35th Annual International Conference of the IEEE EMBS
Osaka, Japan, 3 - 7 July, 2013

GPU technology as a platform for accelerating local complexity analysis of protein sequences

Agathoklis Papadopoulos, *Student Member, IEEE,* Ioannis Kirmitzoglou, Vasilis J. Promponas, Theocharis Theocharides, *Senior Member, IEEE*

Abstract— **The use of GPGPU programming paradigm (running CUDA-enabled algorithms on GPU cards) in Bioinformatics showed promising results [1]. As such a similar approach can be used to speedup other algorithms such as CAST, a popular tool used for masking low-complexity regions (LCRs) in protein sequences [2] with increased sensitivity. We developed and implemented a CUDA-enabled version (GPU_CAST) of the multi-threaded version of CAST software first presented in [3] and optimized in [4]. The proposed software implementation uses the nVIDIA CUDA libraries and the GPGPU programming paradigm to take advantage of the inherent parallel characteristics of the CAST algorithm to execute the calculations on the GPU card of the host computer system. The GPU-based implementation presented in this work, is compared against the multi-threaded, multi-core optimized version of CAST [4] and yielded speedups of 5x-10x for large protein sequence datasets.**

I. INTRODUCTION

Proteomics – the study of the proteins of biological species (including human) – has revolutionized the way biological research is currently performed. As huge amounts of sequence data are currently being produced worldwide at an increasing pace, extensive downstream computational analysis is required. Typical computational pipelines for proteomics feature a computationally intensive sequence comparison component; this is justified by the empirical observation that genes and proteins with similar sequences usually perform similar functions. Therefore, sequence similarity search serves for inferring functional and structural analogy for biological macromolecules [5] [6].

Traditionally, such complex algorithms have been implemented on high-performance computing clusters and multiprocessor/multicore systems [7]. There has been a tremendous amount of effort in designing efficient systems that can take advantage of the inherent parallelism opportunities of such algorithms; however, the specialized nature of algorithms targeting bioinformatics, limits the number of end-users for such platforms, thus the costs of possible custom hardware solutions tend to be extremely high. As such, alternative technologies that can better balance the cost and performance constraints can be more efficient for targeting the bioinformatics research communities.

Emerging technologies such as general purpose computing on graphics processing units (GPGPU) are prime candidates for improving performance of bioinformatics applications. GPGPU implementations have been recently developed to facilitate the BLAST (Basic Local Alignment Search Tool) algorithm and the resulting GPU-BLAST exhibited a 4x speedup [1]. GPGPU-enabled GPU cards are composed of a massive number of processing elements designed to host multiple threads that execute the same instructions over different data in parallel. Using this programming paradigm, normally we can accelerate data-intensive applications, such as CAST.

In this paper, we present a first attempt to accelerate the performance of CAST using GPGPU programming paradigm. The GPGPU-enabled version developed in this work is compared against the optimized multi-threaded version of CAST presented in earlier work in order to evaluate the proposed implementation. The paper is partitioned as follows: Section II provides background information on CAST algorithm; Sections III and IV discuss the GPGPU programming paradigm and related work in bioinformatics; Section V and VI describe and evaluate respectively the proposed implementation, and Section VII concludes the paper.

II. BACKGROUND

Comparison of biological macromolecular sequences (i.e. protein, DNA, RNA) is a key computational task for the characterization of genes and proteins massively identified through genome projects. This process is facilitated by rapid and sensitive tools, such as the BLAST heuristic algorithm [8]. In principle, high sequence similarity between a query sequence and an annotated database entry, can be used to obtain a reliable function prediction for the query. A pitfall of this approach may originate from the presence of LCRs, which are source of unnaturally high scores for compositionally biased sequence pairs that may otherwise be biologically unrelated. A way to overcome this situation is

Agathoklis Papadopoulos is with KIOS Research Center ,Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus (corresponding author; phone: +357-22893461; e-mail: papadopoulos.agathoklis@ucy.ac.cy).

Ioannis Kirmitzoglou and Vasilis J. Promponas are with Bioinformatics Research Laboratory, Department of Biological Sciences, University of Cyprus, Nicosia, Cyprus (e-mail: vprobon@ucy.ac.cy).

