Applying SIMD Approach to Whole Genome Comparison on Commodity Hardware

Arpith Jacob[1], Marcin Paprzycki[2,3], Maria Ganzha[2,4], and Sugata Sanyal[5]

[1] Department of Computer Science and Engineering
Vellore Institute of Technology, Vellore, India
arpith@arpith.com
[2] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
{Marcin.Paprzycki,Maria.Ganzha}@ibspan.waw.pl
[3] Computer Science Department, Warsaw Management Academy, Warsaw, Poland
[4] Department of Administration, Elblag University of Humanities and Economics, Elblag, Poland
[5] School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India
sanyal@tifr.res.in

Abstract. Whole genome comparison compares (aligns) two genome sequences assuming that analogous characteristics may be found. In this paper, we present an SIMD version of the Smith-Waterman algorithm utilizing Streaming SIMD Extensions (SSE), running on Intel Pentium processors. We compare two approaches, one requiring explicit data dependency handling and one built to automatically handle dependencies and establish their optimal performance conditions.

1 Introduction

Sequence similarity searches are frequently performed in Computational Biology. They identify closely related genetic sequences, assuming that high degree of similarity often implies similar function or structure. To establish similarity an *alignment score* is calculated. Exact algorithms to calculate the alignment score, based on the dynamic programming, are very slow (even on fastest workstations). Hence, heuristic alternatives are used; but they may not be able to detect distantly related sequences. Among exact methods, the Smith-Waterman algorithm is one of the most popular. However, due to its compute-intensive nature, it is rarely used for large scale database searches. In this paper we describe an efficient implementation of the Smith-Waterman algorithm that exploits the fine-grained parallelism. We use Intel's MMX/SSE2 SIMD extensions to speedup the algorithm within a single processor.

2 Related Work

Parallelization of the Smith-Waterman algorithm proceeds on two fronts: fine-grained and coarse-grained parallelism. In the fine-grained approach the pairwise

R. Wyrzykowski et al. (Eds.): PPAM 2007, LNCS 4967, pp. 1220–1229, 2008.

comparison algorithm is parallelized and each processing element performs a part of matrix calculation to help determine the optimal score. This approach was widely used on single instruction multiple data parallel computers at the time when they were very popular. For multiple instruction, multiple data computers, subsets of the database are independently searched by processing elements.

Five architectures used for sequence comparison were described in Hughey [1]: (1) special purpose VLSI, (2) reconfigurable hardware, (3) programmable co-processors, (4) supercomputers and (5) workstations. Special purpose VLSI provides the best performance but is limited to a single algorithm. The Biological Information Signal Processing (BISP) system was one of the first systolic arrays used for sequence comparison. Reconfigurable hardware is typically based on Field Programmable Gate Arrays (FPGAs). They are more versatile than special purpose VLSI and can be adapted to different algorithms. A number of FPGA systems, such as DeCypher [3], accelerate Smith-Waterman algorithm, reporting several orders of magnitude speedup. These systems can easily be ported to newer generations of FPGAs with only a minimum re-design. Programmable co-processors strive to balance the flexibility of reconfigurable hardware with the speed and high density of processing elements. Kestrel [19] is a 512 element array of 8-bit PEs that was used for sequence alignment. For high performance computers let us mention the BLAZE [4], an implementation of the Smith-Waterman algorithm, written for the SIMD MasPar MP1104 computer with 4096 processors. As far as workstations are concerned, Wozniak [5] presented an implementation that used the SIMD visual instruction set of Sun UltraSparc microprocessors to simultaneously calculate four rows of the dynamic programming matrix. Rognes and Seeberg [6] used the SIMD multimedia extension instructions on Intel Pentium microprocessors to produce one of the fastest implementations on workstations. Networks of workstations have also been used effectively by Strumpen [7], who utilized a heterogeneous environment consisting of more than 800 workstations, while Martins and colleagues [8] presented an event-driven multithreaded implementation of the sequence alignment algorithm on a Beowulf cluster consisting of 128 Pentium Pro microprocessors.

