BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 60, No. 3, 2012
DOI: 10.2478/v10175-012-0062-1

G-PAS 2.0 – an improved version of protein alignment tool with an efficient backtracking routine on multiple GPUs

W. FROHMBERG[1], M. KIERZYNKA[1,3], J. BLAZEWICZ[1,2], and P. WOJCIECHOWSKI[1*]

[1] Institute of Computing Science, Poznań University of Technology, 3 Piotrowo St., 60-965 Poznań, Poland
[2] Institute of Bioorganic Chemistry, Polish Academy of Sciences, 12/14 Z. Noskowski St., 61-704 Poznań, Poland
[3] Poznań Supercomputing and Networking Center, 12/14 Z. Noskowski St., 61-704 Poznań, Poland

Abstract. Several highly efficient alignment tools have been released over the past few years, including those taking advantage of GPUs (Graphics Processing Units). *G-PAS* (GPU-based Pairwise Alignment Software) was one of them, however, with a couple of interesting features that made it unique. Nevertheless, in order to adapt it to a new computational architecture some changes had to be introduced. In this paper we present *G-PAS* 2.0 – a new version of the software for performing high-throughput alignment. Results show, that the new version is faster nearly by a fourth on the same hardware, reaching over 20 GCUPS (Giga Cell Updates Per Second).

Key words: pairwise alignment, GPU computing, alignment with backtracking procedure.

1. Introduction

1.1. Background. Pairwise alignment algorithms are of great importance in computational biology. Methods based on dynamic programming, despite their high computational complexity, have been widely used over a few decades due to their accuracy. However, in real-life applications constantly increasing number of biological sequences has started to cause serious difficulties.

In the past few years, modern computing architectures have not only shown their potential, but also have become a standard in many branches of science owing to their high computational power for a reasonable price [1–3]. Although many GPU-based applications have been released lately in the area of sequence alignment, e.g. [4–7], most of them address the problem of database scan and as such cannot be applied efficiently to all the problems like MSA, where each sequence has to be aligned with each other (cf. [8]), or DNA assembly (see e.g. [9] and [10]). Moreover, these problems apart from the alignment score require the alignment itself to be computed as well. Yet, computing the alignment is usually omitted in the majority of implementations. Therefore, in response to presented needs, we developed an efficient tool for pairwise alignment called *G-PAS* (previously known as gpu-pairAlign). Its key features included: optimizations for aligning every sequence with each other, backtracking routine performed in linear time and utilization of multiple GPUs. The method was thoroughly described in [11] and proved to be well adapted to many practical applications. However, with the advent of a new GPU architecture –Fermi[1], we decided to release a new version of our software – *G-PAS* 2.0. Computational tests prove its supremacy on new as well as old GPUs.

1.2. Methods Three basic algorithms for pairwise alignment of biological sequences can be found in the literature: Needleman-Wunsch [12] for computing global alignment (NWG), its modification for semiglobal alignment (NWS) and Smith-Waterman [13] for local alignment (SW). All these algorithms use the idea of dynamic programming and to some degree work similarly. Although *G-PAS* 2.0 concentrates only on these original methods, below we provide also a short discussion of other algorithms outlining their main advantages and drawbacks.

Chao et al. proposed in [15] to lower the computational costs by calculating only those cells in the dynamic programming matrix that lie in the relative proximity of its main diagonal. However, one obvious disadvantage of the method is that its results may be not optimal. Moreover, this approach cannot be applied to the local or semiglobal alignment.

Another improvement introduced to the aligning methods allows to deal with so called *profiles*, i.e. sequences that may have alternative residues at certain positions. In Gribskov et al. [16, 17], each position of a sequence is described by a vector of values corresponding to the probability of occurrence of a specific residue on that position. However, such an approach is, in great many cases, not needed, e.g. DNA sequencing machines specify always only one nucleotide at a given position within a read. Moreover, storing *profiles* requires more memory, and that certainly would be an issue in the GPU environment.

