Accelerating BLASTP on the Cell Broadband Engine

Huiliang Zhang, Bertil Schmidt, and Wolfgang Müller-Wittig

School of Computer Engineering, Nanyang Technological University, Singapore
{hlzhang,asbschmidt,askwmwittig}@ntu.edu.sg

Abstract. The enormous growth of biological sequence databases has caused bioinformatics to be rapidly moving towards a data-intensive, computational science. As a result, the computational power needed by bioinformatics applications is growing rapidly as well. The recent emergence of low cost parallel accelerator technologies has made it possible to reduce execution times of many bioinformatics applications. In this paper, we demonstrate how the PlayStation®3, powered by the Cell Broadband Engine, can be used as an efficient computational platform to accelerate the popular BLASTP algorithm.

1 Introduction

Scanning genomic sequence databases is a common and often repeated task in molecular biology. The scan operation consists of finding similarities between a particular query sequence and all sequences of a bank. Dynamic programming (DP) based alignment algorithms whose complexities are quadratic with respect to the length of the sequences can detect similarities between the query sequence and a subject sequence [13]. One frequently used approach to speed up this prohibitively time consuming operation is to introduce heuristics in the search algorithm [2, 7, 9]. Among these heuristics, BLAST (the *B*asic *L*ocal *A*lignment *S*earch *T*ool [1, 2]) is the most popular software. It is used to run millions of queries each day. However, evaluating a single query to a large database with BLAST usually takes several minutes on a modern workstation. These scan time requirements are likely to become even more severe due to the rapid growth in the size of these databases. Hence, finding fast solutions is of highest importance to research.

In this paper we present a new approach to accelerate BLASTP for scanning protein databases on the Cell Broadband Engine (Cell BE). The Cell BE [6, 8, 11] is a recently introduced single-chip heterogeneous multi-core processor, which has been jointly developed by Sony, Toshiba and IBM. Previous work on parallelizing sequence analysis applications on the Cell BE focused on DP-based algorithms such as Smith-Waterman and HMMer [12,14]. Compared to these highly regular applications, parallelization of BLASTP is more challenging since it consists of a pipeline of computations with different memory and processing requirements.

The rest of the paper is organized as follows. Section 2 highlights features of the Cell BE architecture. An overview of the BLASTP algorithm is given in Section 3. Section 4 presents our parallelization approach on the Cell BE. Performance is evaluated in Section 5. Section 6 concludes the paper.

2 Cell Broadband Engine

The Cell BE [6] is a single-chip heterogeneous multi-core processor. It contains two types of processors: a *PowerPC Processor Element* (PPE) and eight *Synergistic Processor Elements* (SPEs) [8,11]. An integrated high-bandwidth bus called the *Element Interconnect Bus* (EIB) connects the processors and their ports to external memory and I/O devices. A block diagram of the Cell BE is shown in Figure 1. The PPE is a 64-bit PowerPC architecture. It is fully compliant with the 64-bit Power Architecture specification and can run 32-bit and 64-bit operating systems and applications. Each SPE is able to run its own individual application program. It consists of a processor designed for streaming workloads, a local memory, and a globally coherent DMA engine. The SPE implements a Cell-specific set of SIMD instructions. With all eight SPEs active, the Cell BE is capable of a peak performance of around 200 GFlops using single precision floating point arithmetic.

