Assessment of graphics processing units for acceleration of sequence database search programs

Alex Stivala

February 20, 2009

Student(s):	Alex Stivala
Association:	Victorian Partnership for Advanced Computing
Supervisor(s):	Mike Kuiper
Discipline:	Life Sciences

1 Research Objective

We aim to determine whether general-purpose graphics programming units (*GPGPU*s) are practical as a low-cost technique for acceleration of optimal sequence alignment algorithms in order for higher precision searches in comparable time to fast heuristic methods.

2 Motivation

Searching sequence databases for sequences similar to a query sequence is an important and frequently occurring task in bioinformatics. For example, such database searches allow some deductions about the function of a newly sequenced protein to be made on the basis of its sequence similarity to existing well-studied proteins, even in the absence of a known structure.

The Smith-Waterman algorithm (Smith and Waterman, 1981) provides a method to do this by mathematically optimal local alignment of the sequences. However, traditionally it has been too slow for practical large scale use, and heuristic methods such as BLAST (Altschul et al., 1990) and FASTA (Pearson and Lipman, 1988) are usually used instead.

The recent emergence of graphics processing units (*GPU*s) as relatively low-cost and high-performance processing parallel processing units, and, relative to CPUs, their currently faster rate of improvement in peak processing speed and memory bandwidth (NVIDIA, 2008), has lead to their use for purposes other than graphics rendering. Programming models that allow programming of GPUs in a general-purpose rather than in a graphics-specific manner, such as NVIDIA *CUDA* (Compute Unified Device Architecture) (NVIDIA, 2008), are likely to make it more practical to write code to make use of the speed and parallelism of GPUs.

3 Science Background

The Smith-Waterman algorithm uses dynamic programming to find the optimal local alignment of two sequences using a scoring matrix for nucleotides or amino acids and a linear gap cost, with quadratic time complexity (Smith and Waterman, 1981). The algorithm was enhanced by Gotoh (1982) to allow affine gap costs, that is, there is both a gap opening penalty and a separate gap extension penalty. The time complexity remains quadratic. References to the "Smith-Waterman algorithm" are often taken to include the affine gap penalty, and we do so here.

The "standard" implementation of the Smith-Waterman algorithm is the `SSEARCH` program in the FASTA package (Pearson and Lipman, 1988).

Sequence alignment algorithms can operate on arbitrary sequences, and in the context of bioinformatics we are generally operating on either DNA or RNA sequences (of nucleotides) or protein sequences (of amino acids). In this study we use only protein sequences.

Local sequence alignment can be used for sequence database search by finding the alignment score of a query sequence against each sequence in the database; the top-scoring "hits" are then ranked in order, giving a ranked list of sequences similar to the query sequence.

Heuristic methods such as FASTA (Pearson and Lipman, 1988), BLAST (Altschul et al., 1990) and PSI-BLAST (Altschul et al., 1997) use a variety of techniques, such as finding small exact matching seed alignments and extending them, and only reporting hits above a threshold, to achieve significant speed improvements over the Smith-Waterman algorithm, at some cost (at least in theory) to sensitivity. In addition, these methods, rather than reporting the raw alignment score, compute an E-value, giving the number of hits with the same (or better) similarity score that would be expected

to occur purely by chance, given the sizes of the input sequence and the searched database.

Another, more sensitive but slower, method for searching sequence databases is to use *profile Hidden Markov Models* (profile HMMs) (Durbin et al., 1998; Eddy, 1998). This method uses a statistical model to find sequences in the database that are similar to the sequences in the multiple alignment that was used to build the profile HMM used as the query. The HMMER software package[1] implements this technique.

3.1 Accelerating the Smith-Waterman algorithm

One method for accelerating the Smith-Waterman algorithm, without using a graphics card or other hardware, is to use single instruction multiple data (*SIMD*) vector instructions available on some processors. Wozniak (1997) used the Sun Ultra SPARC Visual Instruction Set to compute four rows of the dynamic programming matrix in parallel, achieving a better than two-fold speedup. Rognes and Seeberg (2000) use Intel *MMX* (multimedia extension) and *SSE* (streaming SIMD extensions) instructions where the SIMD registers contain values parallel to the query sequence, and a pre-computed "query profile" to achieve a six-fold speeedup. Farrar (2007) improves upon the Rognes and Seeberg (2000) method by "striping" the query profile in order to move data dependencies out of the inner loop, along with some further optimizations, achieving a further two to eight fold speedup. The current version of the `SSEARCH` implementation of Smith-Waterman in the FASTA package[2] incorporates the Farrar (2007) SSE2 acceleration.

[1] `http://hmmer.janelia.org`

[2] Version 35.4.2, `http://faculty.virginia.edu/wrpearson/fasta/fasta3.tar.gz`, downloaded 5/01/09

The Smith-Waterman algorithm can be accelerated by using special purpose hardware or FPGAs (Oliver et al., 2005), or by taking advantage of the parallelism provided by commodity graphics cards such as those manufactured by NVIDIA. This parallelism can be very fine-grained; for example the NVIDIA GTX 280 card has 30 multiprocessors, each with eight Scalar Processor cores, and hundreds of threads may be concurrently active (NVIDIA, 2008). Importantly, this hardware now supports double-precision floating point, which has not previously been the case (NVIDIA, 2008).

One method of taking advantage of the parallelism of the GPU is to compute all elements on the same antidiagonal of the Smith-Waterman dynamic programming matrix in parallel, thereby accelerating each single comparison of a query to a database sequence. This method is used by Liu et al. (2007). Another method is to simply allocate each thread in the GPU to an alignment of the query sequence to a single database sequence, thereby computing many alignment scores simultaneously. This method is used by Manavski and Valle (2008). As a consequence of the CUDA programming model, it is important for efficiency that all threads in a *grid* of threads should terminate as nearly at the same time as possible, and for this reason the database sequences should be sorted by length so that adjacent threads operate on database sequences of similar length. The `swcuda` implementation of Manavski and Valle (2008) does this automatically.

