BIOINFORMATICS APPLICATIONS NOTE

Vol. 26 no. 5 2010, pages 683–686
doi:10.1093/bioinformatics/btq017

Sequence analysis

Advance Access publication January 16, 2010

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

Sergio Gálvez[1,*], David Díaz[1], Pilar Hernández[2], Francisco J. Esteban[3], Juan A. Caballero[4] and Gabriel Dorado[5]

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

Associate Editor: Dmitrij Frishman

ABSTRACT

Motivation: 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. As 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: galvez@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

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. Whereas multi-core processors with a few cores provide a relative improvement in desktop applications, many-core processors (chips with tens of cores) represent the next big step in affordable high-performance computation. Proposals and prototypes include Intel Terascale Processor (Mattson *et al.*, 2008), Sun Microsystems UltraSPARC T2 Pro (Shah *et al.*, 2007) and Plurality Hypercore Architecture Line (HAL, announced in Grenoble, December 2008). Pioneering such developments, Tilera has released several Tile64 processor-based cards (Bell *et al.*, 2008), together with the Tilera's Multicore Development Environment (MDE). From a research group point of view, one of the main advantages of these cards is that they can be inserted into a PCI-Express slot of a personal computer (PC) and may be used as a standalone implementation. At the same time, the host computer central processing unit (CPU) may be used for desktop purposes, while the parallel workload is carried out in the background by the Tile64 card. Also, there is no need to require the services of an expensive remote server to do such work. A Tile64 contains 64 cores (tiles), communicating over a 31 Tbps bandwidth iMesh architecture. Yet, most current applications cannot exploit such power, requiring parallelized algorithms. These cards have been deployed so far on high throughput video and networking (Flohr, 2008). As an example of the next-generation of bioinformatics, we have demonstrated multi-core performance by developing from scratch a parallelized FastLSA algorithm [an improvement of the Needleman–Wunsch (NW) aligner]. The Tile64 RISC processor lacks floating point instructions, so it is not suitable for all types of supercomputing; yet we have found that it can be exploited for biological information management, where only integer numbers are used.

The NW algorithm may be considered the most classical alignment method when the affine gap improvement is applied to the cost function (Gotoh, 1982). FastLSA is an intuitive NW implementation when there are multiple cores and lots of main memory available (Driga *et al.*, 2006). Thus, we have used it as a practical and useful implementation of our development, which has been opened publicly by means of a free web service at http://www.sicuma.uma.es/multicore.

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.*, 2008), 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,

*To whom correspondence should be addressed.

cache, internal registers and network functionalities. As a System on Chip (SoC), each core of the Tile64 processors is capable of running an independent operating system (Linux) and may be seen as a general purpose-independent computer. For this reason, it can run applications written in any general purpose programming language, such as the C/C++ used for standard PC. This allows many multi-threading and sequential existing programs to be easily adapted to the Tilera's Application Programming Interface (API), in order to exploit the parallelism of the card, enhancing communications between cores and iMesh.

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 Ω-JalView, integrated in the Ω-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 an integrated development environment based on Eclipse (MDE). MultiCore64-NW is a parallel FastLSA deployed on 64 cores, taking advantage of the hardware enhancements: (i) three cache levels for faster memory access; (ii) improved mesh for faster inter-core communication and cache sharing (Bell *et al.*, 2008); and (iii) 8 GB of shared memory/high-speed solid state disk (SSD) to allocate big quantities of temporary storage. To optimize the use of these resources, MultiCore64-NW has been written from scratch to use the Tilera's API. In addition, we have changed the inherent shape of the parallel wave front of the algorithm, to maximize a two-level cache usage.

MultiCore64-NW is freely available on Internet as a web service, which allows specifying the usual parameters for a FastLSA: sequences to align, open and extend gap costs, match/replace cost and matrix to be used: NUC.4.4, NUC.4.2, etc. The k default size is 750. The resulting alignment is shown on the web server or (optionally) sent to the provided e-mail address.

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.

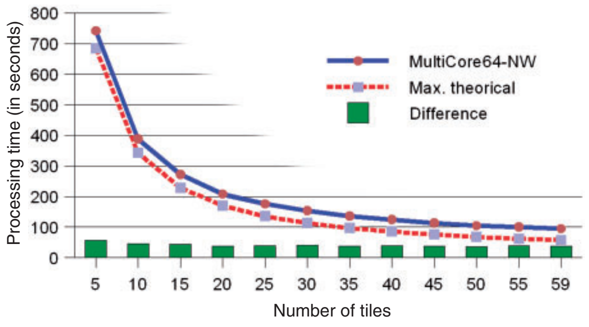

Fig. 1. Processing time of MultiCore64-NW with the increasing number of tiles, using sequences of 250 kb.

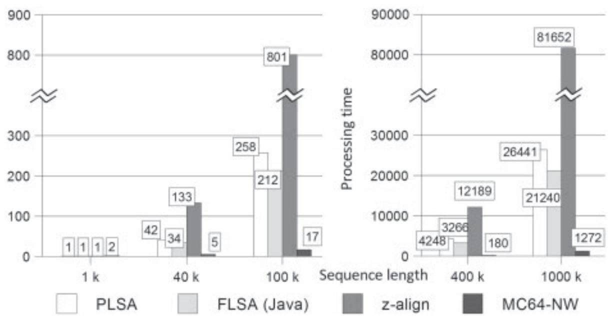

Fig. 2. Benchmark of first stage of main wavefront parallel NW/SW implementations. Time values are in seconds and sequence length in kb.