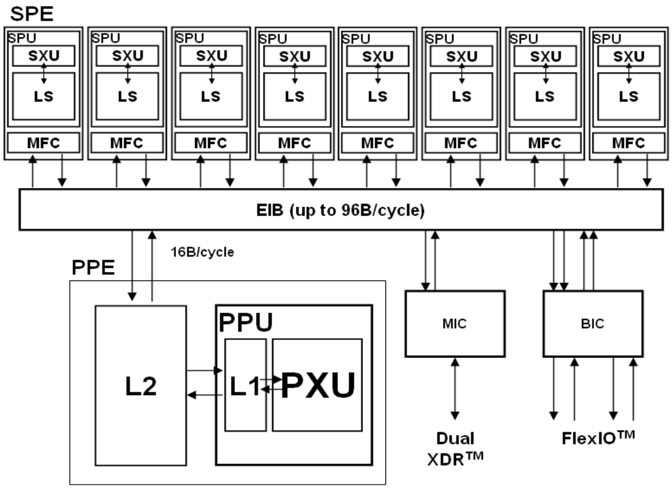


Fig. 1. Block diagram of the Cell BE architecture

Although it is a multiprocessor system on a chip, the Cell BE processor is not a traditional shared-memory multiprocessor. One of the major characteristics is that an SPE can execute programs and directly load and store data only from and to its private *Local Store* (LS). Since SPEs lack shared memory, they must communicate explicitly with the PPE or other SPEs using one of three available communication mechanisms: *DMA transfers*, *mailbox messages*, or *signal-notification messages*. All three communication mechanisms are controlled by the SPE's MFC (Memory Flow Controller).

The design of a parallel algorithm on the Cell BE requires an efficient partitioning of the computation between PPE and SPEs. A general approach is to perform as much as possible computations on the SPEs while the PPE is used for coordinating the control flow. Furthermore, the local memory (LS) of a PPE is very limited (only 256

KB for storing both instructions and data). Therefore, DMA data transfers between main memory and SPEs is often a bottleneck in Cell BE applications and should therefore be minimized.

3 BLASTP Algorithm

The basic idea for fast sequence database search is *filtration*. Filtration assumes that good alignments usually contain short exact matches. Such matches can be quickly computed by using data structures such as lookup tables. Identified matches are then used as seeds for further detailed analysis. The analysis pipeline of the BLASTP algorithm is shown in Figure 2. It consists of four stages. Each stage progressively reduces the search space in the database for significant alignment. We briefly describe each step in the following. More details can be found in [1, 2].

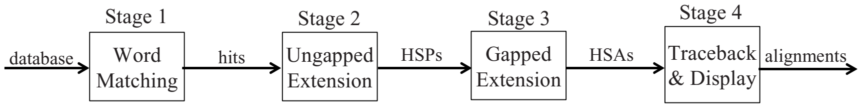


Fig. 2. The BLASTP processing pipeline

Stage 1: This stage identifies *hits*. Each hit is defined as an offset pair (*i*,*j*) for which $\sum_{k=0}^{w-1} sbt(Q[i+k], D[j+k]) \geq T$, where *sbt* is a amino acid substitution matrix (e.g. BLOSUM65), *w* is the user-defined word length, *T* is a user-defined threshold, *Q* is the query sequence and *D* is the database. BLASTP implements this stage by preprocessing *Q* as follows. For each position *i* of *Q* the neighborhood $N(Q[i \ldots i+w-1], T)$ is computed consisting of all *w*-mers *p* for which $\sum_{k=0}^{w-1} sbt(Q[i+k], p[k]) \geq T$. The complete neighborhood of a query is typically stored in an efficient data structure such as a lookup table or a finite-state automaton. The default parameter values are w=3 and T=11.

Stage 2: Stage 2 outputs *HSPs* (high-scoring segment pairs) between *Q* and *D*. HSPs are identified by performing an ungapped extensions on a diagonal *d* which contains a non-overlapping hit pair (i_1, j_1), (i_2, j_2) within a window *A*; i.e. $d = i_1 - j_1 = i_2 - j_2$ and $w \leq i_2 - i_1 \leq A$. If the resulting ungapped alignment scores above a certain threshold it is passed to Stage 3.

Stage 3: This stage outputs *HSAs* (high scoring alignments) between *Q* and *D*. HSAs are identified by performing a seeded banded gapped dynamic programming based alignment algorithm using the previously identified HSPs as seeds. Alignments that score above a certain threshold are then passed to the final stage.

Stage 4: The final alignments of the highest scoring sequences are calculated and displayed to the user. This requires the computation of the traceback path using the Smith-Waterman algorithm.

An execution profiling of the BLASTP algorithm for scanning the Genbank non-redundant protein database shows the following breakdown of execution time:

Stage 1: 37%, Stage 2: 31%, Stage 3: 30%, Stage 4: 2%.

Hence, in order to efficiently map BLASTP on the Cell BE all stages except Stage 4 need to be parallelized. Previous work on parallelizing BLASTP has focused on distributed memory architectures such as clusters [10] and reconfigurable hardware [12]. This paper is to our knowledge the first ever reported parallelization of BLASTP on the Cell BE.