Theocharis Theocharides is with KIOS Research Center ,Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus (e-mail: ttheocharides@ucy.ac.cy).

masking LCRs to significantly improve the reliability of homology detection and the quality of function prediction.

CAST [2] is an iterative method (see Fig.1 for details) for detecting and masking LCRs in protein sequences, resulting in more sensitive database searches compared to SEG [9]. It is formulated as a clever modification of the Smith-Waterman local sequence alignment algorithm. CAST identifies local similarities of a query sequence against degenerate sequences corresponding to the 20 possible amino acid homopolymers of arbitrary length without permitting for gaps, using a suitable substitution matrix. Therefore, on top of the detection of LCRs, CAST is also able to associate LCRs with a specific residue type – thus enabling selective masking. CAST iteratively performs LCR detection and masking steps to prevent unnecessary masking due to cross-dependencies between amino acid residue types, provided the detected similarities exceed an empirically defined threshold value (T). With the use of the BLOSUM62 substitution matrix, the optimal value T = 40 is used. In practice, a variant of BLOSUM62 serves as the default scoring matrix: the scores of each residue type against the neutral type 'X', are computed as the mean value of the amino acid substitution scores for the respective residue type.

The algorithm shown in Fig.1 receives as input a protein sequence, and, searches for the LCR candidates (highest scoring segments- HSS) of each natural amino acid type. It then selects the HSS with the maximum score, and if that score is less than the threshold T, it ends outputting discovered LCRs; otherwise, it replaces each occurrence of the max scoring residue type in the highest scoring segment region with an 'X' (i.e. a neutral amino acid) and iterates through the updated sequence. For each discovered LCR its residue type, the sequential position (start and end) and computed score are reported. Further details of the algorithm can be found in [2].

III. GPGPU Programming Paradigm

GPGPU (General Purpose computing on Graphic Processing Units) is a programming model that enables the use of the processing capabilities of the graphic cards present in modern-day computers for executing code that traditionally was executed on CPUs. However, the processing elements of GPU cards are designed specifically for graphics and thus are very restrictive in operations and programming. As such, GPUs are only effective for problems that can be solved using stream processing and their hardware can be used only when using specialized API such as OpenCL and nVidia CUDA libraries.

CUDA is the programming model developed by nVidia and implemented by the company's graphic cards [10]. CUDA libraries give access to a virtual instruction set for industry-standard programming languages (such as C/C++ and Fortran) that allows software developers to efficiently map applications to be executed on GPU cards. A typical CUDA processing flow is: a) Copy data from CPU memory to GPU memory; 2) CPU instructs the GPU to start processing the desired code - *kernel*; 3) GPU executes the kernel code *in parallel*; 4) Copy the results from GPU memory back to CPU memory.

Using CUDA has limitations such as: unavoidable memory transfer overheads between CPU and GPU memories, limited memory resources per GPU core and adjustments needed for double floating number operations. However, provides a tool that enables developers to harness the processing capabilities of the hundreds cores present on nVidia's graphic cards.

```
Algorithm CAST
Input: A protein sequence S
Output: The sequence masked for LCRs

residues  ← (A, C, D, ..., Y)
hscore    ← ( 0, 0, 0, ...0)
from      ← 0
to        ← 0
neutral   ← X

do
       for each rest in residues
           (hscore[res], from, to) ← detectBias(S, res)

       hscoreMAX ← max(hscore[A], hscore[C], .., hscore[Y])
       lcrType ← residues[ argmax(hscore[A], hscore[C], .., hscore[Y])

       if (hscoreMAX >=Threshold)
             S ← mask(S, lcrType, from, to)

while ((hscoreMAX>=Threshold)

function detectBias(Sequence: S, Residue: r)
       (maxscore, from, to) ← alignSmithWaterman(S, poly-residue)
       return (maxscore, from, to)

function mask(Sequence: S, Residue: lcrType, Start: from, End: to)
       for pos in  (from .. to)
             if (S[pos] equals lcrType)
                    S[pos] ← neutral
       return (S)
```

Figure 1. Pseudocode for the CAST algorithm. LCR detection is performed by iteratively comparing input S with degenerate homopolymers (poly-residues) of the 20 naturaly occuring amino acids. For further details see [2].

