Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment

Sergio Galvez1,*, David Diaz1, Pilar Hernandez2, Francisco J. Esteban3, Juan A. Caballero4 and Gabriel Dorado5

1Departamento Lenguajes y Ciencias de la Computación, Universidad de Málaga 29071 Málaga, 2Instituto de Agricultura Sostenible (IAS-CSIC), Alameda del Obispo, s/n, 14080 Córdoba, 3Computer Services, 4Department Estadística and 5Department Bioquímica y Biología Molecular, Universidad de Córdoba 14071 Córdoba, Spain

Associate Editor: Dmitrij Frishman

1 INTRODUCTION

Bioinformatics algorithms and computing power are the main bottlenecks for analyzing huge amount of data generated by the current technologies, such as the 'next-generation' sequencing methodologies. At the same time, most powerful microprocessors are based on many-core chips, yet most applications cannot exploit such power, requiring parallelized algorithms. An example of next-generation bioinformatics, we have developed from scratch a new parallelization of the Needleman–Wunsch (NW) sequence alignment algorithm for the 64-core Tile64 microprocessor. The unprecedented performance it offers for a standalone personal computer (PC) is discussed, optimally aligning sequences up to 20 times faster than the non-parallelized version, thus saving valuable time.

Availability: This algorithm is available as a free web service for the scientific community at http://www.sicuma.uma.es/multicore. The open source code is also available on such site.

Contact: gaizaz@uma.es

Supplementary information: Supplementary data are available at Bioinformatics online.

Received on July 6, 2009; revised on January 12, 2010; accepted on January 13, 2010

2 BACKGROUND AND CONTEXT

Modern computer engineering is driven by multi-threading and emerging multi-processing technologies. The graphics processing unit (GPU) is evolving as well to take advantage of its potential computing power in general-purpose applications (Owens et al. 2007) and hybrid architectures of CPU/GPU, such as the Sony–IBM–Toshiba Cell Broadband Engine (Kahle et al., 2005), the Intel Larrabee and AMD Fusion.

A few bioinformatics applications have been developed for GPU (Manavski and Valle, 2008), Cell and MMX processors (Rognes, 2001; Sachdeva et al., 2005), special SIMD array parallel processors (Di Blas et al., 2005) or even many PC-node clusters (Li et al., 2005), obtaining remarkable acceleration factors and scalable solutions with parallel implementations. So, the tendency is to increase the number of cores to further exploit the parallelism, but the concept of 'core' differs a lot between GPU and CPU. Though they have similar computing power, a GPU core has a very limited set of resources when compared with a CPU core, such as local memory.
API. In addition, we have changed the inherent shape of the matrix of NW with Gotoh’s modifications, but instead of storing a whole sequence to align, open and extend gap costs, match/replace cost which allows specifying the usual parameters for a FastLSA: usage.

parallel wave front of the algorithm, to maximize a two-level cache of temporary storage. To optimize the use of these resources, memory/high-speed solid state disk (SSD) to allocate big quantities and cache sharing (Bell et al., 2008) and (ii) improved mesh for faster inter-core communication. To build a parallelization of an algorithm, the developer focuses on the problem and not on the architecture. This is an extremely important difference when programming for GPU from a productivity point of view, though the number of languages and libraries available for general purpose GPU (GPGPU) is growing (Che et al., 2008).

Finally, we focus on the FastLSA algorithm to align a sequence $s_1$ of length $l_1$ with a sequence $s_2$ of length $l_2$. FastLSA is the same as NW with Gotoh’s modifications, but instead of storing a whole matrix of $l_1 \times l_2$ data, it stores only one row/column from each $k$ row/column, so that it can be implemented through a wavefront parallelism. These intermediate values allow FastLSA to obtain the alignment quickly, when compared to other linear space algorithms as Hirschberg’s (Hirschberg, 1975; Driga et al., 2006). With less storage requirements, FastLSA is especially suitable for aligning very long sequences. To focus on the main discrepant regions of an extremely long alignment, the researcher may use a viewer like the Omega1-JalView, integrated in the Omega1-Brigid tool that we have developed for quality control to detect fraudulent olive oil (Díaz et al., 2009).

3 IMPLEMENTATION AND PARAMETERS

Our algorithm (MultiCore64-NW) has been developed entirely in C and deployed into TilExpress-20G cards by means of a parallel implementation of FastLSA used for benchmarking in BioBench (Albayraktaroglu et al., 2005); both use the best suitable $k$ size. In addition, we have included $z$-align (Batista et al., 2008) and our own reference implementation in Java; a comparison between execution times of fine-tuned C and Java programs may be considered well-balanced (Shafl et al., 2009). All of them have been executed on a Quad Core Intel Xeon 2.0 GHz PC with 8 GB of DDR2 memory in quad-channel. GPU-based algorithms, such as compute unified device architecture (CUDA) Smith–Waterman (Manavski and Valle, 2008) and other widely known algorithms do not support long-sequence alignments; e.g. EMBOS’S needle cannot manage sequences above $\sim 40\,000$ (40k) bp, and thus are not shown in this comparison chart.