According to Wirawan et al. (2008), the best performing accelerated Smith-Waterman implementation published to date is their CBESW program for the Cell Broadband Engine in the Sony PlayStation 3.

3.2 Accelerating profile HMM algorithms

The computationally intensive algorithms involved in database searches using profile HMMs have also been accelerated using several different techniques, including the use of SIMD instructions, multiprocessor systems, FPGAs and graphics hardware (Walters et al., 2007). Recently, GPU-HMMER, a CUDA acceleration of HMMER was released,[3] as well as an alpha release of the new HMMER version 3, which contains heuristics to accelerate HMMER (Eddy, 2008) and makes use of the SSE2 instruction set.[4]

4 Methods

4.1 Software

We assessed software described in any of the papers previously cited if it was freely available (at least for academic use) and would run on our hardware (PC and NVIDIA graphics card). A list of the software assessed is shown in Table 1. For each program, the latest version currently available was used, and we compiled the software from source on our system. We tried both the GNU Compiler Collection (gcc) version 4.1.2 and the Intel C compiler version 11.0. In most cases there was little or no difference in speed of the compiled code based on the compiler used, so the gcc compiled version was used (on the basis that this compiler is available to everyone). An exception is HMMER 3.0a1 (Eddy, 2008), where the Intel compiler produced code that ran approximately twice as fast, so we used the Intel compiled version.

For CUDA programs, we built with the NVIDIA CUDA Toolkit and Software Development Kit Version 2.0.

In the case of `swsse2`, the striped Smith-Waterman implementation by Farrar (2007), the code was written for the Windows platform. In order to build it on Linux, we had to make minor changes (only to the timing code) and compiled it with gcc with the optimization `-O3` option.

For `swcuda`, the CUDA implementation of Smith-Waterman (Manavski and Valle, 2008), we made a minor change to allow the program to operate on FASTA files containing lowercase amino acid codes.

We found that PSI-BLAST crashed and gave warning messages and bad output as described by Osowski et al. (2007), and so we applied the patches[5] to to fix these bugs and optimize PSI-BLAST described therein.

We wrote scripts to implement the evaluations described in Section 4.3 in Python[6] using BioPython[7] including the Bio.SCOP interface (Casbon et al., 2006) to read SCOP and ASTRAL data. We modified the Bio.SCOP library slightly in order to not remove sequences whose identifiers do not start with 'd', since the genetic domain sequences consisting of multiple concatenated chains have identifiers starting with 'g' in the ASTRAL sequence database.[8] All graphs were created using the R statistics package.[9]

4.2 Data sets

For evaluating the accuracy of sequence database search using structural similarity as the gold standard, we used the ASTRAL (Brenner et al., 2000; Chandonia et al., 2004) compendium and the SCOP database (Murzin

[3] `http://www.mpihmmer.org`

[4] `http://cryptogenomicon.org`, January 12th, 2009

[5] `http://www.it.abo.fi/finhpc/blastpgp/`

[6] `http://www.python.org`

[7] `http://www.biopython.org`

[8] `http://astral.berkeley.edu/scopseq-os-1.73.html`

[9] `http://www.r-project.org`

Program	Version	Publication	GPU	Download location
NCBI BLAST	2.2.19	Altschul et al. (1990)	N	`ftp://ftp.ncbi.nih.gov/blast`
NCBI PSI-BLAST	2.2.15	Altschul et al. (1997)	N	`ftp://ftp.ncbi.nih.gov/blast`
FASTA / SSEARCH	35.4.2	Pearson and Lipman (1988)	N	`http://faculty.virginia.edu/wrpearson`
swcuda	1.92	Manavski and Valle (2008)	Y	`http://www.manavski.com`
swsse2		Farrar (2007)	N	`http://farrar.michael.googlepages.com`
HMMER	2.3.2	Eddy (1998)	N	`http://selab.janelia.org`
GPU-HMMER	0.9	Balu et al. (2008)	Y	`http://www.mpihmmer.org`
HMMER	3.0a1	Eddy (2008)	N	`http://selab.janelia.org`

Table 1: Software assessed. The GPU column indicates whether or not the software makes use of the GPU.

et al., 1995; Andreeva et al., 2004), version 1.73. We made use of both the full ASTRAL genetic domain sequence database based on PDB SEQRES records (92 913 sequences) and its 40% sequence identity non-redundant subset (9 536 sequences).

For accuracy evaluation, we used two query sets. The first is a set of 1551 query sequences generated by by selecting the sequence of one domain from each superfamily in the SCOP 1.73 that contains at least two domains. This gives a representative query set, in that each superfamily is represented by a single query. The second query set is all-against-all in the 40% sequence identity nonredundant subset, giving 9 536 queries.

Since ASTRAL contains only sequences that have a structure from the SCOP database, for timing experiments where known structures or classifications are not required for accuracy evaluation, we used a larger sequence database. We used UniProtKB/Swiss-Prot (Boeckmann et al., 2003) release 14.5, consisting of 402 482 sequences. In order to avoid problems with programs such as `swcuda` that do not handle characters other than the standard 20 amino acids symbols in FASTA files, we replaced all occurrences of U (selenocysteine) and O (pyrrolysine) with X (any).

When evaluating the GPU-HMMER `cuda_hmmsearch` program, we used a copy of the ASTRAL SCOP sequence database that was sorted by sequence length with the `hmmsort` program included in GPU-HMMER. This way, adjacent threads operate on database sequences of similar length, so that threads in a grid terminate as near to simultaneously as possible, in order to minimize thread latency and hence maximize speedup (Balu et al., 2008).

