Abstract: We present hardware and software versions of our general purpose sequence search and alignment algorithm DASH (Diagonal Aggregating Search Heuristic) [1, 2, 3] in both, and comparison with NCBI-Blast [4], PatternHunter [5], BLAT [6] and Smith-Waterman [7] optimal alignment, performed with 400 complete DNA and Protein sequences randomly selected from the Human genome and Genpept databases, respectively. We show that DASH improves the sensitivity and gives an order of magnitude speed improvement over NCBI-Blast, DASH is also twice as fast as BLAT and has much greater sensitivity. DASH-H has achieved Smith-Waterman sensitivity, but with a $10^2$ to $10^3$ speed-up over Smith-Waterman in hardware. The corresponding speed-up over Smith-Waterman is software is several orders of magnitude on top of this. DASH-H has also achieved a speed-up of $10^3$ times NCBI-BLAST in software, but with greater sensitivity.

Key-Words: Sequence Alignment, BLAST, Smith-Waterman, DNA, Protein, BLAT

1 Introduction

The first successful gapped sequence alignment algorithm was due to Smith-Waterman [7], they formulated the gapped alignment problem as a finite optimization problem which was solved by dynamic programming. Although database sizes have increased such that Smith-Waterman is no longer practical to use, it is helpful as a base line by which to measure both the performance and quality of heuristic algorithms. In this paper we benchmark all the algorithms we have tested against it as it does provide the optimal solution, and practically all newer heuristic sequence alignment algorithms are derived from it in one way or another. As this algorithm is quite simple and regular, a number of authors have developed hardware architectures to solve the speed issues. In fact, several such products are commercially available.

The most successful algorithms used currently, such as NCBI-BLAST [4], make use of heuristics to reduce the search space by several orders of magnitude, and are indispensable for searching today's large genomic and proteomic databases. However, a significant contributor to the search time for such algorithms is still the dynamic programming evaluations, indeed BLAST spends around three quarters of its time budget in this area.

Genomic databases are already growing faster than Moore's law, so that processor speeds will not be able to keep up. If more aggressive heuristics are devised to make up for this fact, then users will have to expect less sensitivity. Evidence of this is that slower and (generally) more sensitive algorithms, have already been largely abandoned by users due to their speed penalty. The specific goal of the DASH project is to revisit the search problem with the intent of raising the sensitivity bar, while maintaining or improving speed. It is our intention to develop a general purpose sequence search and alignment tool which represents a tangible and compelling advance in the state of the art, therefore easing the problem of false negatives in heuristic algorithms.

In a series of articles we have developed a novel sequence alignment algorithm, DASH, which we have shown to be superior to BLAST in both speed and sensitivity [2, 1], and have excellent sensitivity with respect to Smith-Waterman. Our project was originally motivated by the desire to develop an algorithm which would allow extensive parallelism for optimum use of reconfigurable hardware, such as FPGAs. In this context we designed DASH around the principle of considering genomic and proteomic sequence alignments to typically consist of regions of high homology interspersed with regions of low homology, Figure 1. Finding the regions of high homology (the diagonals, zone 1 in the figure) requires high throughput, but low pro-
cessing power, which is ideally matched to the massively parallel resources available on FPGAs (Field Programmable Gate Arrays). The regions of low homology, zones 2 and 3 in the figure, are solved by DP, but as they are strictly limited in size, the extra hardware resources required is small.

In the software variant of the algorithm the common practice of using an indexed version of the database to approximate the exhaustive search which DASH-H performs is applied in order to obtain attractive run times. The speed gained by the use of the hashed index structure is in return for a relative degradation of sensitivity to alignments of particularly low homology, which we briefly explore when comparing the relative performance of DASH with other algorithms.

2 The DASH Algorithm

The DASH algorithm is based on the assumption that sequence alignments generally consist of alternating regions of high and low homology. This approach is consistent with the understanding that gene loci frequently contain a number of coding regions which are spliced together by transcription factors. Further, for more distant homologs it is normal to expect that there will be a mixture of more and less conserved regions, often corresponding with functional domains, particularly in the case of coding regions.

2.1 Three Stage Alignment Procedure

The DASH algorithm separates the process of identifying each sequence alignment into three main stages:

1. Identification of the seeds
2. Joining the seeds to form gapped alignments
3. Completing gapped alignments by performing dynamic programming at the extremities of each gapped alignment

These stages are illustrated in Figure 1. Three seeds are identified first (1), the two small regions of dynamic programming are evaluated to discover how to join the seeds (2). Finally, banded dynamic programming is performed at each end (3) to complete the alignment process.

