The Potential of Intel® Xeon Phi™ for DNA Sequence Analysis

Suejb Memeti* and Sabri Pllana*

* *Department of Computer Science, Linnaeus University, 351 95 Växjö, Sweden,*

ABSTRACT

Genetic information is increasing exponentially, doubling every 18 months. Analyzing this information within a reasonable amount of time requires parallel computing resources. While considerable research has addressed DNA analysis using GPUs, so far not much attention has been paid to the Intel Xeon Phi coprocessor. In this paper we present an algorithm for large-scale DNA analysis that exploits the thread-level and the SIMD parallelism of the Intel Xeon Phi coprocessor. We evaluate our approach for various numbers of cores and thread allocation affinities in the context of real-world DNA sequences of mouse, cat, dog, chicken, human and turkey. The experimental results on Intel Xeon Phi show speed-ups of up to $10\times$ compared to a sequential implementation running on an Intel Xeon processor E5.

KEYWORDS: DNA Analysis, Intel Xeon Phi, Many-core, Pattern Matching, k-mers

1 Introduction

There is a growing interest in molecular biology community to understand the information that is encoded within the Deoxyribonucleic Acid (DNA) sequences of each organism [AAB06]. A DNA sequence contains specific genetic instructions that make the living organisms function in a proper way.

Discovery of differences and similarities of organisms and exploration of the evolutionary relationship between them, often require comparisons of the corresponding DNA sequences. Examples include: checking whether one sequence is a sub-sequence of another, or finding a sub-sequence that appears in the same order in both DNA sequences [Ben00]. The process of searching for certain sub-sequences of length *k*, so called *k-mers*, is performed with pattern matching algorithms.

According to Benson et al. [BCC$^+$13] the number of DNA sequences and nucleotide bases in these sequences is doubling every 18 months. Real-world DNA sequences comprise several Gigabytes and the process of extracting the important information demands the adequate use of parallel computing resources to be completed within a reasonable time. A quick DNA analysis may have a decisive role in many applications including: preventing the evolution of different viruses and bacterias during an early phase [CGGG03]; early diag-

[1] E-mail: {suejb.memeti,sabri.pllana}@lnu.se

nosis of genetic predispositions to certain diseases (such as, cancer, cardiovascular diseases) [MHC$^+$11]; and DNA forensics (such as, parentage testing, criminal investigation) [LR00].

Related research has addressed extensively DNA analysis using GPUs [LLCC13, KM09, BAA$^+$13, TV10]. So far not much research was focused on DNA analysis using pattern matching algorithms designed specifically for the Intel Xeon Phi coprocessor.

In this paper, we present a Finite Automata (FA) based parallel algorithm for DNA analysis that is designed to exploit the thread- and data-level (SIMD) parallelism that is available on the Intel® Xeon Phi coprocessor and is optimized using different algorithmic strategies. An empirical evaluation of our algorithm is performed with real-world DNA sequences of different living species. The results show speedup of up to $10\times$ compared to the sequential version of the algorithm running on Intel Xeon E5 CPU.

2 Approach

The key features of our algorithm and implementation for parallel DNA analysis on Intel Xeon Phi are: (1) decomposition of the input DNA sequence across the available threads, (2) exploiting the SIMD parallelism, and (3) reducing the memory references using a suitable representation for the State Transition Table.

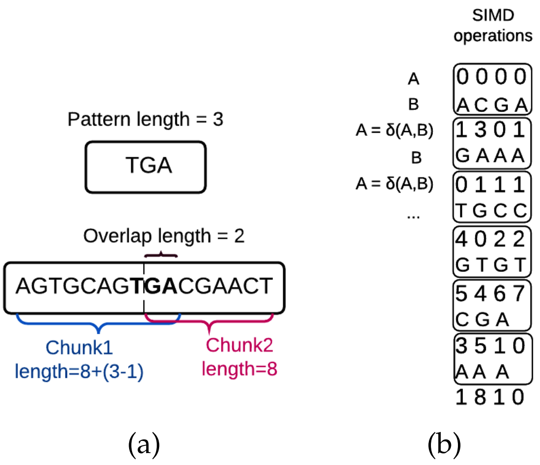

Figure 1: Thread-level and SIMD parallelism; (a) splitting the DNA sequence into chunks; (b) vectorization of the transition function.