One of the main problems of pairwise alignment is that the optimal alignment of sequences that maximizes the similarity of residues may not necessarily reflect the biologically correct alignment. To overcome this limitation the idea of *suboptimal* alignments has been introduced. These are the align-

*e-mail: Pawel.Wojciechowski@cs.put.poznan.pl
[1]see http://www.nvidia.com/object/fermi_architecture.html

ments that have nearly the same score as the optimal one. In [18–20] the results consist of a set of suboptimal alignments and the user may choose one based on their biological knowledge. However, this turned out to be impractical, mainly due to the great number of alternative results, especially in the context of massive pairwise alignment computing. In [21, 22] the authors use the information that is common to many suboptimal alignments to identify significant regions in the final alignment. The main drawback here is that it increases already high computational complexity of the problem.

Methods based on Hidden Markov Models (HMM) are yet another family of alignment algorithms. Viterbi et al. [23], for example, computes the highest probability of pairwise alignment for the most common three-state standard (match, insertion, deletion). Taking into account all the modifications mentioned above the HMM-based solution is probably the most interesting. However, current GPU implementations of Viterbi's algorithm [24, 25] present rather poor performance. Nevertheless, we consider the application of HMM in this area as one of our future work.

2. Implementation

G-PAS 2.0 implements all three methods of pairwise sequence alignment, namely NWG, NWS and SW. In addition, it uses the Gotoh's enhancement [14] to provide affine gap penalties. The software offers the option to compute either only score values for a given set of input sequences or score values together with actual alignments. In order to retrieve the alignment, the idea of so called *backtracking matrices* was introduced in *G-PAS*, cf. [11]. This turned out to be more efficient than the algorithm of Myers et al. [26] for relatively short sequences that are commonly used for example in DNA assembly. *G-PAS* performs an alignment of every given sequence with each other and thus may be found especially useful e.g. as a robust solution for the first phase of progressive multiple sequence alignment algorithms. Obviously, the algorithm is executed on GPU which makes it very quick.

The first version of *G-PAS*, released in 2010, was designed mainly for G80 and GT200 GPU architectures. However, as the new architecture of Fermi brought some significant changes, the algorithm needed to be redesigned to fully utilize hardware resources. In *G-PAS* 2.0 access to the shared and the constant memory was optimized to minimize negative effect of bank conflicts. For instance, the substitution matrix is now entirely kept in the shared memory. This, together with some other changes, resulted in a significant performance boost on the Fermi architecture. Furthermore, improved management of the global memory, including e.g. reduction of memory allocation/deallocation calls and memory reusage, contributed to the overall efficiency as well. As a result the software is up to 24% faster on the Fermi architecture. Yet, some noticeable benefits can be also observed on G80 and GT200 architectures.

The multiple GPUs support works very well thanks to our load balancer which uses the largest processing time first (LPT) rule. Although this rule does not guarantee an optimal schedule, it ensures that the upper bound of execution time is equal to $\left(\frac{4}{3} - \frac{1}{3m}\right) \cdot t_{opt}$, where t_{opt} is the optimal execution time and m is the number of processing units [27, 28]. In practice, applying the LPT rule leads to nearly linear speedup in our software.

3. Tests

The main goal of performed tests was to compare the new version of *G-PAS* with the old one. For this purpose we used real biological sequences from the *Ensembl Databases – Release 55*[2]. Three subsets containing 2000, 4000 and 6000 sequences with the average length of 257 amino acids were randomly selected from the Homo Sapiens translation predictions. BLOSUM62 was used as a substitution matrix and the gap penalties were: $G_{open} = 10$, $G_{ext} = 2$. Tests were performed on the following hardware: CPU: Intel Core 2 Quad Q8200, 2.33 GHz, GPU: 2 × NVIDIA GeForce GTX 580 (Fermi) with 1.5 GB of RAM, main RAM: 8 GB.

Figure 1 shows that on average the *G-PAS* 2.0 is faster than its previous version from 19.6% to 24.1%, depending on the algorithm. However, if we compute only the alignment scores, without the backtracking, obtained speedup grows to 29.3–38.6% (see Fig. 2). The results in the figures are presented using a well-known measure of GCUPS which represents the number of updates in the dynamic programming matrix per second. To be more precise, in our case it is:

$$\frac{\sum_{i\in\{1,2,\ldots,n-1\}} \sum_{j\in\{1,2,\ldots,i\}} length_i \cdot length_j}{t \cdot 10^9} \text{ [GCUPS]} \quad (1)$$

where n is the number of input sequences, $length_i$ is the length of the i-th sequence, t represents the time in seconds and the result is given in giga (10^9) CUPS. It is worth noting that the time needed by all side operations, like computation of the values in auxiliary matrices or performing the backtracking procedure, counts in. Hence, the measure underestimates the performance of the algorithms with the backtracking procedure.