4 Parallelizing the BLASTP Algorithm on the Cell BE

In order to achieve an efficient parallelization of map the BLASTP algorithm on the Cell BE we need to address the following challenges.

1. *Limited local storage of the SPE.* A major limitation when designing SPE kernels is that their local memory is only 256 KByte for both instructions and data. Using default parameter for w and T the size of the lookup table used for Stage 1 by NCBI BLASTP is already around 400KByte for 100 randomly selected query sequences. Therefore, we need to use an alternative data structure which requires significantly less memory.
2. *Data transfer and coordination between PPE and SPEs.* The different stages of the BLASTP algorithm constitute a processing pipeline where the throughput of each stage in the pipeline depends on the filtration efficiency of the previous stage. Therefore, an efficient and flexible mechanism to transfer sequences from the database to the SPEs needs to be implemented. The PPE needs to coordinates this data transfer.

Figure 3 shows our mapping of the different stages of the BLASTP algorithm onto the Cell BE. Stage 4 includes a ranking procedure on all database sequences that have passed Stages 1-3: The top 500 or less matching sequences whose scores exceed a certain threshold are displayed in descending order. Thus, this stage is performed by PPE. SPE kernels filter the database as follows. Information about all subject sequences from the database that have passed Stages 1-3 on an SPE are sent to the

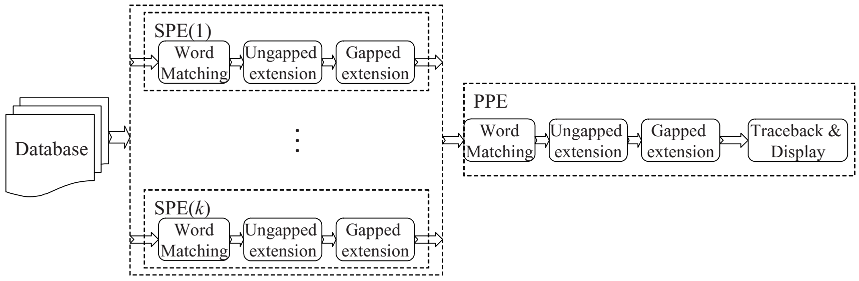
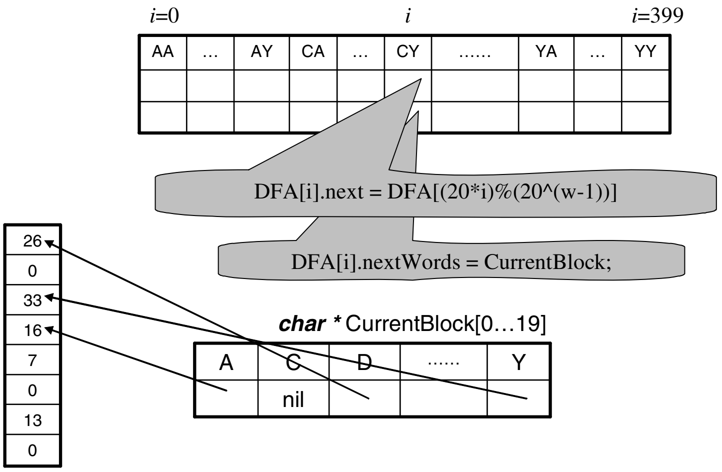


Fig. 3. Mapping of the different stages of the BLASTP algorithm onto the Cell BE

PPE. Upon receiving this information, the PPE completes Stages 1-4 for these subject sequences. The reason why not only Stage 4 is performed on the PPE is that this stage requires additional information from the previous stages and storing this on the SPEs would be too memory-intensive. However, since this redundant computation is merely performed for very few subject sequences the additional runtime is negligible (see Section 5 for details).