Figure 1a illustrates our thread-level parallelization strategy based on domain decomposition, which means the input DNA sequence is evenly split into chunks among the available threads. We use an overlapping approach of $m-1$ to match the occurrences of patterns that cross the chunks boundaries, where m represents the pattern length.

With respect to the SIMD-parallelism our algorithm implementation uses the potential of the 512-bit vector registers of the Intel Xeon Phi architecture for the transition function of FA. Our algorithm exploits vector units of Intel Xeon Phi, by splitting the chunks further into v parts, where v represents the vector length. The operations (such as, determining the next state , or identifying the final states) are performed on multiple data points simultaneously. Figure 1b illustrates an example of the vectorized transition function (δ) with respect to the input from Fig. 1a. Assuming that the vector length is four, the first SIMD δ operations will be performed on the characters at positions 0, 4, 8, and 12 of the input, while in the second iteration the SIMD operations are performed on the characters 1, 5, 9, and 13, until the end of the input is reached.

Furthermore, algorithmic optimization strategies are used to achieve high performance, such as: (1) eliminating the failure transitions from the AC FA to avoid the drawback of the non-deterministic transitions [MP14], (2) using ASCII representation of the Transition Table to reduce the memory references by trading off memory space with access speed, and (3) reordering the number of final states to simplify the process of identifying the final states.

3 Results

We evaluated our algorithm on the Intel Xeon Phi 7120P with different number of threads using the real-world DNA sequences of mouse (2.7GB), cat (2.4GB), dog (2.4GB), chicken (1GB), human(3.2GB) and turkey(0.2GB). Furthermore, we have varied the threads affinity, by allocating the threads under *compact*, *balanced*, and *scatter* mode.

From the results of varying thread allocation types, we obtained that for 240 threads, the *balanced* mode is the fastest one for all tested DNA sequences. Therefore, the performance data for scalability (Fig. 2) and the speedup (Fig. 3) is collected for the *balanced* thread affinity mode.

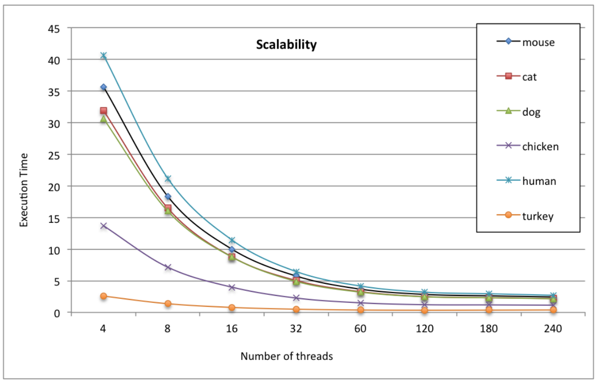

Figure 2: The scalability of our algorithm on the Xeon Phi for various number of threads and problem sizes.

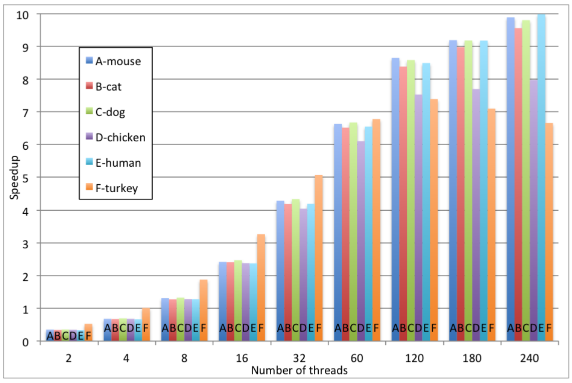

Figure 3: Speedup of our algorithm with respect to a sequential version running on an Intel Xeon E5-2695v2 CPU.