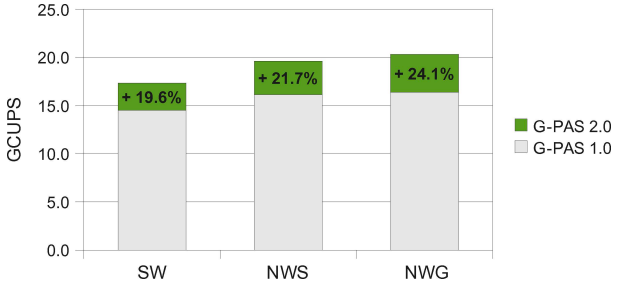

Fig. 1. Mean percentage increase of performance between G-PAS 1.0 and 2.0 on the Fermi architecture. The results for three algorithms are expressed in GCUPS

[2]Available at ftp://ftp.ensembl.org/pub/release-55

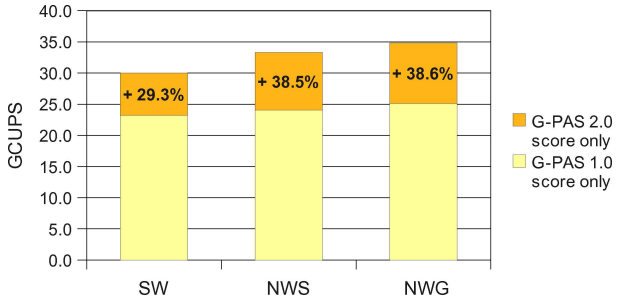

Fig. 2. Mean percentage increase of performance between G-PAS 1.0 and 2.0 on the Fermi architecture in case the algorithms were run in their score only version, i.e. without the backtracking procedure. The results are expressed in GCUPS

We performed also a test on the previous GPU architecture of GT200 to see how the changes influence its speed. This time, instead of Fermi graphics cards, the computer was equipped with two Tesla C1060 GPUs. The new software turned out to be 6.7%, 7.8% and 7.7% faster for SW, NWS and NWG, respectively.

In this section we do not compare our implementation to other state-of-the-art solutions, since it was already done for *G-PAS* 1.0 in [11].

4. Conclusions

G-PAS is a unique software tool that performs pairwise sequence alignment by computing not only the scores but also the actual alignments using an efficient backtracking routine. In this article we presented its new version which is especially optimized to take full advantage of new graphics cards. The software is freely available and may be run on commodity hardware which make it a perfect tool for everyday scientific use.

Availability and requirements

- Project home page: http://gpualign.cs.put.poznan.pl
- Requirements: Linux operating system, CUDA 2.0 or higher, CUDA compliant GPU, make, g++
- License: GNU GPLv3

Acknowledgements. This research project was supported by the grant NO DEC-2011/01/B/ST6/07021 from the National Science Centre, Poland.

REFERENCES

[1] M. Ciznicki, M. Kierzynka, K. Kurowski, B. Ludwiczak, K. Napierala, and J. Palczynski, "Efficient isosurface extraction using marching tetrahedra and histogram pyramids on multiple GPUs", *Lecture Notes in Computer Science* 7204, 343–352 (2012).

[2] M. Blazewicz, S.R. Brandt, M. Kierzynka, K. Kurowski, B. Ludwiczak, J. Tao, and J. Weglarz, "CaKernel – a parallel application programming framework for heterogenous computing architectures", *Scientific Programming* 19 (4), 185–197 (2011).

[3] W. Jendernalik, J. Jakusz, G. Blakiewicz, R. Piotrowski, and S. Szczepanski, "CMOS realisation of analogue processor for early vision processing", *Bull. Pol. Ac.: Tech.* 59 (2), 141–147 (2011).