For timing in the UniProt database, we also used two query sets. The first is the set of 11 queries first defined by Altschul et al. (1997) and subsequently used by others including Rognes and Seeberg (2000); Farrar (2007); Manavski and Valle (2008). Sequence accession P10318 was replaced with P03989 (1B27_HUMAN) as the latter replaces the former in UniProtKB/Swiss-Prot release 14.5. The second query set is a set of 10 query sequences less than 360 amino acids in length, generated so that they are approximately evenly spaced in length from 10 to 360 amino acids. These are used for testing `swcuda` using the GPU only, and no CPU, since it automatically load balances between GPU(s) and CPU(s) available to it, and can only use solely the GPU for queries less than 360 amino acids long.

In finding models for the query sequences as described in Section 4.4, we used the Pfam database (Finn et al., 2008) release 23.0 (July

2008, 10340 families), in binary format. We calibrated the database with `hmmcalibrate` and indexed it with `hmmindex`.

4.3 Accuracy evaluation

We evaluate the specificity and sensitivity of sequence database search methods as a measure of homology by using membership of two sequences in the same SCOP family or superfamily as the gold standard for homology. This method was used by Brenner et al. (1998) and others subsequently including Rognes (2001) and Hulsen et al. (2006). Brenner et al. (1998) and Rognes (2001) used membership of the same SCOP superfamily as the criterion for true homology, while Hulsen et al. (2006) uses SCOP family. We evaluate using both criteria separately. This allows us to compare differences in sensitivity to more or less distant homology between methods, since membership of the same superfamily as the homology criterion assesses the ability of a method to detect more distant homology than using family.

Using this definition of true homology, we evaluate the accuracy of sequence database search methods using both ROC_{50} scores (Gribskov and Robinson, 1996) and coverage versus error plots (Brenner et al., 1998).

4.3.1 ROC_{50} calculation

Receiver Operating Characteristic (ROC) curves are used to measure the sensitivity and specificity of a classifier (in this case we are using sequence comparison methods to classify the query sequence as to whether or not it is in the same family or superfamily as a database sequence). The ROC curve is a plot of the true positive rate of a classifier against the false positive rate, as the cutoff score for the positive classification is varied. The area under the ROC curve (AUC) is then a measure of the performance of the classifier: a "perfect" classifier has an AUC of 1.0, and a random classifier has as its ROC curve the line with slope 1.0, which has AUC 0.5. The AUC is equivalent to the Mann-Whitney U statistic (Hanley and McNeil, 1982).

A shortcoming of the AUC measure is that the method must generate a score for every sequence in the database, which is an extremely large volume of results, and is not realistic in the case of sequence database searches, where typically only the top few hundred "hits" are used (and where many methods only generate a few hundred top scores). Gribskov and Robinson (1996) overcome these problems by defining ROC_{50} as the area under the ROC curve plotted until 50 true negatives are found. ROC_{50} or ROC_n for other values of n was subsequently also used in several other studies including Schäffer et al. (2001); Altschul et al. (2005); Hulsen et al. (2006). We compute the ROC_{50} score by sorting the results by score and calculating the Mann-Whitney U statistic (Mann and Whitney, 1947) for the top scores up to the first 50 false positives, and dividing this value by the product of the number of false positives (i.e. 50) and the total number of true positives possible in the data (i.e. the number of sequences that are in the same family or superfamily as the query). The average ROC_{50} score over all queries then gives the ROC_{50} score for the method and data and query set being assessed.

4.3.2 Coverage versus error calculation

Coverage versus error plots were introduced by Brenner et al. (1998) as an alternative to ROC curves for comparing sequence comparison methods which have a huge background of false positives (non-homologs), and were subsequently also used by Rognes (2001) and Hulsen et al. (2006). To generate the plot we use each sequence in the query set to search the database, and sort the combined results by

score. The cutoff score for considering a score as a hit is then varied from highest to lowest score (smallest to largest E-value).

Coverage is defined as the number of true positives divided by the total number of possible true positives (i.e. the number of sequences in the same family or superfamily as the query). Errors per Query (*EPQ*) is defined as the number of false positives divided by the number of queries. The plot of EPQ against coverage for each score threshold is the coverage versus error graph.

4.4 HMMER

HMMER database searches using `hmmsearch` are different from the other programs, in that the query is a profile HMM rather than a sequence. For this reason we cannot assess `hmmsearch` queries in the same way as the others by using membership of the same SCOP family or superfamily as the gold standard for homology. We use a less direct approach, using `hmmpfam` to find the best profile HMM (model) for each sequence in the query set. These models are then used as the `hmmsearch` query, and membership of the database sequence in the same SCOP family or superfamily as the sequence that was used as the input to `hmmpfam` to generate the query model was used as the gold standard for homology. Note that not all sequences find a model, and there may also be cases where the most significant model found is not really the best model for the sequence. Therefore some caution should be used in interpreting these results as evaluations of `hmmsearch` accuracy, and we assess the different versions of `hmmsearch` only against each other, not against other methods which were evaluated in the direct manner.

HMMER version 3 (3.01a) introduced a new program `phmmer` which takes a single sequence as the query, and hence it can be assessed in the same way as BLAST and the others.

4.5 Hardware

All tests were run on a PC with an Intel Q8200 CPU (4 cores) running at 2.33GHz with 4 GB memory running CentOS 5.2 Linux in 64-bit mode. The graphics card used was an NVIDIA GTX 280 with 1 GB memory.

5 Results

All the sequence search programs were run using the BLOSUM62 scoring matrix (Henikoff and Henikoff, 1992) with gap opening penalty of 11 and gap extension penalty of 1. The `phmmer` program in HMMER 3.0a1, which works differently and uses gap open and extend probabilities rather than penalties; we used the defaults (open probability 0.02, extend probability 0.4). The BLOSUM62 scoring matrix (the default) was used. The `hmmsearch` program works differently from sequence search programs, and the concepts of substitution matrices and gap costs are not relevant.