As mentioned above, the size of the codeword lookup data structure used by NCBI BLAST is too large for the local store of the SPEs. Therefore, we are using a more memory-efficient data structure for Stage 1. The utilized data structure is a compressed *deterministic finite-state automaton* (DFA), which is similar to the approach used by FSA-BLAST [3, 4]. The compressed DFA for w=3 is illustrated in Figure 4.



i=0
i
i=399
AA ... AY CA ... CY ...... YA ... YY
DFA[i].next = DFA[(20*i)%(20^(w-1))]
DFA[i].nextWords = CurrentBlock;
char * CurrentBlock[0...19]
A C D ...... Y
nil
26
0
33
16
7
0
13
0

Fig. 4. Illustration of the compressed FSA data structure for w=3

Each possible prefix of lengths $w-1$ is represented by a state; i.e. for w=3 there are 400 states representing the prefixes AA to YY, which are stored in the array DFA[i] in Figure 4. Each state has two transitions: one to the next state (DFA[i].next) and one to a list of 20 words (DFA[i].nextWords). Each entry in this list (currentBlock[0..19]) contains a pointer to an array of query positions. These query positions represent the neighborhood $N(w,T)$ of the associated w-mer. This data structure allows the compression of frequently used query positions that are in neighborhoods of similar w-mers. For example in Figure 4, N('CYC',T) = {33, 16, 7} and N={'CYA',T} = {16, 7}. By storing these positions in subsequent order terminated by "0" it is possible to re-use memory for both neighborhoods. Our experiments have shown that the size the compressed DFA is only 43.8 KByte on average. Hence, it is possible to store the complete data structure on each SPE for most queries.

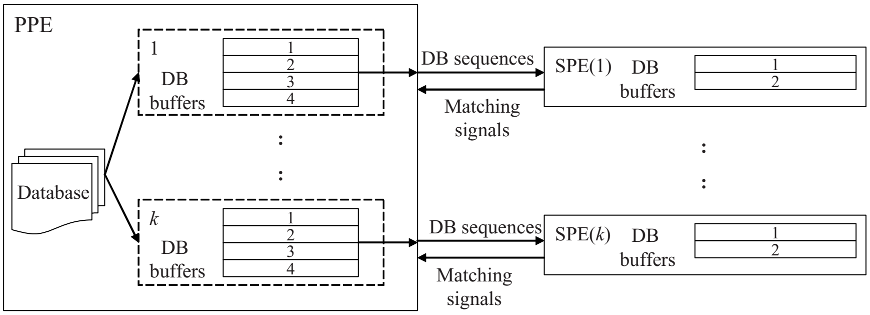


Fig. 5. Buffering scheme

The DFA is transferred into each SPE. The PPE then reads sequences from the database and transfers them to the SPEs by Direct Memory Access (DMA). In order to hide latencies and achieve good load balancing, we have implemented four buffers on the PPE per SPE and two buffers on each SPE (see Figure 5). Our double buffering scheme allows SPEs to receive a new subject sequence through DMA while processing another previously received sequence. The PPE continuously prepares sequence data for free buffers. Once a buffer is filled, the PPE sends a mailbox notification to the corresponding SPE. The number of buffer in the PPE for each SPE is therefore restricted by the size of the SPE's Read Inbound Mailbox (which is four). Furthermore, the PPE dynamically assigns protein sequences to buffers depending on their lengths and the available memory. The maximum number of sequences inside a buffer is 32.

All sequences inside a buffer are filtered by Stages 1-3 on one of the SPEs. If a sequence passes all these stages, the corresponding bit in the matching signal (32 bits) is set. After all sequences are processed, this matching signal is sent back to the PPE via a mailbox. The PPE then identifies all sequences that have passed Stages 1-3 on SPE and perform Stages 1-4 on them.

Pseudocodes of the programs running on the PPE and each SPE are shown in Figures 6 and 7. Because of the limited storage of each SPE (256 KBytes) it is important to analyze the associated memory consumption. The size of SPE program is 100KByte. Thus, we have at most 156KByte for storing the DFA data structure, the two buffers as well as other parameters and intermediate results. Hence, we have assigned 10KByte to each buffer and up to 80KByte to the DFA. 80KByte is sufficient for DFAs for query sequences of up to 2000 base-pairs (bps). In our experiment, the average DFA size is 43.8KByte. If the length of a subject sequence is over 10Kbps, it will be put directly into the sequence queue of the PPE without sending it to an SPE. Furthermore, some database sequences exceed a certain memory threshold during they are processed on the SPE. Such sequences will be marked and passed to the PPE for further processing. Although, this creates additional work, the number of such sequences is usually negligible. It is also another reason why the PPE performs all stages of the BLASTP algorithm instead of only Stage 4. Further note, that we do not return results of matching sequences from SPEs because we do not want to increase SPE code size by increasing program complexity to return the search results.

```
1. Initialization
2. Create DFA
3. Start SPEs and send parameters and DFA lookup table to SPEs
4. Check whether there is mail from SPEs
   If there is a mail
      Collect information of sequences that passed stages 1-3 and keep in a queue
      Mark the corresponding buffer as free
5. Check whether there is a free buffer
   If a free buffer is found
      Prepare data into it and mark it as occupied
   Else
      Do BLASTP searching stages 1-2 for sequences in the queue
6. Repeat 4-5 until there is no sequence in database
7. Send commands to SPEs to complete last buffered sequences
8. Wait until all buffers are marked as free
9. Do BLASTP stages 3-4
```