3 The Smith-Waterman Algorithm

Initially, Needleman and Wunsch [9] and Sellers [10] introduced the global alignment algorithm based on the dynamic programming approach. Smith and Waterman [11] proposed an $O(M^2N)$ algorithm to identify common molecular subsequences, which took into account evolutionary insertions and deletions. Later, Gotoh [12] modified this algorithm to run at $O(MN)$ by considering affine gap penalties. These algorithms depended on saving the entire $M \times N$ matrix in order to recover the alignment. The large space requirement problem was solved by Myers and Miller [13] who presented a quadratic time and linear space algorithm, based on a divide and conquer approach. Finally, Aho, Hirschberg and Ullman [14] proved that symbol comparing algorithms (to see if they are equal or not), have to take time proportional to the product of their string lengths.

		T	C	G	A	C	A	T	A
	0	0	0	0	0	0	0	0	0
A	0	0	0	0	5	0	5	0	5
C	0	0	5	0	0	10	3	1	0
T	0	5	0	1	0	3	6	8	1
A	0	0	1	0	6	0	8	2	13
G	0	0	0	6	0	2	1	4	6
G	0	0	0	5	2	0	0	0	0
C	0	0	5	0	1	7	0	0	0
A	0	0	0	1	5	0	12	5	5

Fig. 1. Comparison Matrix: Optimal score: 13, Match: 5, Mismatch: -4, Penalty: $0+7k$. Optimal Alignment: A C A T A, A C - T A.

Let us now describe the Smith-Waterman algorithm (an example of its operation was depicted in Figure 1). Let us consider two genomic sequences A and B of length M and N respectively, to be compared using a substitution matrix ∂, and utilize the affine gap weight model. The gap penalty is given by: $W_i + kW_e$ where $W_i > 0$ and $W_e > 0$. W_i is the penalty for initiating the gap and W_e is the penalty for extension of the gap, which varies linearly with the length of the gap. The substitution matrix ∂ lists the probabilities of change from one "structure" into another in the sequence. There are two families of matrices used in the algorithm: the Percent Accepted Mutation (PAM) and the Block Substitution Matrices (BLOSUM). Maximization relation is used in order to calculate the optimum local alignment score according to the following recurrence relations (the highest value in the H matrix gives the optimal score):

$$E(i,j) = H(i,j) = F(i,j) = 0, \quad \text{for } i = 0 \text{ or } j = 0$$

$$E(i,j) = \max \left\{ \begin{matrix} E(i-1,j) - W_e \\ H(i-1,j) - W_i - W_e \end{matrix} \right\}$$

$$F(i,j) = \max \left\{ \begin{matrix} F(i,j-1) - W_e \\ H(i,j-1) - W_i - W_e \end{matrix} \right\}$$

$$H(i,j) = \max \left\{ \begin{matrix} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) + \partial(A_i, B_j) \end{matrix} \right\}$$

These recurrences can be understood as follows: the E (F) matrix holds the score of an alignment that ends with a gap in the sequence A (B). When calculating the $E(i,j)$th ($F(i,j)$th) value, both extending an existing gap by one space, or initiating a new gap is considered. The $H(i,j)$th cell value holds the best score of a local alignment that ends at position A_i, B_j. Hence, alignments with gaps in either sequence, or the possibility of increasing the alignment with

a matched or mismatched pair are considered. A zero term is added in order to discard negatively scoring alignments and restart the local alignment. One of possible many optimal alignments can be retrieved by retracing steps taken during computation of matrix H, from the optimal score back to the zero term.

To quantify the performance of dynamic programming algorithms, the measure: *millions of dynamic programming cell updates per second* (*MCUPS*) has been defined. It represents the number of cells in the H matrix computed per second, and includes all memory operations and corresponding E/F matrix cell evaluations.

