable long time when using serial sorting algorithms such as QuickSort or MergeSort. By contrast, Bitonic sorting networks have been introduced to run sorting steps in parallel. Bitonic sorting networks have been proved as efficient measures for GPU computing not only for its time complexity but also for its data-dependency, meaning that the time consumption of sorting depends on the size of the data set but not the

SPAA '16 July 11-13, 2016, Pacific Grove, CA, USA

ACM ISBN 978-1-4503-4210-0/16/07.

DOI: http://dx.doi.org/10.1145/2935764.2935804

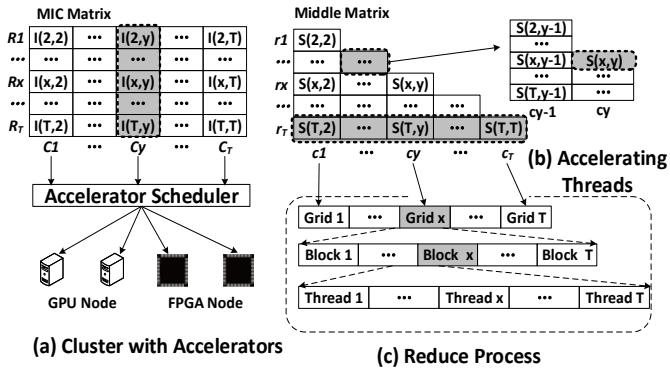

Figure 1: Architecture of the parallel algorithm, the MIC matrix is first processed by GPU cluster scheduler (in a), and then organized into a Middle Matrix, which will be operated in CUDA threads (b and c).

original sequence of it. The time complexity of Bitonic sorting is $O(log_2(N))$, and it is a comparator based sorting network. Thus, no additional memory is required as in Quicksort. The other two steps of initial partition are not suitable for coarse-grained parallelization due to the inter-task dependencies between them. Finally, it can be concluded that the steps 1 and 3 run on accelerators while steps 2 and 4 run on CPU. Though data transmission takes place between CPU and accelerators in Bitonic sorting networks, the overall performance is still significantly improved than the serial algorithms. Besides, when the CPU is executing step 2, execution of step 3 on GPU can be fully overlapped, reducing the overall time consumption of the implementation of the algorithm. When the coarse-grained parallelism finishes, a series of two-dimensional vectors (TDVs) are stored. The TDVs contain the partitions of each (x, y) pair, which will be used to compute the MIC in the fine-grained parallelism.

2.2 Fine-grained parallelism

In the coarse-grained parallelism, the TDV that holds the necessary data for computing a certain column of MICM is generated. In this section, we design a parallel algorithm to accelerate the computing process at each node. Fig.1 (b) presents the computing process of each column of MICM. The left part of Fig.1 (b) is a middle matrix (MM) that temporarily stores the intermediate results. Each element of MM is $S(x, y)$, which refers to the maximum entropy when partitioning the first x columns of TDV into y columns. MM is a lower triangular matrix, where the last row finally stores the results of the column of MICM, e.g. the data of rT in Fig.1(b) is cy in Fig.1(a) after transposition, both rT and cy are surrounded with dashed line. The right part of Fig.1(b) demonstrates the data dependencies when computing an element in MM. Each S(x,y) only relies on the elements whose x-axis index is no more than x of the $(y-1)$ column, as shown in the right part of Fig.1(b), the dark elements will affect the value of $S(x, y)$. When computing $S(x, y)$, each element produces a new entropy and then $S(x, y)$ selects the maximum one of the final value. Based on the above analysis, accelerator threads are organized at three levels, Grid level, Block level, and Thread level, respectively, as illustrated in Fig.1 (c).

Grid level: Each column of MM is calculated simultaneously. As adjacent columns have data dependencies, thus each column is organized as a grid in the middleware (like CUDA), a synchronization function is inserted between adjacent grids. The k^{th} column is corresponding to the k^{th} grid.

Block level: Different elements of a column have different data dependencies as described above. To efficiently use the shared memory of each block, each separate component is organized as a block.

Thread level: As shown in the right part of Fig.1 (b), the computation of $S(x, y)$ has $(x-1)$ separate intermediate calculations, and finally, the peak value is selected. So the u^{th} block has $(u-1)$ threads, a parallel reduce process is applied to find the maximum value and its position.

If thread index exceeds the maximum number allowed by accelerators, the associating block is then automatically divided into smaller subblocks. Accordingly, the reduce process is also divided into subsequences, and a merge process will be handled after all the sub-tasks are finished.

2.3 Preliminary results and analysis

To evaluate the effectiveness of the MIC++ accelerators using GPU and FPGA-based accelerators, we compare it against a CPU with 256-bit SIMD (Intel Core i5 4460 3.3Hz, 8GB memory) over different data scales. We take a modern GPU card (NVIDIA GTX 750, 1.04 TFlops peak, 2GB GDDR5, 80.2 GB/s memory bandwidth, 28nm technology, CUDA SDK5.5) as the baseline. We use the MovieLen1M Benchmark to evaluate the speedup and hardware cost. 1000 movies are selected, and each movie is represented by a 6040*32bit vector.

We have evaluated four case studies for users and items in the benchmark, with specific parts of the application offloaded to accelerator. Preliminary estimated results show that the GPU accelerator achieves an average kernel speedup of 6X-14X on the SIMD CPU implementation. By contrast, the FPGA based hardware accelerator achieves 4X-13X in estimation, which is slightly lower than the GPU-based accelerators. By the further data analysis, it can be concluded that the GPU can access the data with up to 80.2 GB/s memory bandwidth while the FPGA accelerators only have a peak bandwidth at 14.9GB/s. The reason that GPU and FPGA are achieving similar speedup is that although FPGA is faster than GPU accelerators, the bandwidth of the FPGA development board limits the peak speedup. Regarding the hardware cost, simulation shows that the area utilization for each FPGA-based accelerator is less than 5%, which means a modern FPGA platform can accommodate about 20 accelerators. In that case, the speedup of a single FPGA chip can achieve 260X kernel speedup against state-of-the-art processors. Meanwhile, the power consumption for a single accelerator is only 210mW, which is much more energy-efficient than CPU and GPU-based parallel approaches. Therefore, GPU and FPGA-based accelerators have great potential to accelerate MIC application.

Acknowledgment. This work was supported by the NSFC No. 61379040, 61272131, Jiangsu Provincial NSF SBK201240198, Anhui Provincial NSF 1608085QF128, State Key Lab of Computer Architecture, and CCF-Tencent Open Research Fund. Xi Li and Aili Wang are the corresponding authors of this paper.

3. REFERENCES

[1] Karpinets T.V., et al., Analyzing large biological datasets with association networks. Nucleic Acids Res, 2012. 40(17):e131.

[2] Reshef, D.N., et al., Detecting Novel Associations in Large Data Sets. Science, 2011. 334: p. 1518-1524.

[3] Tang, D., et al., RapidMic: Rapid Computation of the Maximal Information Coefficient. Evolutionary Bioinformatics, 2014.

[4] Wang, C., et al., Accelerating Computation of Large Biological Datasets using MapReduce Framework, IEEE/ACM Trans. on Computational Biology and Bioinformatics, 2016.