PSI-BLAST was run with a maximum of 5 iterations (with the `-j 5` option).

The HMMER 3.0a1 `phmmer` program with the `--max` option to disable heuristic filters (labelled as phmmer-max in the tables and graphs) was too slow to complete the all-against-all queries in practical time, and so is not included in the results.

5.1 Accuracy

Table 2 and Table 3 show the ROC_{50} values for the 1551 query sequences, one for each superfamily in SCOP, evaluated at the family and superfamily levels, respectively. We consider the methods as divided into four classes, the profile HMM methods (phmmer, with and without filter heuristics enabled), the Smith-Waterman methods (swsse2, swcuda, ssearch), the heuristic methods (BLAST, FASTA), and PSI-BLAST. We consider PSI-BLAST in a

method	roc50
ncbi-psiblast	0.896
ssearch	0.893
phmmer-max	0.892
phmmer	0.891
swcuda	0.890
swsse2	0.889
fasta	0.878
ncbi-blast	0.878

Table 2: Average ROC_{50} values for different methods for the 1551 queries against the ASTRAL SCOP 1.73 all sequences data set, evaluated at the family level.

method	roc50
ncbi-psiblast	0.793
phmmer	0.790
phmmer-max	0.789
ssearch	0.779
swsse2	0.778
swcuda	0.777
ncbi-blast	0.765
fasta	0.763

Table 3: Average ROC_{50} values for different methods for the 1551 queries against the ASTRAL SCOP 1.73 all sequences data set, evaluated at the superfamily level.

class of its own as it is both a heuristic method, and an iterative method that builds its own position-specific scoring model (like HMMER, but in a more *ad hoc* manner).

At both the family and superfamily levels, PSI-BLAST performs best on the ROC_{50} evaluation measure, followed by HMMER, then the Smith-Waterman methods and finally the heuristic methods. The exception is SSEARCH, which is (slightly) better than HMMER at the family level, but at the superfamily level we note that the increased sensitivity

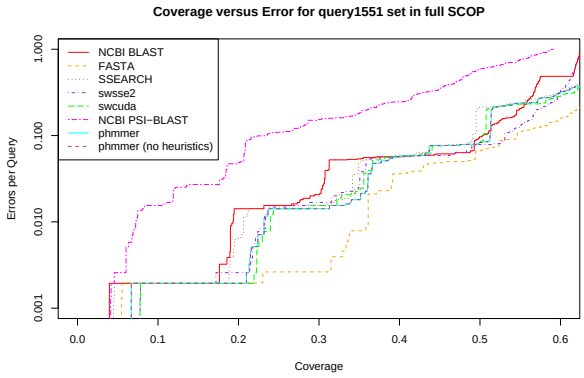

Figure 1: Coverage versus Errors Per Query plots for different methods for the 1551 queries against the ASTRAL SCOP 1.73 all sequences data set, evaluated at the family level.

of the statistical models used by HMMER and PSI-BLAST give them significantly better performance than the Smith-Waterman methods.

SSEARCH performs better than the other Smith-Waterman methods, due to its use of an E-value calculated by length regression statistics (Pearson, 1995; Rognes, 2001; Hulsen et al., 2006), rather than raw Smith-Waterman scores.

Figure 1 and Figure 2 show the coverage versus error graphs for the 1551 query set evaluated at the family and superfamily levels, respectively. Here we can see that at the family level, the Smith-Waterman methods and HMMER perform similarly on this measure, while BLAST is slightly worse. Interestingly, at the family level, PSI-BLAST has a noticeably higher level of errors per query, while FASTA is noticeably better. These distinctions remain at the superfamily level, but to a lesser degree.

Table 4 and Table 5 show the ROC_{50} values for the all-against-all queries in the ASTRAL SCOP 40% sequence nonredundant subset evaluated at the family and superfamily levels, respectively. Note that `phmmer-max`,

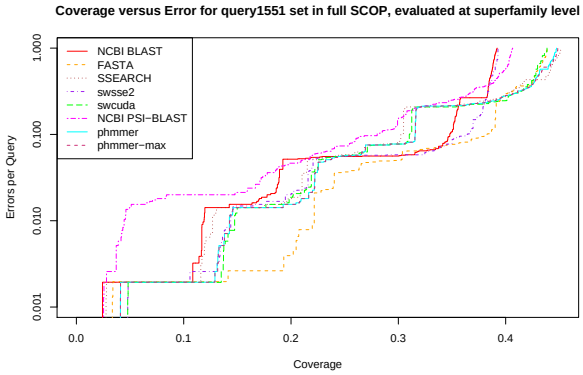

Figure 2: Coverage versus Errors Per Query plots for different methods for the 1551 queries against the ASTRAL SCOP 1.73 all sequences data set, evaluated at the superfamily level.

method	roc50
ncbi-psiblast	0.670
ssearch	0.604
phmmer	0.601
swcuda	0.600
swsse2	0.593
ncbi-blast	0.559
fasta	0.556

Table 4: Average ROC_{50} values for different methods on all-against-all queries in the ASTRAL SCOP 1.73 40% sequence non-redundant data set, evaluated at the family level.

method	roc50
ncbi-psiblast	0.412
ssearch	0.337
phmmer	0.335
swcuda	0.333
swsse2	0.328
fasta	0.302
ncbi-blast	0.299

Table 5: Average ROC_{50} values for different methods on all-against-all queries in the ASTRAL SCOP 1.73 40% sequence non-redundant data set, evaluated at the superfamily level.

the HMMER `phmmer` program with heuristic filters disabled, was not evaluated in this test as it was too slow to complete in practical time. The results are consistent with the previous set of queries, with only some minor variations in the rankings within the classes we have defined, such as `swcuda` and `swsse2` exchanging positions at the superfamily level, and BLAST and FASTA swapping position in the last two places. An exception is that SSEARCH has a better ROC_{50} score than HMMER at in the all-against-all queries at the superfamily level (and has similar ROC_{50} score to the Smith-Waterman methods, rather than a significantly higher one as it does at the family level), which is somewhat unexpected, as we expect HMMER to be more sensitive to distant homology than Smith-Waterman methods (and hence have a better score at the superfamily than family level). We might generally expect an increased sensitivity to come at the cost of decreased specificity, however, in this case this it not apparent from the coverage versus error graph.