4 SIMD-Based Approach

Multimedia extensions have been added to the Instruction Set Architectures (ISAs) of most microprocessors [15]. They exploit low-level parallelism, where computations are split into subwords, with independent units operating on them simultaneously (a form of SIMD parallelism). Intel introduced the Pentium MMX microprocessor [16] in 1997. The MMX (MultiMedia eXtensions) technology aliased the eight 64-bit MMX registers with the floating point registers of the x87 FPU, allowing up to eight byte operations performed in parallel. SIMD processing was enhanced with the addition of the SSE2 (Streaming SIMD Extensions) in the Pentium 4 microprocessor. It allows handling sixteen simultaneous byte operations in 128-bit XMM registers. Note however that because of the smaller number of available bits, overflows or underflows occur more frequently. They are handled by two methods: *wraparound arithmetic*, which truncates the most significant bit; and *saturation arithmetic*. In the latter case the result saturates at an upper or lower bound and the result is limited to the largest/smallest representable value. Hence, for unsigned integer data types of n bits, underflows are clamped to 0, and overflows to $2n - 1$. *Saturation arithmetic* is advantageous because it offers a simple way to eliminate unneeded negative values and automatically limits results without causing errors, and thus is used in our implementation. Finally, let us note that compiler support for SIMD instructions is still somewhat rudimentary, and thus hand coding in assembly language using SIMD instructions is often required.

4.1 Challenges in Parallelizing the Smith-Waterman Algorithm

Parallelizing the dynamic programming algorithm is done by calculating multiple rows of the H matrix simultaneously. Figure 2 shows the data dependencies of each cell in the H matrix. Value in the $(i, j)^{th}$ cell depends on $(i - 1, j - 1)^{th}, (i-1, j)^{th}$ and $(i, j-1)^{th}$ cell values. Hence, before a cell can be computed, cells immediately above, to the left and diagonally across must be available. This figure also shows why a systolic-array would be ideal for this type of calculations. The alignment matrix H, can be evaluated in parallel rows (columns) or anti-diagonals.

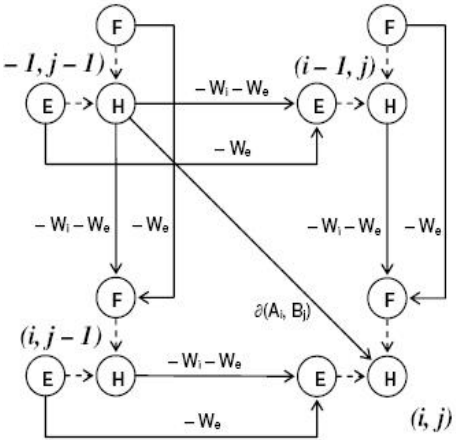

Fig. 2. Data dependencies in the similarity matrix

4.2 Diagonal Approach

When computation proceeds diagonally across the alignment matrix the interdependencies are automatically handled. The main disadvantage is that substitution scores cannot be accessed linearly from memory, but have to be independently loaded for each diagonal cell. Symbols from two sequences have to be read, and a look-up into the substitution table made in order to calculate the corresponding match or mismatch score. This procedure has to be repeated for every element in the diagonal before parallel computation can proceed. The second disadvantage is that the size of the diagonal varies at the beginning and the end of the matrix sweep. For example, if parallel computation involves four cells at a time, the first three diagonals of the alignment matrix with one, two and three cells respectively do not contain enough cells to load the 4-way SIMD word. To solve this problem three dummy symbols both at the beginning and the end of the query sequence are added. Furthermore, appropriate entries must be added in the similarity table between the dummy symbol and each symbol in the alphabet (the dummy symbol included), with a score of zero, so that the optimal score remains unchanged. Computation then proceeds along successive diagonals through the entire length of the query sequence. The next four rows of the matrix are then computed in the same manner. If the database sequence length is not a multiple of the SIMD word length, the sequence must be concatenated at the end with an appropriate number of dummy symbols. This process is illustrated in figure 3 (panel a). SIMD operations can be performed with signed or unsigned integers. To avoid reducing the maximum representable integer value all elements in the similarity matrix are biased by a positive value (forcing them to remain positive). The bias is then subtracted without affecting the optimal score.

In order to obtain optimum performance, a number of techniques have been used to speedup the code. (1) To optimize utilization of cache memory arrays E and H are interleaved as an array of structures. (2) SIMD words in memory

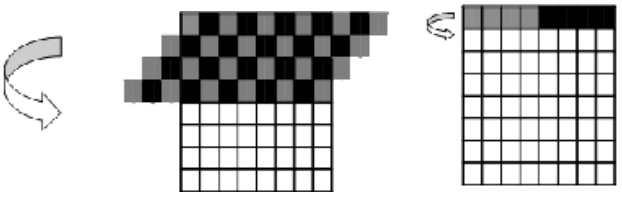

Fig. 3. Fine-grained parallelization of the Smith-Waterman algorithm using 4-way subword processing. a. Diagonal approach. b. Horizontal approach.

are aligned at appropriate boundaries: 64-bit memory access with MMX registers requires the target address to be aligned at 8 byte boundaries and 128-bit memory access with XMM registers requires alignment at 16 byte boundaries. (3) to reduce effects of memory latency memory references for subword access (one addition, one multiplication and two memory reference instructions in the outer loop, along with three memory read instructions in the inner loop) can be appropriately re-arranged.