IV. Related Work

GPGPU programming is extensively used in bioinformatics since the beginning [10], as most of the applications used in the field are computationally complex and execute of massive datasets - characteristics that allow extensive utilization of the GPGPU programming capabilities. While not all algorithms perform well when executed on GPUs [11], previous attempts on developing CUDA-enabled implementations for bioinformatics applications showed promising results. The GPU-based implementation of Smith-Waterman [12] showed speedups of 2x-30x over any previous implementation. GPU-BLAST [1] showed speedups between 3x-4x for all test cases. GPU technology used in Markov clustering showed speedups of 3x-8x [13]. The pairwise local sequence alignment program proposed in [14] showed 10-fold speedup as well. Taking the above non-exhaustive list into consideration, we can indeed deduct that algorithms used in bioinformatics applications are prime candidates for developing GPU-based versions.

V. Proposed Implementation

GPU_CAST follows a similar workflow with the optimized multi-threaded software presented in [4]. However, some fundamental differences do exist in order to allow higher performance when executed on the GPU.

The sequence data are fetched in batches of predefined size (*N=50*, due to GPU shared memory size limitations). For each batch, a kernel execution is prepared by the host CPU. Firstly, the batch's data is transferred from the main memory to the GPU memory and stored in the GPU's global memory. Then, the kernel is invoked. The GPU_CAST calculation is

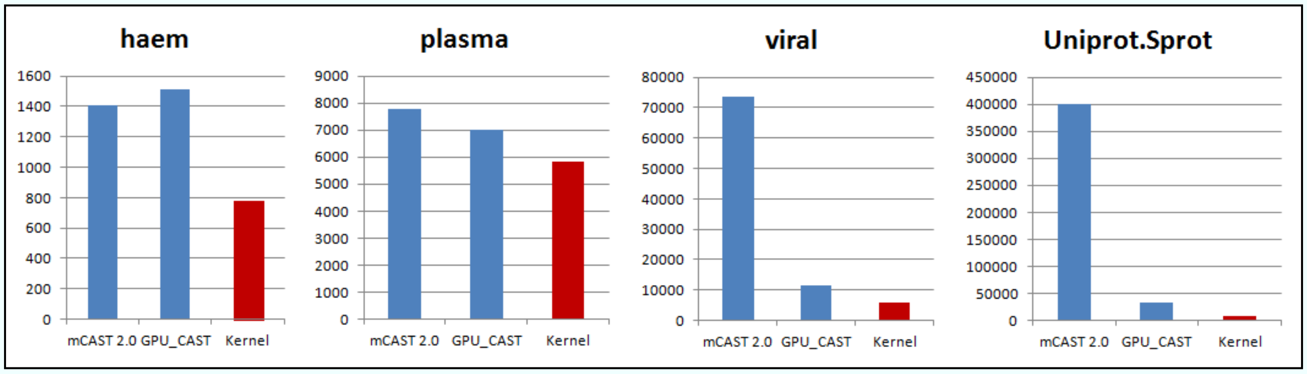

Figure 2. Execution Times (ms) of mCAST 2.0 and GPU_CAST (Total and Actual Kernel execution time).

executed in *20*N* threads on the GPU card's cores. Each set of 20 threads, corresponds to the CAST execution over the homopolymers for each one of the N sequences processed in parallel, thus making efficient use of the GPU card's processing elements. The data resulting from each set of threads, will then be synchronized in the GPU's shared memory and used by thread 0 of the set to calculate the current iteration's LCR.

Upon kernel completion - all threads terminated - the host CPU will regain control and mask each sequence in the batch based on the kernel results. This process iterates if needed; the kernel is invoked on the updated batch in order to discover the remaining LCRs of its sequences. Next, the host CPU proceeds in fetch the next batch of *N* sequences. The GPU_CAST developed in this work, was programmed to maintain the same input/output interface as the original tool while ensuring it produces identical results.

VI. Evaluation and Results

The proposed GPU_CAST tool was evaluated against the optimized multi-threaded version – mCAST 2.0 – proposed in [4]. We used a computer system having: Intel Core i7 3960X @ 3.3GHz, 32GB RAM running Microsoft Windows 7, equipped with a nVidia GeForce GTX690 graphic card. We compared the execution times, while executing the benchmark datasets shown in Table I.