[4] L. Ligowski and W. Rudnicki, "An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases", *IPDPS* doi:10.1109/IPDPS.2009.5160931 (2009).

[5] Y. Liu, D.L. Maskell and B. Schmidt, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions", *BMC Research Notes* 3, 93 (2010).

[6] S.A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment", *BMC Bioinformatics* 9 (2), S10 (2008).

[7] L. Ligowski, W.R. Rudnicki, Y. Liu, and B. Schmidt, "Accurate scanning of sequence databases with the Smith-Waterman algorithm", *GPU Computing Gems, Emerald Edition* 1, 155–157 (2011).

[8] J. Blazewicz, W. Frohmberg, M. Kierzynka, and P. Wojciechowski, "G-MSA – A GPU-based, fast and accurate algorithm for multiple sequence alignment", *J. Parallel and Distributed Computing*, doi:10.1016/j.jpdc.2012.04.004 (2012).

[9] K. Kwarciak and P. Formanowicz, "A greedy algorithm for the DNA sequencing by hybridization with positive and negative errors and information about repetitions" *Bull. Pol. Ac.: Tech.* 59 (1), 111–115 (2011).

[10] J. Blazewicz, P. Formanowicz, F. Guinand, M. Kasprzak, "A heuristic managing errors for DNA sequencing", *Bioinformatics* 18, 652–660 (2002).

[11] J. Blazewicz, W. Frohmberg, M. Kierzynka, E. Pesch, and P. Wojciechowski, "Protein alignment algorithms with an efficient backtracking routine on multiple GPUs", *BMC Bioinformatics* 12, 181 (2011).

[12] S.B. Needleman and C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins", *J. Mol. Biol.* 48 (3), 443–53 (1970).

[13] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences", *J. Molecular Biology* 147, 195–97 (1981).

[14] O. Gotoh, "An improved algorithm for matching biological sequences", *J. Molecular Biology* 162, 705–708 (1981).

[15] K.-M. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band", *Comput. Appl. Biosci.* 8 (5), 481–487 (1992).

[16] M. Gribskov, A.D. McLachlan, and D. Eisenberg, "Profile analysis: detection of distantly related proteins", *PNAS* 84, 4355–4358 (1987).

[17] M. Gribskov, R. Luthy, and D. Eisenberg, "Profile analysis", *Methods in Enzymology* 183, 146–159 (1990).

[18] M.S. Waterman, "Sequence alignments in the neighborhood of the optimum with general application to dynamic programming", *PNAS* 80, 3123–3124 (1983).

[19] M.S. Waterman and T.H. Byers, "A dynamic programming algorithm to find all solutions in a neighborhood of the optimum", *Math. Biosci.* 77, 179–188 (1985).

[20] D. Naor and D. Burtlag, "On suboptimal alignment of biological sequences", *Proc. 4th Annual Symposium on Combinatorial Pattern Matching* 1, 179–196 (1993).

[21] M. Zuker, "Suboptimal sequence alignment in molecular biology, alignment with error analysis", *J. Mol. Biol.* 221, 403–420 (1991).

[22] M. Vingron and P. Argos, "Determination of reliable regions

in protein sequence alignments", *Protein Engin.* 7, 565–569 (1990).

[23] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm", *IEEE Trans. Inf. Theory* IT-13, 260–269 (1967).

[24] D. Zhang, "An implementation of Viterbi algorithm on GPU", *First Int. Conf. on Information Science and Engineering* 1, CD-ROM (2009).

[25] Z. Du, Z. Yin and D.A. Bader, "A tile-based parallel Viterbi algorithm for biological sequence alignment on GPU with CUDA", *Ninth IEEE Int. Workshop on High Performance Computational Biology Atlanta, GA* 1, CD-ROM (2010).

[26] E.W. Myers and W. Miller, "Optimal alignments in linear space", *Comput. Appl. Biosci.* 4 (1), 11–17 (1988).

[27] R.L. Graham, "Bounds on multiprocessing timing anomalies", *SIAM J. Appl. Math.* 17 (2), 416–429 (1969).

[28] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on Scheduling: From Theory to Applications*, Springer, Berlin, 2007.