4.3 Horizontal Approach

When computation proceeds horizontally along the rows of the alignment matrix the interdependencies are not resolved. To calculate the value of the $(i,j)^{th}$ cell in the H matrix, the values of the $(i,j-1)^{th}$ cell in the F and H matrices are required and thus parallel calculation of horizontal cells of H is impossible.

An interesting empirical observation concerning utilization of the Smith-Waterman algorithm for biological sequences was made by Phil Green (and implemented in the SWAT program, [18]). In most cells of E, F and H matrices, values are clamped to zero (when using saturated arithmetic) and thus do not contribute to H. Specifically, the $(i,j)^{th}$ cell value in the F matrix will remain zero if the $(i,j-1)^{th}$ cell value is already zero, as long as $H(i,j-1) \leq W_i + W_e$.

$$F(i,j) = \max \begin{Bmatrix} F(i,j-1) - W_e \\ H(i,j-1) - W_i - W_e \end{Bmatrix}$$

If the H value is below this threshold, F will remain zero within that row. For example, when using a 4-way SIMD word, the F values can be ignored from the iteration if the four H values in its relation are below the threshold $W_i + W_e$. If one or more of the H values exceeds this threshold, the F values must be recalculated sequentially. This effect depends on the threshold value. If the gap open and gap extend penalties are very small, most H values are above the threshold and there will be no speedup in the algorithm. On the other hand if the threshold values are too large results will be "incorrect" as useful information may be lost.

An advantage of the horizontal method is that substitution scores can be loaded with a single memory read operation using a query sequence profile table. The query sequence profile table contains the substitution scores of the query

sequence placed horizontally across the matrix, versus an imaginary sequence made up of all symbols in the alphabet and is created once for the query sequence. This process is illustrated in Figure 3 (panel b). Note that most optimization techniques used in the diagonal method are relevant here. The query sequence profile table is computed once before the database comparison procedure and is usually small enough to fit in the first level cache of the microprocessor. The conditional loop presents a problem because it is cumbersome to implement using Intel's media processing ISA. Further, it increases the runtime because of the possibility of misprediction of the branch target address. Thus, the SIMD conditional loop is unrolled.

4.4 Experimental Results

The two algorithms were implemented using MMX and SSE2 technology and tested on a Pentium III 500Mhz with 128MB RAM, running Windows 2000, and a Pentium 4, 1.4Ghz with 128MB RAM, running Windows NT. Finally, we have run two series of experiments on the Intel Pentium 4, 2.80GHz with 1GB of RAM, running Windows XP. The user interface, file handling and memory allocation code was written in C and compiled using the Visual C/C++ 6.0 compiler. The Smith-Waterman algorithm was written in assembly language and compiled using the Netwide ASeMbler 0.98.08 (NASM). Timings were measured by reading the microprocessor timestamp (using the assembly mnemonic RDTSC) before and after completion of the target function and dividing by the microprocessor clock speed in Hz. For each test, the total program runtime, total I/O overhead, total time spent in the Smith-Waterman function, MCUPS, and their averages were noted.

In the tests, the local alignment score between two DNA sequences was calculated without recovering the alignment. The pam47 substitution matrix was used which assigns a value +5 for a match and -4 for a mismatch between two nucleotides. A bias value of +4 was used to eliminate negative elements from the substitution matrix. An affine function $0 + 7k$ was used for the gap open and gap extension penalties.

Tests were performed using query sequences ranging in length from 100 to 1000 nucleotides; in steps of 100. We used the annotated Drosophilia genome release 3.0 [17], containing 17,878 sequences with a total of 28,249,452 nucleotides.