A. Proteomic Benchmarks

To compare the proposed GPU-based implementation against the *mCAST2.0* software, we used a number of datasets stemming from four actual protein sequence databases. The first database belongs to *Haemophilus influenza* bacteria (*haem database*) and the second consists of protein sequences from the malaria parasite genome *Plasmodium falciparum* (p.falciparum database). The third and fourth databases are a reference viral database and a unified protein database from NCBI used for benchmarking bioinformatics applications (*viral.1.protein*, *uniprot.sprot*). Table I lists all the test databases used during the evaluation process.

B. Results

The execution times of mCAST 2.0 and GPU_CAST are shown in Fig.2 expressed in milliseconds. The GPU_CAST when executed on the nVidia GeForce GTX690 graphic card shows no significant execution time differences for the relatively small *haem* and *p.falciparum* (plasma) datasets. The GPGPU implementation paradigm shows its benefits when GPU_CAST executes over *Viral.1.Protein* (viral) and *Uniprot.sprot* datasets and speedups between 5x-11x are achieved. We can see that the CAST algorithm is accelerated when executed on the GPU card; however the speedup greatly depends of the dataset. For example the execution time is cut in tenth for *Uniprot.sprot* but for the smaller *haem* dataset GPU_CAST yields similar (yet slower) execution times as mCAST 2.0.

While GPU_CAST is proven to be more efficient for executing CAST algorithm over large datasets, a comparison between Fig.2's GPU_CAST and kernel columns, shows that a significant amount of time is consumed in tasks different that actually performing CAST (shown by the kernel column). As such, further analysis is required for identifying these overheads.

Executing software programs directly on the GPU card, the data has to be present on the GPU memory, as already discussed in Section III. As such, the data needed for calculating CAST - the information of the sequences in the dataset and the substitution matrix - must be transferred to the GPU memory before the kernel of CAST executes, and move the results from the GPU back to the host memory as well. As such, memory transfers between the CPU and the GPU have to be evaluated as potential performance bottlenecks, in order to determine their actual impact on GPU_CAST execution times. Fig.3 shows the time consumed in CUDA API-related tasks such as memory transfers and kernel execution.

Memory transfer overheads sum up to 51% of the CUDA tasks for *haem*, 31% for *viral.1.protein*, and 29% for *Uniprot.sprot* dataset respectively, while remain other 10% for *plasma* dataset. We can understand that the memory

TABLE I. PROTEIN SEQUENCE DATASETS

Dataset Name	# of sequences	Average length	% LCRs*
Haem	1,743	305.1	7.97 (0.64)
p.falciparum (plasma)	5,491	755.9	72.81 (14.42)
Viral.1.Protein (viral)	101,537	274.4	16.56 (2.05)
Uniprot.sprot	537,593	354.9	33.80 (0.16)

* % of sequences having at least one LCR (% of residues masked by CAST)

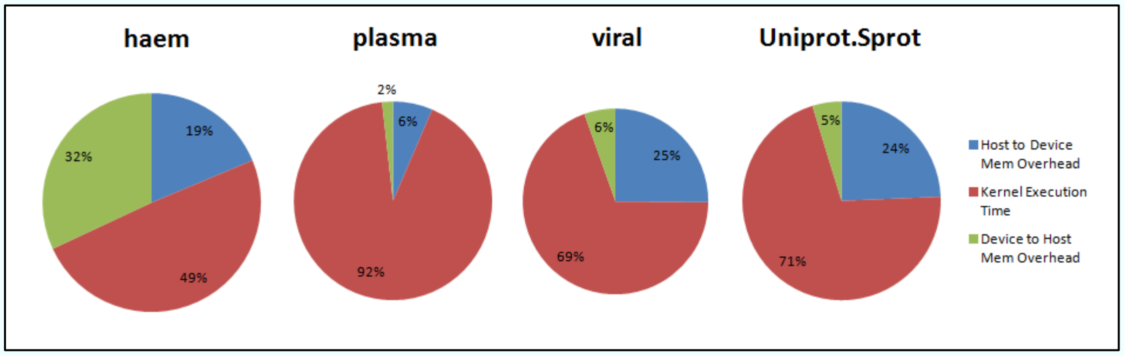

Figure 3. CUDA API Times (ms) Breakdown (Memory Transfer Overheads and Kernel Execution Times)

overheads are greatly affected by the size and the characteristics of the dataset. For instance, the low memory overheads for *plasma* dataset can be justified by the fact that more than 70% of the sequences in the dataset have at least one LCR (as shown in Table I), thus CAST kernel has to iterate more. The dynamic nature of CAST algorithm makes difficult to accurately predict memory overheads, however the average memory overheads of all the datasets is calculated to be 30% which is a significant portion of the overall execution time.