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 (Shafi *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. EMBOSS' needle cannot manage sequences above ∼40 000 (40 k) 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

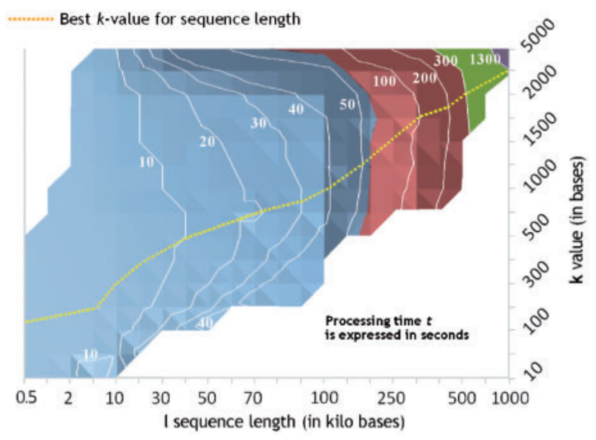

Fig. 3. Relationship between k size, sequence length and processing time.

implementation, its first stage is very similar in complexity to NW. Therefore, against the fastest implementation in Java (z-align shows higher times of execution, even using four cores), MultiCore64-NW obtains a gain that ranges from 6.80 in small sequence lengths to a range between 16.70 and 18.14 in large ones. Figure 2 reveals that the performance ratio of MultiCore64-NW decreases from 400 to 1000 kb sequences, due to the higher k value required to run the algorithm inside the 8 GB of memory of the TilExpress-20G card. MultiCore64-NW can align longer sequences with the only limitation of memory resources.

On the other hand, Figure 3 shows the processing t time (Z-axis) required to align sequences of l length (X-axis) with several values of k (Y-axis). The axes are non-linear to better visually understand the relationship between the values. A dotted line marks the minimum time required for each pair of sequences of l length and, as result, the optimal k value to perform the alignment.

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 through iMesh are extremely low, in comparison to processing time, so the k optimal values should maximize the number of processing tiles, without excessively sub-dividing the matrix. Even more, with very long sequences (>750 kb) the k value must be incremented to decrease the amount of SSD memory required. This may be overcome in future versions by using the host file system in addition to the TilExpress-20G SSD. We want to remark that to obtain such high performance the algorithm has been written from scratch, in order to maximize the use of the Tilera's card resources. Such approach is inherent to any parallel architecture optimization.

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 (Löytynoja 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 easily 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

Albayraktaroglu,K. *et al.* (2005) BioBench: a benchmark suite of bioinformatics applications. In *Proceedings of 2005 IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, TX, USA, pp. 2–9.
Batista,R.B. *et al.* (2008) A parallel strategy for biological sequence alignment in restricted memory space. *J. Parallel Distrib. Comput.*, **68**, 548–561.
Bell,S. *et al.* (2008) TILE64^{TM} processor: a 64-core SoC with mesh interconnect. In *Proceedings of 2008 IEEE Integrated Solid State Circuits Conference*, San Francisco, CA, USA, pp. 88–90.
Che,S. *et al.* (2008) A performance study of general-purpose applications on graphics processors using CUDA. *J. Parallel Distrib. Comput.*, **68**, 1370–1380.
Di Blas,A. *et al.* (2005) The UCSC Kestrel parallel processor. *J. Parallel Distrib. Comput.*, **68**, 1370–1380.
Díaz,D. *et al.* (2009) Intuitive bioinformatics for genomic applications: omega-brigid workflow framework. In *Proceedings of 10th International Work-Conference on Artificial Neural Networks*, Salamanca, Spain, pp. 1084–1091.
Driga,A. *et al.* (2006) FastLSA: a fast, linear-space, parallel and sequential algorithm for sequence alignment. *Algorithmica*, **45**, 337–375.
Flohr,W (2008) *Implementation of an MPEG Codec on the Tilera^{TM} 64 processor. Research Project*. Washington University, St. Louis, pp. 1–11.
Gotoh,O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705–708.
Hirschberg,D. (1975) A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, **18**, 341–343.
Kahle,J.A. *et al.* (2005) Introduction to the cell multiprocessor. *IBM Syst. J.*, **49**, 589–604.
Li,E. *et al.* (2005) Parallel linear space algorithm for large-scale sequence alignment. In *Proceedings of Euro-Par 2005*, Lisboa, Portugal, pp. 1207–1216.
Löytynoja,A. and Goldman,N. (2008) Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, **320**, 1632–1635.
Manavski,S.A. and Valle,G. (2008) CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment. *BMC Bioinformatics*, **26** (Suppl. 2), S10.

Mattson,T.G. *et al*. (2008) Programming the Intel 80-core network-on-a-chip Terascale processor. In *Proceedings of 2008 ACM/IEEE Conference on Supercomputing*. Austin, TX, USA, pp. 1–11.

Owens,J.D. *et al.* (2007) A survey of general-purpose computation on graphics hardware. *Comp. Graph. Forum*, **26**, 80–113.

Rognes,T (2001) ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Res.,* **29**, 1647–1652.

Sachdeva,V. *et al.* (2008) Exploring the viability of the cell broadband engine. *Parallel Comp.*, **34**, 616–626.

Shafi,A. *et al.* (2009) A comparative study of Java and C performance in two large-scale parallel applications. *Concurr. Comput. Prac. Exper.*, **21**, 1882–1906.

Shah,M. *et al*. (2007) UltraSPARC T2: a highly-treaded, power-efficient, SPARC SoC. In *Proceedings of 2007 IEEE Asian Solid-State Circuits Conference*, Jeju, Korea, pp. 22–25.