Plots of search times versus query lengths for different SIMD implementations are shown in Figure 4. The bulk of the program time (96-97%) is spent in the Smith-Waterman sequence comparison function. Only a small percent of the time is spent as overhead for reading the sequences from the disk. For a gap penalty of $0 + 7k$, the diagonal method was found to be 1.30 to 1.87 times faster than the horizontal method. Using the 128-bit XMM registers on processors with SSE2 technology doubles the size of the SIMD word as compared to the 64-bit MMX registers of the older MMX technology. Theoretically, this should result in a two fold speed increase. Practically, the speedups ranged from 1.17 to 1.40 as with an increase in the SIMD word length there is a corresponding increase

in clock cycles. Surprisingly, the horizontal approach using the byte precision within the SSE2 technology was slower than its MMX implementation by 14%.

As expected, searches using 8-bit subwords in the SIMD word, as compared to searches using subwords of 16-bits, were found to be faster by a factor of 1.31 to 1.79. Most comparison scores in sequence searches are well below the maximum value representable in 8 bits. Any score close to 255 represents an interesting match which is investigated by other means, irrespective of its actual score. Hence in most cases, byte precision is sufficient for database searching.

Another interesting observation is the scalability of the SIMD implementation between processors in the same family. The diagonal approach using MMX technology results in a performance boost of 3.68 and 3.91 for the byte and word precisions respectively. The horizontal approach achieved more modest speedups of 1.44 and 1.63 for the byte and word precisions. Finally, we have found out that move from the 1.4 GHz Pentium 4 to the 2.8 GHz Pentium 4 resulted in the MCUPS rate (for query length 1000) to jump from 215 to 488 for the diagonal method, and from 165 to 373 for the horizontal approach. While we do not have a direct explanation of the jump that is more than two-fold; let us note that both machines have been running different operating systems and had substantially different amounts of available memory. What matters is the fact that for the algorithm in question the clock-speed of the processor (and thus its raw power) translates directly into performance.

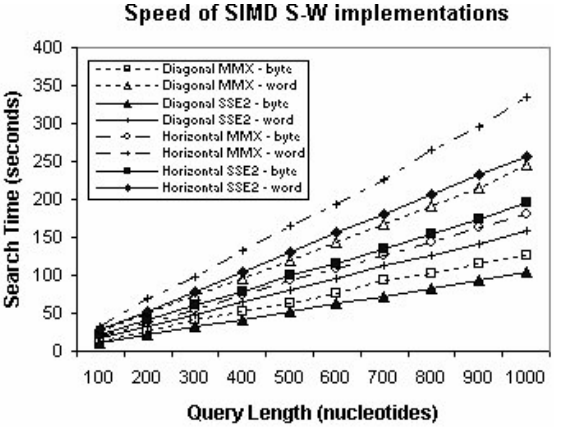

Fig. 4. Search times versus query lengths for different implementations

To complete our investigation, in figure 5 we depict effect of gap penalties on the horizontal method. Searching the database using a query of length 900 with a gap penalty of $0+7k$ takes 174 seconds, while a search with a penalty of $40+2k$ takes only 43.9 seconds (however, with the change in the gap penalty the optimal scores are no longer equal). The speed of the horizontal method varies from 142 MCUPS with a gap penalty of $0 + 7k$, to its saturation point at approximately

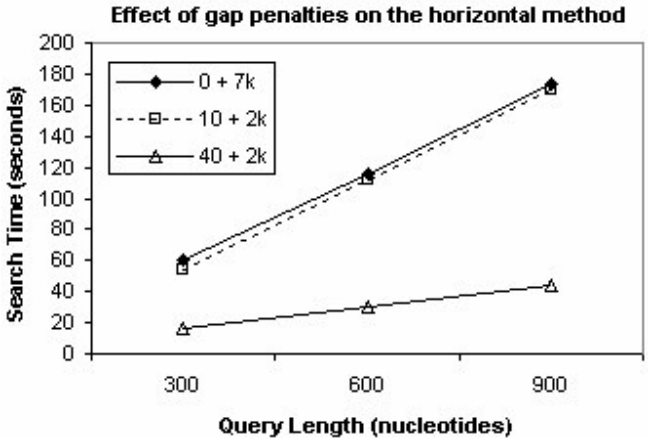

Fig. 5. Effect of SWAT optimization

580 MCUPS with a gap penalty of $40 + 2k$. The diagonal method, on the other hand, does not incorporate the SWAT optimization and runs at constant speeds for different gap penalties. Hence, database searching with a high gap penalty favors the horizontal method over the diagonal method (as long as the final results remain reliable).