Fig. 6. Pseudocode of the program running on the PPE

```
1. Initialization
2. Receiving parameters and DFA from PPE
3. Receiving mail with command from PPE
4. If command is new-data-available
     DMA the new data
     If this is the 1st data
        Goto 3
     Else
4.5     Wait for last data to be completely DMA transferred
        Do Stages 1-3 for sequences in the last data
4.7     Return matching signal to PPE through SPU Write Outbound Mailbox
        Goto 3
5. If command is finish-last-sequence
     Do Stages 1-3 for sequences in the last data
     Return matching signal to PPE through SPE Write Outbound Mailbox
     Exit
```

Fig. 7. Pseudocode of the program running on the SPE

5 Performance Evaluation

We have implemented the described Cell BE BLASTP program using CELL BE SDK 3.0 and evaluated it on a PlayStation®3 (PS3), which contains a Cell BE as its main processor. In order to evaluate the performance on a PS3, we have installed LINUX version 2.6.23-rc3 (gcc version 4.1.1 20061011 (Red Hat 4.1.1-30)). Please note that on the PS3 two of eight SPEs are used by the operating system running. Therefore, our experiments can only use up to six SPEs.

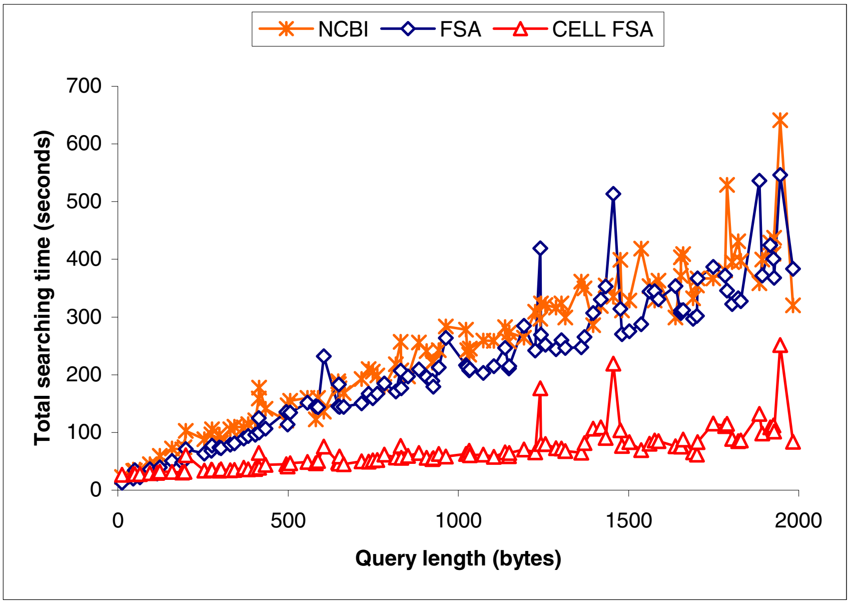


Fig. 8. Runtime comparison between FSA-BLASTP on a dual-core P4 3GHz and Cell BE BLASTP on a PS3 for varying query sequence lengths

We have compared the performance of our Cell BE BLASTP program to FSA-BLASTP (available form www.fsa-blast.org) and NCBI-BLASTP (www.ncbi.nlm.nih.gov/BLAST/developer.shtml). FSA-BLAST uses an optimized sequential algorithm and is around 15% faster than NCBI-BLASTP with no loss in accuracy [3, 4]. FSA-BLASTP and NCBI-BLASTP are tested on a HP workstation xw4200 with Dual-core Pentium®4 (P4) CPU 3GHz, 2GB of RAM. Two-hit model [2] is used for all BLASTP programs. Default values of W=3 and T=11 are adopted. The produced matching results by FSA-BLASTP and Cell BE BLASTP are exactly the same.