Fig. 2 shows the scalability of our algorithm when we increase the number of threads on the Intel Xeon Phi coprocessor. Our algorithm scales well up to 120 threads for most of the tested DNA sequences. Increase of the number of threads to 180 or 240, results with a modest performance improvement due to the thread management overhead. The performance gain when using a larger number of threads is higher for larger DNA sequences. Thus the best scalability we observe for the human DNA sequence, which is the largest DNA sequence used in our experiments.

Fig. 3 presents the achieved speedup. The maximal speedup of $10\times$ is achieved for the human DNA sequence using 240 threads compared to a sequential version running on an Intel Xeon E5-2695v2 CPU.

4 Summary and Future Work

In this paper we have presented an approach for accelerating DNA analysis using the Intel Xeon Phi coprocessor. The proposed parallel algorithm is based on finite automata and is used for counting and extracting the location of k-mers in a DNA sequence. Our approach exploits the thread-level and SIMD parallelism of the Intel Xeon Phi coprocessor, and therefore it is suitable for large-scale DNA sequences. Experiments with real-world data-sets of several GB demonstrate that the algorithm scales well with respect to various numbers of threads and problem sizes. The best scalability we observed for the human DNA sequence, which was the largest DNA sequence used in our experiments.

Future work will address the DNA analysis on the upcoming generation of the Intel Xeon Phi coprocessor known as the *Knights Landing*.

References

[AAB06] Srinivas Aluru, Nancy M. Amato, and David A. Bader. Editorial: Special section on high-performance computational biology. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):737–739, 2006.

[BAA$^+$13] X Bellekens, I Andonovic, RC Atkinson, C Renfrew, and T Kirkham. Investigation of gpu-based pattern matching. In *The 14th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet2013) (PGNet2013)*, 2013.

[BCC$^+$13] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic acids research*, 41(D1):D36–D42, 2013.

[Ben00] David R. Bentley. Decoding the human genome sequence. *Human Molecular Genetics*, 9(16):2353–2358, 2000.

[CGGG03] Francis S Collins, Eric D Green, Alan E Guttmacher, and Mark S Guyer. A vision for the future of genomics research. *Nature*, 2003.

[KM09] Charalampos S Kouzinopoulos and Konstantinos G Margaritis. String matching on a multicore gpu using cuda. In *Informatics, 2009. PCI'09. 13th Panhellenic Conference on*, pages 14–18. IEEE, 2009.

[LLCC13] Cheng-Hung Lin, Chen-Hsiung Liu, Lung-Sheng Chien, and Shih-Chieh Chang. Accelerating pattern matching using a novel parallel algorithm on gpus. *Computers, IEEE Transactions on*, 62(10):1906–1916, Oct 2013.

[LR00] M. Luftig and S. Richey. Dna and forensic science. *New Eng. L. Rev.*, 2000.

[MHC$^+$11] Alexander Mellmann, Dag Harmsen, Craig A Cummings, Emily B Zentz, Shana R Leopold, Alain Rico, Karola Prior, Rafael Szczepanowski, Yongmei Ji, Wenlan Zhang, et al. Prospective genomic characterization of the german enterohemorrhagic escherichia coli o104: H4 outbreak by rapid next generation sequencing technology. *PloS one*, 6(7):e22751, 2011.

[MP14] S. Memeti and S. Pllana. Parem: A novel approach for parallel regular expression matching. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 690–697, Dec 2014.

[TV10] A. Tumeo and O. Villa. Accelerating dna analysis applications on gpu clusters. In *Application Specific Processors (SASP), 2010 IEEE 8th Symposium on*, pages 71–76, June 2010.