As we would expect, and consistent with the results reported by Hulsen et al. (2006), the increasing redundancy (i.e. the inclusion of a large number of similar proteins) in the

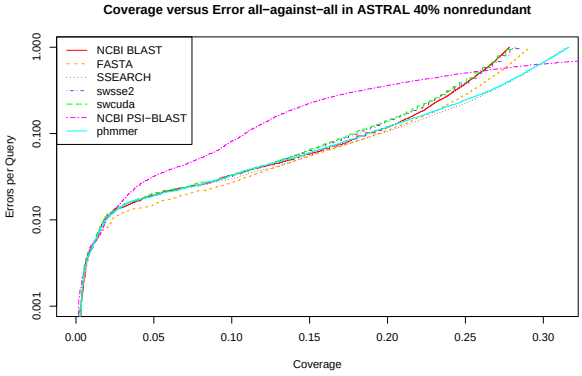

Figure 3: Coverage versus Errors Per Query plots for different methods on all-against-all queries in the ASTRAL SCOP 1.73 40% sequence non-redundant data set, evaluated at the family level.

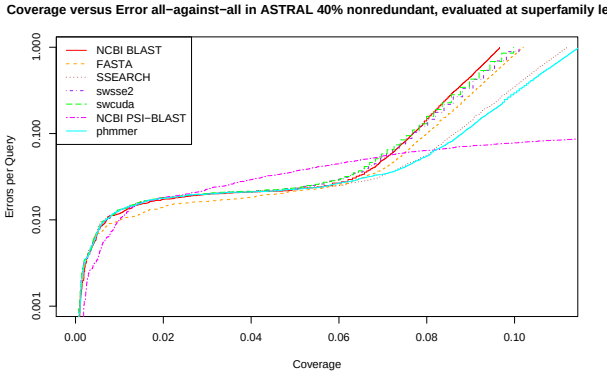

Figure 4: Coverage versus Errors Per Query plots for different methods on all-against-all queries in the ASTRAL SCOP 1.73 40% sequence non-redundant data set, evaluated at the superfamily level.

database leads to higher average ROC_{50} scores. Hence the values in Tables 2 and 3, where the search is in the entire ASTRAL SCOP sequence database, are higher than those in Tables 4 and 5, where the search is in the 40% sequence nonredundant ASTRAL SCOP database.

Similarly, ROC_{50} scores are lower at the superfamily than at the family level as sequence-based methods are better at finding more closely related proteins; at the extreme, if we were to evaluate at the fold level, scores would be even lower (data not shown), but structure-based methods would perform better (note, of course, that the SCOP classifications we are using as gold standard are based on protein structures).

Figure 3 and Figure 4 show the coverage versus error graph for the all-against-all queries in the 40% sequence nonredundant ASTRAL SCOP subset evaluated at the family and superfamily levels, respectively. Here the results are quite different from the 1551 query (one per superfamily) set. Consistent with the results of Hulsen et al. (2006), all the curves appear quite similar, with the exception of PSI-BLAST (not included by Hulsen et al. (2006)). At both the family and superfamily levels, SSEARCH and HMMER have fewer errors per query at the higher coverage end of the curve (which may be expected to be statistically more reliable (Hulsen et al., 2006)). This is more pronounced at the superfamily level, where HMMER shows better results. However, particularly striking is the curve for PSI-BLAST, which at the family level is inferior up to coverage of approximately 0.25, but at higher values has a clearly lower rate of growth of errors per query. At the superfamily level this is much more pronounced, and shows a quite different behaviour for PSI-BLAST than the other methods, reinforcing its superior sensitivity and specificity, also reflected in a significantly higher ROC_{50} value.

5.1.1 HMMER

Table 6 and Table 7 show the ROC_{50} scores for different implementations of `hmmsearch` evaluated at the family and superfamily levels, respectively. Figure 5 and Figure 6 show the cor-

method	roc50
cuda_hmmsearch	0.909
hmmsearch	0.909
hmmsearch3	0.885
hmmsearch3-max	0.885

Table 6: Average ROC_{50} values for different `hmmsearch` implementations for the `hmmsearch` HMM query set derived from the 1551 queries, evaluated at the family level.

method	roc50
cuda_hmmsearch	0.936
hmmsearch	0.936
hmmsearch3	0.815
hmmsearch3-max	0.813

Table 7: Average ROC_{50} values for different `hmmsearch` implementations for the `hmmsearch` HMM query set derived from the 1551 queries, evaluated at the superfamily level.

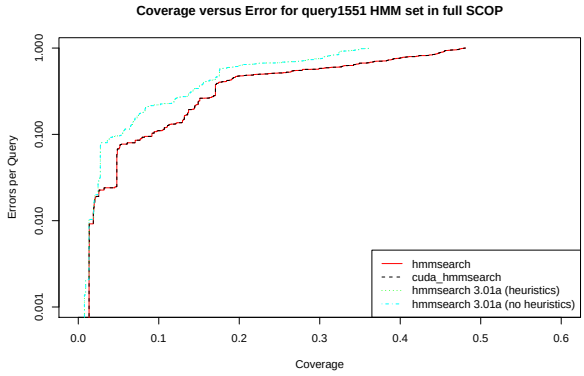

Figure 5: Coverage versus Errors Per Query plots for different versions of `hmmsearch` for the `hmmsearch` HMM query set derived from the 1551 queries, evaluated at the family level.