Finally, MultiCore64-NW shows a good scalability, as shown in Figure 1. The additional time consumed by MultiCore64-NW when related to a maximum theoretical is due to the communication between workers and scheduler cores. We have implemented the complete algorithm, but have focused (as most authors) on the forward phase (1st stage) of FastLSA—much more expensive computationally—with sequences of different lengths.

![Fig. 1. Processing time of MultiCore64-NW with the increasing number of tiles, using sequences of 250 kb.](image)

![Fig. 2. Benchmark of first stage of main wavefront parallel NW/SW implementations. Time values are in seconds and sequence length in kb.](image)

4 BENCHMARKING

The MultiCore64-NW effectively uses 59 working tiles. The remaining five cores correspond to one dedicated to communications with the host, three shared for internal operations and one distributing the jobs among the worker tiles.

Figure 2 shows the time consumed by MultiCore64-NW compared with the parallel linear space algorithm (PLSA) implementation of FastLSA used for benchmarking in BioBench (Albayraktaroglu et al., 2005); both use the best suitable $k$ size. In addition, we have included $z$-align (Batista et al., 2008) and our own reference implementation in Java; a comparison between execution times of fine-tuned C and Java programs may be considered well-balanced (Shafl et al., 2009). All of them have been executed on a Quad Core Intel Xeon 2.0 GHz PC with 8 GB of DDR2 memory in quad-channel. GPU-based algorithms, such as compute unified device architecture (CUDA) Smith–Waterman (Manavski and Valle, 2008) and other widely known algorithms do not support long-sequence alignments; e.g. EMBOS’S needle cannot manage sequences above $\sim 40\,000$ (40k) bp, and thus are not shown in this comparison chart.

In Figure 2, we have selected the implementations of wavefront algorithms, which generate results up to 1000 kb. Our Java implementation of FastLSA shows similar results as PLSA. The PLSA (Li et al., 2005) is a parallel implementation of FastLSA slower than a sequential implementation in Java, even using the four cores of the Xeon at their full potential (PLSA source cannot be migrated to Tile64 architecture, since it uses very specific resources unavailable for Tile64). Although PLSA is a Smith–Waterman
Within the limits of a standalone PC, this is the first time we are focusing on the PRANK algorithm (Li and Goldman, 2008). This algorithm may take a dendrogram (phylogenetic tree) as parameter, being very sensible to its quality, and providing very good results when the tree approximates the best achievable result. The triangular matrix of distances that allows generating this dendrogram is obtained with MultiCore64-NW, up to 20 times faster than with a non-parallelized algorithm. Our development offers exceptional performance in standalone PC; the mentioned matrix for 10 sequences of approximately 100 kb can be obtained from 55 NW pairwise alignments in 765 s (~13 min). These and similar developments will have a significant impact on bioinformatics, allowing to answer the need for new parallelized algorithms and massive computational power, which are already demanding the current technologies, such as the so-called ‘next-generation’ sequencing methodologies.

ACKNOWLEDGEMENTS

We are grateful to Tilera for providing hardware and software tools. http://www.tilera.com. S.G. and G.D. have contributed to the project leadership.

Funding: ‘Ministerio de Ciencia e Innovación’ (AGL2006-12550-C02-01, AGL2006-12550-C02-02); ‘Consejería de Agricultura y Pesca’ of ‘Junta de Andalucía’ (041/C/2007); ‘Grupo PAI’ (AGR-248); ‘Universidad de Córdoba’ (Ayuda a Grupos), Spain.

Conflict of Interest: none declared.

REFERENCES


5 RESULTS, DISCUSSION AND FURTHER WORK

Within the limits of a standalone PC, this is the first time that a general-purpose many-core chip has been used for bioinformatics, showing a great potential for algorithms in which there is no need of floating point calculi, and demonstrating the potential of parallelization and many-core chips for the next-generation bioinformatics. Thus, in comparison with widespread implementations of NW, such as EMBOSS’ needle or parallelized FastLSA, the MultiCore64-NW obtains the optimal alignment up to 20 times faster. Controller scheduling time and memory access time are the limitation of memory resources. MultiCore64-NW can align longer sequences with the only restriction of memory space.

We are further developing a multiple alignment algorithm in order to obtain new benchmarks against classical methods such as ClustalW and PRANK. Using MultiCore64-NW, we have obtained as well optimal pairwise phylogenetic distances of mid-length sequences (around 100 kb), so now we are focusing on the PRANK algorithm (Li and Goldman, 2008).
Sergio Gálvez et al.