The basic technique for discovering seeds is to looking up each \( k \)-mer of the subject sequence. Un-gapped extension is attempted at the location specified by each entry in the appropriate index. Any seed which is shorter than some threshold length is rejected. Generally, \( k = 8 \) for nucleotide, and \( k = 3 \) for protein. Extension continues as long as three of the four bases being compared correspond, in other words, if we denote a 1 for a match and a 0 otherwise, then we extend the match as long as the match pattern of the next four bases is \((1111, 0111, 1011, 1101, 1110)\). This approach is similar to PatternHunter [5], which uses a set of binary match patterns to find seeds of a specified length. For example, a match pattern of \(11010010100110111\) would give a seed of length 18.

Our technique is far simpler to implement, faster, and we show in the following sections that it improves sensitivity. Further, our algorithm is ideally suited to hardware implementation (see next section). Multiple groups of four bases can be matched in parallel by using look-up tables.

The dynamic programming effort of our approach contrasts strongly with Smith-Waterman, Fast-A/P and NCBI-BLAST. Smith-Waterman performs dynamic programming for the entire space. Fast-A/P performs dynamic programming for a band of the space. BLAST performs dynamic programming around the HSP for a score limited band. DASH performs dynamic programming only for the inter-alignment gaps and the extremities. In this way DASH is able to greatly reduce the time allocated to dynamic programming, and use this extra time budget to increase sensitivity. The general approach of localising dynamic
programming effort is not unique to DASH, for example [6, 8, 9, 5] use similar (though not identical) schemes.

2.2 Overview of Hardware Architecture

The major difference between the hardware and software flavors of DASH is in the identification of the seeds. Our hardware matches every query residue against every base residue, and so the parallelism offered by hardware is used to efficiently identify seeds of a lower homology than is possible in a similar number of processor cycles in software. Practically, this translates to the hardware being able to identify seeds which meet the score thresholds but do not necessarily contain highly conserved spans. This allows the hardware algorithm to provide significantly improved sensitivity to low-homology alignments over the current breed of fast software heuristic algorithms.

The architecture of our design is given in Figure 2. The query and subject sequences are input to a match pipeline 3, which finds the seeds, each individual match unit checks four bases in the nucleotide case and one amino acid in the protein case. A hit continues while three out of four bases match, or until two consecutive mismatches in the protein case. Substitutions, but not gaps, are allowed at this point. Those seeds (represented by query & subject start addresses and match length) of more than a specified length and score are then output to a FIFO (first-in first-out buffer). From there these prospective seeds are input into the Score unit, to be scored and output to a Stack (LIFO). The Dynamic Programming (DP) units then take the seeds from this Stack and find the optimum path for the inter-diagonal regions, zone 2, and the ends, zone 3. The raw input data is filtered by the Match Unit at high-speed giving the Score and DP Units more than enough time to process the inter-diagonal regions. Given sufficient on-board memory all these units can operate in parallel, and the critical overall system delay will be the throughput of the Match Unit. There are several well-known architectures for the processing elements (PE) in the DP Unit for Smith-Waterman in hardware, [10]. However, in our case the DP areas are not fixed but depend on the locations of the seeds and the score matrices have to be loaded dynamically into each PE, so a different architecture has been designed to accommodate this.

Figure 3: Match Unit Pipeline

3 Method

The DASH and DASH-H algorithms were compared with NCBI-BLAST and BLAT using 200 complete sequences randomly selected from a draft of the Human Genome for nucleotide (database size $2 \times 10^9$ bases) and from a GenPept release (database size $5 \times 10^8$ acids). Comparison was also made using the academic version of PatternHunter for the nucleotide searches. Except for the academic version of PatternHunter, which is available only for the Windows platform, all tests were performed on a 1.8GHz AMD Opteron processor with 8GB of main memory running RedHat Enterprise Linux using 32bit binaries. Due to the differing methods of measuring time expended by a process of Windows and UNIX systems, a direct comparison of run times between PatternHunter and the other algorithms was not possible.

The DASH-H algorithm was implemented in VHDL. DASH-H results were simulated by using the software version of DASH with parameters which approximate the hardware design, for example, allowing shorter seeds with longer non-conserved regions. However, in order to produce the simulations in a timely manner the hashed index was still used. This means that the DASH-H results presented will be less sensitive than the final hardware device since the index prevents the detection of seeds which lack a well conserved region of sufficient size. Therefore the results for DASH-H presented here represent something of a lower bound of the final expected sensitivity.

Sensitivity comparisons among the algorithms were made using the method described in [5] where the number of alignments where a given algorithm identifies at least 50% of an alignment identified by a complete Smith-Waterman approach is tallied. By considering the top 100 of the Smith-Waterman results scores will range from 0 to 100 inclusive, with higher scores being better. A second comparison was also performed, using a less lenient method of counting the number of consecutive alignments an algorithm identifies 100% of, beginning from the highest scoring alignment returned by Smith-Waterman. This measure
will return a score which corresponds to the rank of the first alignment returned by the Smith-Waterman algorithm, but which a given heuristic algorithm was not able to identify completely. This metric is particularly hard on the dynamic programming stage of heuristic algorithms and recall of high-homology alignments, whereas the PatternHunter style metric is more sensitive to recall of low-homology alignments.