VII. Conclusion and Future Work

A GPU-based implementation of a popular bioinformatics algorithm is presented in this paper. The results suggests that speedups of 5x-10x over multi-threaded versions can be achieved when using GPU technology for CAST algorithm. Future work will explore optimization techniques for minimizing memory overheads which are proven to consume an average 30% of the execution time.

Acknowledgments

This work was co-funded by the European Regional Development Fund of the European Union and the Republic of Cyprus through the Cyprus Research Promotion Foundation programme [ΠΕΝΕΚ/0311/32].

References

[1] P. Vouzis and N. Sahinidis, “GPU-BLAST: Using graphics processors to accelerate protein sequence alignment,” *Bioinformatics,* vol. 27, pp. 182-188, 2010.

[2] V. J. Promponas et al., "CAST: an iterative algorithm for the complexity analysis of sequence tracts," *Bioinformatics,* vol. 16, no. 10, pp. 915-922, 2000.

[3] A. Papadopoulos, V. J. Promponas, T. Theocharides, "Towards systolic hardware acceleration for local complexity analysis of massive genomic data," in *ACM/IEEE Great Lakes Symposium on VLSI - GLVLSI 2012*, Salt Lake City, 2012.

[4] A. Papadopoulos, I. Kirmitzoglou, V. J. Promponas, T. Theocharides, "FPGA-based hardware acceleration for local complexity analysis of massive genomic data," *Integration, The VLSI Journal, Elsevier,* no. November, 2012.

[5] R. Nair, B. Rost, “Sequence conserved for subcellular localization,” *Protein Science,* vol. 11, no. 12, pp. 2836-2847, 2002.

[6] A. J. Enright, S. van Dongen, and C. A. Ouzounis, “An efficient algorithm for large-scale detection of protein families,” *Nucleic Acids Research,* vol. 30, no. 7, pp. 1575-1884, 2002.

[7] A. E. Darling, L. Carey, and W. Feng-chun, “The design, implementation, and evaluation of mpiBLAST,” *ClusterWorld,* pp. 13-15, 2003.

[8] S. F. Altschul et al, “Basic local alignment search tool,” *Journal of Molecular Biology,* vol. 215, no. 3, pp. 403-410, 1990.

[9] J. C. Wootton and S. Federhen, “Statistics of local complexity in amino acid sequences and sequence databases,” *Computers & Chemistry,* vol. 17, no. 2, pp. 149-163, 1993.

[10] J. Nickolls, "Nvidia GPU parallel computing architecture," in *IEEE Hot Chips, IEEE Technical Committee on Microprocessors and Microcomputers*, Stanford, 2007.

[11] J.M. Elble et al., "GPU computing with Kaczmarz's and other iterative algorithms for linear systems," *Parallel Computing,* vol. 36, no. 215-231, 2010.

[12] S. Manavski, G. Velle, "CUDA compatible GPU cards as efficient hardware acceleratos for Smith-Waterman sequence alignment," *BMC Bioinformatics,* vol. 9 (Suppl.2), no. S10, 2008.

[13] A. Bustamam, K. Burrage, N.A. Hamilton, "Fast Parallel Markov Clustering in Bioinformatics Using Massively Parallel Computing on GPU with CUDA and ELLPACK-R Sparse Format," *IEEE/ACM Transactions of Computational Biology and Bioinformatics,* vol. 9, no. 3, 2012.

[14] Michael C Schatz, Cole Trapnell, Arthur L Delcher, Amitabh Varshney, "High-throughput sequence alignment using Graphics Processing Units," *BMC Bioinformatics,* vol. 8, no. 474, 2007.