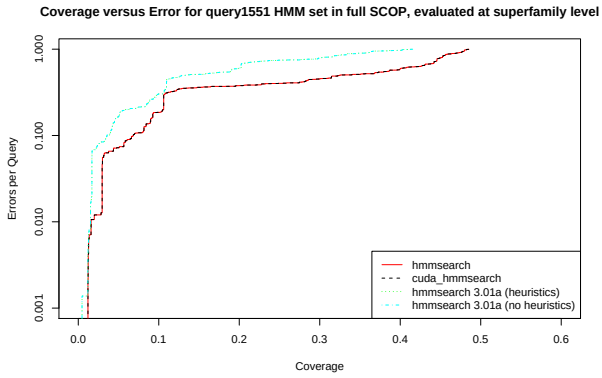

Figure 6: Coverage versus Errors Per Query plots for different versions of `hmmsearch` for the `hmmsearch` HMM query set derived from the 1551 queries, evaluated at the superfamily level.

responding coverage versus error plots. This shows that `cuda_hmmsearch` does indeed give exactly the same results as `hmmsearch`. We can also observe that `hmmsearch` in HMMER 3.01a gives somewhat less accurate results on these measures than version 2.3.2, with the heuristic filters making little or no difference to accuracy.

5.2 Speed

Figure 7 shows the elapsed times for the 11 query set in the UniProt database, and Table 8 shows the speedup relative to `swcuda` running on a single CPU core with no GPU and no SSE2 instructions. Note that the times are linear in the length of the query sequence, except for PSI-BLAST, where the number of iterations of the query is variable (up to 5).

It is apparent that, not unexpectedly, BLAST has the best speedup for a single CPU

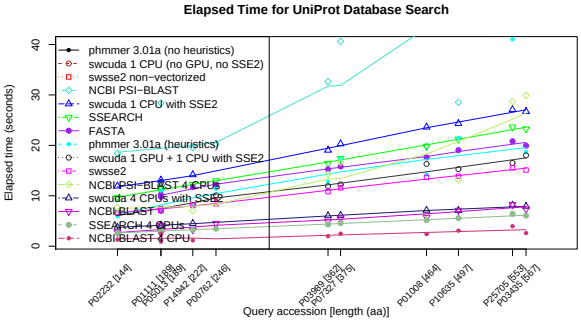

Figure 7: Elapsed times for the 11 queries in the UniProt database. Lines fitted by locally-weighted polynomial regression using the `lowess` function in R.

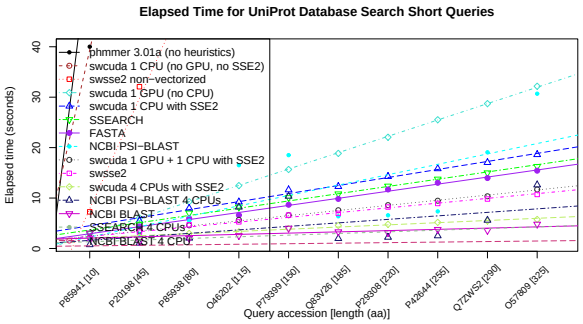

Figure 8: Elapsed times for the short queries in the UniProt database. Straight lines fitted by the `abline` function in R.

method	speedup
phmmer 3.01a (no heuristics)	0.58
swcuda 1 CPU (no GPU, no SSE2)	1
swsse2 non-vectorized	2.1
NCBI PSI-BLAST	16
swcuda 1 CPU with SSE2	23
SSEARCH	26
FASTA	30
phmmer 3.01a (heuristics)	32
swcuda 1 GPU + 1 CPU with SSE2	37
swsse2	40
NCBI PSI-BLAST 4 CPUs	41
swcuda 4 CPUs with SSE2	74
NCBI BLAST	87
SSEARCH 4 CPUs	100
NCBI BLAST 4 CPU	210

Table 8: Average speedup relative to `swcuda` on 1 CPU only with no SSE2 instructions for the 11 queries on the UniProt database.

core, and therefore the best overall speedup is to run BLAST on multiple cores. The best non-heuristic speedup is obtained by using the striped vectorization with SSE2 instructions (Farrar, 2007) on multiple cores: SSEARCH (which includes the Farrar (2007) striped vectorization with SSE2 instructions) on 4 cores is only slightly less than half as fast as BLAST on 4 cores, without using heuristics.

Figure 8 shows the elapsed times for the short query set in the UniProt database, and Table 9 shows the speedup relative to `swcuda` running on a single CPU core with no GPU and no SSE2 instructions. This enables us to test `swcuda` using the GPU only, and no CPU, since it automatically load balances between GPU(s) and CPU(s) available to it, and can only use solely the GPU for queries less than 360 amino acids long.

These results confirm that the use of striped vectorization and the SSE2 instructions gives better speedup than the use of the GPU in `swcuda`. The speedup using SSE2 instructions in `swcuda` is 20 times, but 14 times for using the GPU. The speedup for the SSE2 vectorization used in SSEARCH and `swsse2` is better still.

method	speedup
phmmer 3.01a (no heuristics)	0.56
swcuda 1 CPU (no GPU, no SSE2)	1
swsse2 non-vectorized	2.3
swcuda 1 GPU (no CPU)	14
swcuda 1 CPU with SSE2	20
SSEARCH	22
FASTA	25
NCBI PSI-BLAST	26
swcuda 1 GPU + 1 CPU with SSE2	33
swsse2	34
swcuda 4 CPUs with SSE2	57
NCBI PSI-BLAST 4 CPUs	75
NCBI BLAST	78
SSEARCH 4 CPUs	82
NCBI BLAST 4 CPU	242

Table 9: Average speedup relative to `swcuda` on 1 CPU only with no SSE2 instructions for the short queries on the UniProt database.