To provide a fair comparison of DASH with NCBI-BLAST and BLAT, those algorithms were run repeatedly, first with the default settings, and then with parameter selections intended to reflect DASH. For NCBI-BLAST this involved disabling the filtering of query sequences, while with BLAT it was appropriate to use the same index width as DASH. Also, for NCBI-BLAST protein searching, blastpgp (from the NCBI-BLAST distribution) was used with the -s option to perform Smith-Waterman optimal dynamic programming.

4 Results and Conclusions

4.1 Serial (Software) Algorithm

Which ever measure is used, the software implementation of the DASH algorithm performs favorably compared to BLAT and PatternHunter both in terms of speed and sensitivity. In all cases, even the fastest...
mode of DASH obtains significantly higher sensitivity scores than either of BLAT or PatternHunter, as indicated in the Figures. The margins are especially large when using the sudden death metric which taxes the ability of an algorithm to consistently and reliably return the full extent of the highest scoring alignments. This trend extends to include the gap between BLAST and DASH, even when query sequence filtering is disabled (BLAST no filtering), or Smith-Waterman optimal extension is performed (blastpgp -s). This may suggest that the dynamic programming implementation in DASH is more effective than in NCBI-BLAST, BLAT or PatternHunter, while requiring only a fraction of the time. Indeed, DASH operates up to an order of magnitude faster than NCBI-BLAST for nucleotide or protein alignment or BLAT protein alignment, and around twice as fast as BLAT for nucleotide alignment.

It is worth noting that while DASH outperforms the BLAT and PatternHunter heuristic algorithms in sensitivity, and matches BLAST for high and low homology nucleotide searching and high-homology protein searching, it still trails BLAST at detecting low-homology protein alignments. This appears to be a fundamental limit of using strict hashed indices for protein alignment where significant homology frequently exists without exactly conserving any substantial run of sequential acids. A different approach is required to further accelerate low-homology protein alignment without sacrificing sensitivity.

### 4.2 Parallel (Hardware) Performance

The Match, Score and DP Units were coded in VHDL and our synthesis results show that on a system with 3 Spartan3 Xilinx FPGAs (XC3S5000-5FG900C) allocated to the Match units and one to the Score and DP units, we would obtain a throughput of $4 \times 10^{12}$ base comparisons/sec. With four such boards a typical query of up to 500 bases could be searched against all of Genbank in 2 seconds. This is comparable to performing the same query with NCBI-BLAST on the 128 node IBM 1350 eServer supercomputer at SAPAC*, at better than one-hundredth the cost, and with greater sensitivity. The equivalent throughput in the protein case is $7 \times 10^{11}$ acid matches/sec. This approach is also much faster and more sensitive than current hardware accelerators, such as TimeLogic’s DeCypher (www.timelogic.com).

These search speeds compare with NCBI-BLAST which effectively performs $1.25 \times 10^{11}$ base or $7 \times 10^9$ acid comparison per second on a 1.8GHz Opteron processor (using comparison rate = database size $\times$ total query length $\div$ elapsed time). Hence, for nucleotide searching DASH-H is 30 times faster than NCBI-BLAST, increasing to 100 times for protein searching.

* South Australian Partnership for Advanced Computing
Figure 7: Protein Speed and Sensitivity Comparison (Sudden Death Complete Coverage) of DASH and Other Algorithms Versus Smith-Waterman, with 95% confidence intervals shown by the cross hairs parallel to the relevant axis. Lower-right is better and upper-left is worse. This sensitivity measure counts the number of consecutive alignments returned by the Smith-Waterman search beginning from the highest scoring. Here, as for nucleotide alignment DASH dominates.

At these speeds, DASH-H is by far the fastest of the algorithms compared here. This computed acceleration is used to infer the expected run time distribution for DASH-H in the results tables and figures.

In terms of sensitivity, the exhaustive seed discovery process delivers dividends in the form of sensitivity which matches BLAST with query filtering disabled for nucleotide and probably betters blastpgp with Smith-Waterman optimal extension enabled. When the predicted run time of DASH-H is compared with those algorithms, the acceleration is greater than 100 times for both protein and nucleotide.

5 Conclusions

We have demonstrated the competency of the DASH algorithm in terms of both speed and sensitivity compared with a number of popular algorithms which address the same problem space. The DASH algorithm in software has been shown to provide a significant sensitivity advantage over other post-BLAST fast heuristics, and often achieving equivalent or better sensitivity than NCBI-BLAST. This sensitivity gain over the existing algorithms appears to be in part due to the effectiveness of DASH’s dynamic programming engine. Finally, the hardware variant of DASH demonstrates great potential by offering equivalent or superior sensitivity to NCBI-BLAST combined with a 100 fold speed advantage.

Acknowledgement We would like to thank the CSSIP (Co-Operative Research Center for Sensor, Signal and Information Processing) for supporting this research.

References