The protein sequence database we used in our experiments is the GenBank Non-Redundant Protein Database (downloaded from ftp://ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz), which contains 6,375,605 protein sequences. We have chosen 100 random sequences from the database as queries. The lengths of the query sequences are distributed uniformly between 1 and 2000bps.

A performance comparison of the presented parallel Cell BE BLASTP program to the sequential FSA-BLASTP and NCBI-BLASTP programs are shown in Figure 8. It can be seen that Cell BE BLASTP is faster than FSA-BLAST in most cases. The average searching times are 217.5s for FSA-BLASTP, 244.75s for NCBI-BLASTP, and 67.97s for Cell BE BLASTP. This corresponds to an average speedup of 3.2 and 3.6 respectively.

More detailed statistics are shown in Table 1. From Table 1, we can see that Cell BE BLASTP spends more time on Stage 4. This is because the PPE is a less powerful processor than a P4. The speedup of Cell BE BLASTP mostly comes from stage 1-3 which are running on six PPEs of the PS3 in parallel.

Table 1. More detailed runtime comparison (in seconds) between FSA-BLASTP and Cell BE BALSTP

Query length range	FSA-BLASTP				Cell BE BLASTP				Speedup
	Stages 1-2	Stage3	Stage4	Total	Stages 1-2	Stage 3	Stage 4	Total	
1-300	40.1	5.66	0.30	46.5	28.9	1.77	0.74	32.9	1.41
301-500	74.0	23.09	0.32	97.8	35.4	3.10	0.81	40.9	2.39
501-800	110.3	46.57	0.50	157.8	44.5	4.30	1.10	51.5	3.06
801-1100	151.0	50.98	0.92	203.4	52.8	4.74	1.83	61.1	3.33
1101-1400	183.0	76.32	1.80	261.6	61.8	10.25	4.18	79.0	3.31
1401-1700	216.9	109.01	3.22	329.6	67.2	15.19	7.98	92.4	3.57
1701-2000	241.8	141.53	2.02	385.9	83.9	18.77	4.57	109.0	3.54

Table 2. Average number of sequences processed by each stage of FSA-BLASTP on a P4 and by the PPE in Cell BE BLASTP

Query length	FSA-BLASTP			Cell BE BLASTP (only on PPE)			Matching output
	Stages1-2	Stage3		Stages1-2	Stage3		
		semi	gapped		semi	Gapped	
1-300	6,375,605	96954	9443	2113	2062	1731	328
301-500		334494	13749	2591	2570	1462	324
501-800		617225	19602	5480	5471	3713	443
801-1100		586139	24163	5408	5402	3569	471
1101-1400		761097	34028	7193	7189	5178	443
1401-1700		1096186	43616.1	15404	15402	12901	438
1701-2000		1206705	38761	6734	6733	4126	428

The numbers of sequences that are processed in each stage by FSA-BLASTP and in the PPE by Cell BE BLASTP are shown in Table 2. In FSA-BLASTP, every database sequence is processed by Stages 1-2. The PPE in Cell BE BLASTP only processes a very small faction of database sequences since most sequences have been filtered by SPEs in parallel. This reduced number of sequences contributes to the less total runtime of Cell BE BLASTP. However, the ideal speedup of around six is not reached since the parallel SPE filters add some data transfer and coordination overhead and the PPU is less powerful than a P4. It should also be noted that the speedup for shorter query sequences is generally lower since the runtime is too short to effectively compensate for the associated overheads.

Also note that for Cell BE BLASTP in Table 2, the number of database sequences is larger than the number of found matching sequences. This can be explained as follows. Firstly, if a sequence is too long to be sent to the SPE, it will be processed by the PPE directly. In the experiment, 72 sequences are longer than the maximum buffer length (10KByte). Secondly, some sequences in Stages 1-3 in the SPE exceed the maximum available memory space. These sequences are returned as matches and need further processing on the PPE.