5.2.1 HMMER

Figure 9 shows the elapsed times for the `hmmsearch` queries derived from the 1551 query set. Table 10 shows the speedup of different `hmmsearch` implementations relative to `hmmsearch` from HMMER 2.3.2 on a single CPU core.

We can see that GPU-HMMER (`cuda_hmmsearch`) achieves an average speedup of 12 times, significantly better than using the 4 cores available on our test system. However, HMMER 3.01a with the heuristic filters enabled, which makes use of SSE2 instructions, is better still, achieving an average speedup of 71 times.

Figure 10 plots the speedup factor against the query HMM size. This allows us to see that the speedup for using multiple cores is (as we would expect) constant with no dependence on query size, as is the speedup for HMMER 3.01a with the heuristic filters disabled. When the HMMER 3.01a heuristic filters are enabled however, the speedup

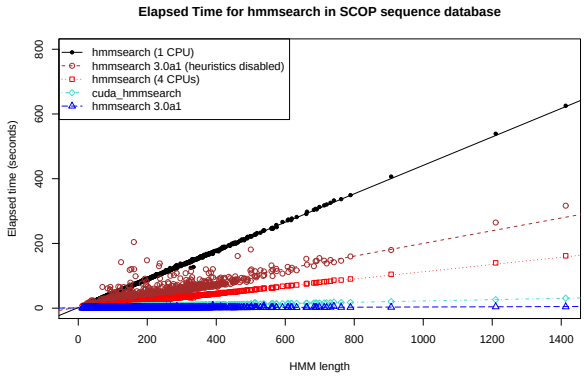

Figure 9: Elapsed times for the `hmmsearch` HMM query set derived from the 1551 queries in ASTRAL. Straight lines fitted by the `abline` function in R.

method	speedup
hmmsearch (1 CPU)	1
hmmsearch 3.01a (heurstics disabled)	2.1
hmmsearch (4 CPUs)	3.9
cuda_hmmsearch	12
hmmsearch 3.01a	71

Table 10: Average speedup relative to `hmmsearch` 2.3.2 on 1 CPU for the HMM query set derived from the 1551 queries in ASTRAL.

depends on particular properties of the individual queries, and so is not correlated with query size. GPU-HMMER (`cuda_hmmsearch`) achieves better speedup with increasing query size, with a peak of approximately 20 times speedup for query size 900 or greater.

6 Discussion

Our results indicate that the speedup of Smith-Waterman based sequence database search programs achieved by using GPUs (specifically the CUDA programming model) with the current implementations available to us, is not as great as that achieved by striped vectorization

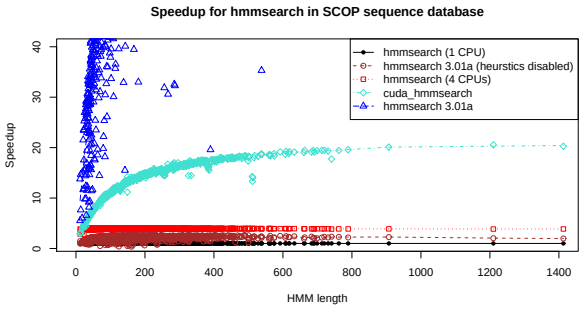

Figure 10: Speedup relative to `hmmsearch` version 2.3.2 for the `hmmsearch` HMM query set derived from the 1551 queries in ASTRAL. Lines fitted by locally-weighted polynomial regression using the `lowess` function in R, with $f = 0.01$.

with the SSE2 instructions available on standard Intel CPUs. It may well be possible that future implementations on GPUs will be faster, as the vectorization using SSE2 instructions is a more mature technique, which has even been incorporated into standard tools such as SSEARCH.

The limited memory available on the GPU, which necessitates balancing queries between the GPU(s) and CPU(s) available, and the overhead of moving data between the host and the GPU, means that GPU based acceleration is more difficult and has higher overheads than using SSE2 instructions which are already available on standard CPUs.

Furthermore, we have found that the PSI-BLAST program is more accurate on our evaluation than the Smith-Waterman implementations, and even the much more computationally intensive HMMER. Although not as fast as `swcuda`, SSEARCH or BLAST, its increased accuracy may be worth the reduction in speed, particularly if multiple processors are available to parallelize database searching.

The GPU-HMMER implementation of `hmmsearch` is a more promising use of the GPU: the `cuda_hmmsearch` program is on average 12 times faster, and for sufficiently long queries up to 20 times faster, than the standard `hmmsearch` program, with identical results. HMMER 3.01a, however, is on average 71 times faster, although with rather lower accuracy. Again, the use of SSE2 instructions (although in this case combined with heuristics) achieves superior results to the use of the GPU. However, as HMMER 3.01a is an alpha version, the HMMER 3.01a results should be treated with caution, as the release version may well behave differently.

The use of multiple GPUs in the same machine can achieve better speedups, as reported by Manavski and Valle (2008). It has also been reported that the Cell Broadband Engine can better accelerate the Smith-Waterman algorithm (Wirawan et al., 2008). However, our results would seem to indicate that efforts may be better concentrated in accelerating PSI-BLAST, since it can obtain superior accuracy to the Smith-Waterman algorithm, and even HMMER.