5 Concluding Remarks

The aim of this work was to propose a fine grain implementation of the Smith-Waterman algorithm utilizing multimedia extensions on Intel processors. Our experiments showed significant speedups on Pentium workstations. Since the general-purpose microprocessors are constantly being updated with more advanced features allowing micro-parallelization of algorithms, it can be expected that features like the newly introduced Simultaneous Multi-threading (hyper-threading) technology will offer further potential for performance increase. However, note that this and other similar approaches heavily rely on assembly coding. As a results codes are not portable at all. For instance our codes run only on 32-bit architectures and are useless for the modern 64-bit processors.

Acknowledgments

We thank the manager of Centre for Technical Support at T. S. Santhanam Computing Centre, Muthu'G., and the laboratory personnel Mr. Ravikumar V. (GA), Mr. Elson Jeeva T. (MIS), Mr. Srinivasan V. and Ms Dharani (ME) for providing equipment required to conduct experiments.

References

1. Hughey, R.: Parallel hardware for sequence comparison and alignment. Computer Applications in the Biosciences 12(6), 473–479 (1996)
2. Yamaguchi, Y., Maruyama, T., Konagaya, A.: High Speed Homology Search with FPGAs. In: Pacific Symposium on Biocomputing, pp. 271–282 (2002)
3. http://www.timelogic.com
4. Brutlag, D.L., Dautricourt, J.P., Diaz, R., Fier, J., Moxon, B., Stamm, R.: BLAZE: An implementation of the Smith-Waterman Comparison Algorithm on a Massively Parallel Computer. Computers and Chemistry 17, 203–207 (1993)
5. Wozniak, A.: Using video-oriented instructions to speed up sequence comparison. Computer Applications in the Biosciences 13(2), 145–150 (1997)
6. Rognes, T., Seeberg, E.: Six-fold speed-up of Smith Waterman sequence database searches using parallel processing on common microprocessors. Bioinformatics 16(8), 669–706
7. Strumpen, V.: Parallel molecular sequence analysis on workstations in the Internet. Technical report, Department of Computer Science, University of Zurich (1993)
8. Martins, W.S., del Cuvillo, J.B., Useche, F.J., Theobald, K.B., Gao, G.R.: A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. In: Proceedings of the Pacific Symposium on Biocomputing, pp. 311–322 (2001)
9. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two sequences. J. of Molecular Biology 48(3), 443–453 (1970)
10. Sellers, P.H.: On the theory and computation of evolutionary distances. SIAM J. of Applied Mathematics 26, 787–793 (1974)
11. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. of Molecular Biology 147(1), 195–197 (1981)
12. Gotoh, O.: An improved algorithm for matching biological sequences. Journal of Molecular Biology 162(3), 705–708 (1982)
13. Myers, E.W., Miller, W.: Optimal alignments in linear space. Computer Applications in the Biosciences 4(1), 11–17 (1988)
14. Aho, A.V., Hirschberg, D.S., Ullman, J.D.: Bounds on the complexity of the longest common subsequence problem. J. of the ACM 23(1), 1–12 (1976)
15. Lee, R.B.: Multimedia extensions for general-purpose processors. In: Proceedings IEEE Workshop on Signal Processing Systems, pp. 9–23 (1997)
16. Peleg, A., Wilkie, S., Weiser, U.: Intel MMX for multimedia PCs. Communications of the ACM 40(1), 25–38 (1997)
17. Berkeley Drosophila Genome Project (2003),http://www.fruitfly.org/sequence/sequence_db/na_wholegenome_CDS_dmel_RELEASE3.FASTA.gz
18. http://www.phrap.org/phredphrap/swat.html
19. Di Blas, A., Dahle, D.M., Diekhans, M., Grate, L., Hirschberg, J.D., Karplus, K., Keller, H., Kendrick, M., Mesa-Martinez, F.J., Pease, D., Rice, E., Schultz, A., Speck, D., Hughey, R.: The UCSC Kestrel Parallel Processor. IEEE Transactions on Parallel and Distributed Systems 16(1), 80–92 (2005)