Table 3. Runtime statistics (in seconds) of three exceptional sequences

Query length	Method	Time				
		Stages 1-2	Stage3		Stage4	Total
			semi	gapped		
605	FSA-BLAST	63.55	160.89	6.40	0.80	232.13
	Cell BE	58.65	11.63	1.85	1.88	75.61
1455	FSA-BLAST	138.97	348.58	0.38	24.96	513.43
	Cell BE	84.48	65.49	1.28	66.17	219.43
1945	FSA-BLAST	225.87	316.33	1.02	2.63	546.35
	CELL	132.11	109.53	1.36	6.98	251.52

Query length	Method	Number of sequences			Matching output
		Stages 1-2	Stage3		
			semi	gapped	
605	FSA-BLAST	6,375,605	1,890,358	33,061	500
	Cell BE	5,536	5,536	2,288	500
1455	FSA-BLAST	6,375,605	2,981,242	23,895	500
	Cell BE	8,344	8,344	4,115	500
1945	FSA-BLAST	6,375,605	1,555,474	170,541	500
	Cell BE	27,681	27,677	25,473	500

Figure 8 also shows that some query sequences require more processing time by both FSA-BLASTP and Cell BE BLASTP than queries of similar lengths. The statistics of the three such exceptional sequences is shown in Table 3. It can be seen that for these three queries, a bigger number of database sequences need to be processed than average. This increases both CPU and PPE workload.

6 Conclusion

In this paper, we have presented a parallel algorithm for accelerating BLASTP on a heterogeneous multi-core system. In order to exploit the characteristics of this type of architecture we have used a compressed deterministic finite state automaton for hit detection in order to reduce memory consumption and a double-buffered communication scheme. Our implementation achieves an average speedup of 3.2 compared to the optimized FSA-BLASTP and 3.6 compared to NCBI-BLASTP on a PS3, which is available for less than US$500 at most local computer outlets. The very rapid growth of biological sequence databases demands even more powerful high-performance solutions in the near future. Hence, our results are especially encouraging since high performance computer architectures are developing towards heterogeneous multi-core systems.

Acknowledgement

The work was supported by the A*Star BMRC Research Grant 04/1/22/19/375.

References

1. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. J. Mol. Biol. 215(3), 403–410 (1990)
2. Altschul, S.F., et al.: Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs. Nucleic Acid Research 25(7), 3389–3402 (1997)
3. Cameron, M., Williams, H.E., Cannane, A.: A Deterministic Finite Automaton for Faster Protein Hit Detection in BLAST. Journal of Computational Biology 13(4), 965–978 (2006)
4. Cameron, M., Williams, H.E., Cannane, A.: Improved Gapped Alignment in BLAST. IEEE Trans. Computational Biology and Bioinformatics 1(3), 116–129 (2004)
5. Jacob, A., Lancaster, J., Harris, B., Buhler, J., Chamberlain, R.: Mercury BLASTP: Accelerating Protein Sequence Alignment. ACM Transactions on Reconfigurable Technology and Systems(to appear, 2008)
6. Kahle, J.A., et al.: Introduction the Cell multiprocessor. IBM Journal of Research and Development 49(4/5), 598–604 (2005)
7. Kent, W.J.: BLAT – The BLAST-like Alignment Tool. Genome Research 12(4), 656–664 (2002)
8. Kistler, M., Perrone, F., Petrini, F.: Cell multiprocessor communication network: built for speed. IEEE Micro. 26(3), 10–23 (2006)
9. Li, M., Ma, B., Kisman, D., Tromp, J.: Patternhunter II: Highly Sensitive and Fast Homology Search. J. Bioinformatics and Computational Biology 2(3), 417–439 (2004)
10. Oehmen, C., Nieplocha, J.: ScalaBLAST: A scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis. IEEE Transactions on Parallel and Distributed Systems 17(8), 740–749 (2006)
11. Pham, D., et al.: The design and implementation of a first-generation Cell processor. In: Proceedings IEEE ISSCC 2005, San Francisco, CA, pp. 184–185 (2005)
12. Sachdeva, V., Kistler, M., Speight, E., Tzeng, T.H.K.: Exploring the viability of the Cell Broadband Engine for bioinformatics applications. In: 6th IEEE International Workshop on High Performance Computational Biology (HiCOMB 2007), Long Beach, CA (2007)
13. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
14. Wirawan, A., Kwoh, C.K., Schmidt, B.: Parallel DNA Sequence Alignment on the Cell Broadband Engine. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967. Springer, Heidelberg (to appear, 2008)