Another consideration is that sequence database searching, although in very widespread use on a large scale, is "sufficiently fast" already, and perhaps efforts in acceleration would be better concentrated on more computationally complex problems such as RNA structural alignment and database search and protein structural database search.[10]

7 Conclusion

The use of the GPU on its own can achieve up to a 14 times speedup over a single CPU core on the Smith-Waterman algorithm, and more

[10]The author discloses here that his PhD research is in algorithms for analysis of RNA and protein structure.

than 20 times in conjunction with the use of a CPU core with SSE2 instructions. However, this speedup is not as great as that achieved by using striped vectorization with SSE2 instructions on CPU cores only. It is also not nearly as fast as BLAST, although BLAST has some cost in accuracy. PSI-BLAST is also significantly more accurate on our evaluation than the Smith-Waterman algorithm, although slower. The GPU-HMMER implementation of the HMMER `hmmsearch` program achieves an average speedup of 12 times, and a peak of 20 times, although, again, the recently released alpha version of HMMER 3.01a achieves better speedup by using SSE2 instruction in conjunction with heuristics (at some cost to accuracy).

References

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

Stephen F. Altschul, John C. Wootton, E. Michael Gertz, Richa Agarwala, Aleksandr Morgulis, Alejandro A. Schäffer, and Yi-Kuo Yu. Protein database searches using compositionally adjusted substitution matrices. *FEBS Journal*, 272:5101–5109, 2005.

Antonina Andreeva, Dave Howorth, Steven E. Brenner, Tim J. P. Hubbard, Cyrus Chothia, and Alexey G. Murzin. SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Research*, 32(Database issue):D226–D229, 2004.

Vidyananth Balu, John Paul Walters, Suryaprakash Kompalli, and Vipin Chaudhary. Evaluating the use of GPUs for life science applications. Technical Report 2008-10, The State University of New York, Bulffalo, NY 14260, May 2008. `http://www.cse.buffalo.edu/tech-reports/2008-10.pdf`.

Brigitte Boeckmann, Amos Bairoch, Rolf Apweiler, Marie-Claude Blatter, Anne Estreicher, Elisabeth Gasteiger, Maria J. Martin, Karine Michoud, Claire O'Donovan, Isabelle Phan, Sandrine Pilbout, and Michel Schneider. The SWISS-PROT protein knowedgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, 2003.

Steven E. Brenner, Cyrus Chothia, and Tim J. P. Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proc. Natl. Acad. Sci. USA*, 95:6073–6078, 1998.

Steven E. Brenner, Patrice Koehl, and Michael Levitt. The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Research*, 28(1):254–256, 2000.

James A. Casbon, Gavin E. Crooks, and Mansoor A. S. Saqi. A high level interface to SCOP and ASTRAL implemented in python. *BMC Bioinformatics*, 7:10, 2006.

John-Marc Chandonia, Gary Hon, Nigel S. Walker, Loredana Lo Conte, Patrice Koehl, Michael Levitt, and Steven E. Brenner. The ASTRAL compendium in 2004. *Nucleic Acids Research*, 32(Database issue):D189–D192, 2004.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998. 11th printing (2006).

Sean R. Eddy. A probabilistic model of local sequence alignment that simplifies statistical significance estimation. *PLoS Computational Biology*, 4(5):e1000069, 2008.

Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 1998.

Michael Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23 (2):156–161, 2007.

Robert D. Finn, John Tate, Jaina Mistry, Penny C. Coggill, Stephen John Sammut, Hans-Rudolf Hotz, Goran Ceric, Kristoffer Forslund, Sean R. Eddy, Erik L. L. Sonnhammer, and Alex Bateman. The Pfam protein families database. *Nucleic Acids Research*, 36(Database issue):D281–D288, 2008.

O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162: 705–708, 1982.

Michael Gribskov and Nina L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers Chem.*, 20(1):25–33, 1996.

James A. Hanley and Barbara J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

Steven Henikoff and Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89: 10915–10919, November 1992.

Tim Hulsen, Jacob de Vlieg, Jack A. M. Leunissen, and Peter M. A. Groenen. Testing statistical significance scores of sequence comparison methods with structure similarity. *BMC Bioinformatics*, 7:444, 2006.

Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. Streaming algorithms for biological sequence alignment on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 18(9):1270–1281, 2007.

Svetlin A. Manavski and Giorgio Valle. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, 9(Suppl 2):S10, 2008.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

Alexey G. Murzin, Steven E. Brenner, Tim Hubbard, and Cyrus Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2008. version 2.0.

Timothy F. Oliver, Bertil Schmidt, and Douglas L. Maskell. Reconfigurable architectures for bio-sequence database scanning on FPGAs. *IEEE Transactions on Circuits and Systems II*, 52(12):851–855, 2005.

Kristoffer Osowski, Jan Westerholm, and Mats Aspnäs. Two cases of data overflow in the protein sequencing program BLASTPGP. Technical Report 813, Turku Centre for Computer Science, Åbo Akademi University, Finland, 2007. `http://www.tucs.fi/publications/attachment.php?fname=TR813.pdf`.

William R. Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4(6):1145–1160, 1995.

William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85(8):2444–2448, 1988.

Torbjørn Rognes. ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Research*, 29(7):1647–1652, 2001.

Torbjørn Rognes and Erling Seeberg. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

Alejandro A. Schäffer, L. Aravind, Thomas L. Madden, Sergei Shavirin, John L. Spouge, Yuri I. Wolf, Eugene V. Koonin, and Stephen F. Altschul. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Research*, 29(14): 2994–3005, 2001.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

John Paul Walters, Xiandong Meng, Vipin Chaudhary, Tim Oliver, Leow Yuan Yeow, Bertil Schmidt, Darran Nathan, and Joseph Landman. MPI-HMMER-Boost: Distributed FPGA acceleration. *The Journal of VLSI Signal Processing*, 48(3):223–238, 2007.

Adrianto Wirawan, Chee Keong Kwoh, Nim Tri Hieu, and Bertil Schmidt. CBESW: Sequence alignment on the PlayStation 3. *BMC Bioinformatics*, 9:377, 2008.

A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *CABIOS*, 13(2):145–150, 1997.

$Id: SeqAlignCaseStudy.tex 265 2009-02-16 01:16:24Z alexs $