High Performance Genome Scale Comparisons for the SAGE Method Utilizing Cray Bioinformatics Library (CBL) Primitives

Malali Gowda[1], Guo-liang Wang[1], Jeff Doak[2], Sankaraganesh Manikantan[3], Eric Stahlberg[3]

[1] The Ohio State University, Department of Plant Pathology, Columbus, Ohio, 43210

[2] Cray Inc., Mendota Heights, Minnesota, 55120

[3] The Ohio Supercomputer Center, Columbus, Ohio, 43212

Introduction

Large volume sequence comparison is of great importance in the current post-genomic and proteomic eras. Much of the importance stems from the need to assign function of a large number of genes or proteins identified from EST and genomic sequencing or proteomics projects. A necessary first step to assign a characteristic function of interested genes is the need to identify the expression pattern of the genes or proteins at a specific tissue or under a certain condition. The serial analysis of gene expression (SAGE) method accomplishes this task employing very large numbers of sequence comparisons to identify most of the expressed genes in the cells and to determine their expression levels quantitatively. The fact that very large numbers of sequences are involved in the comparison, that at least one of the comparative domains is short, and the need to identify both near and perfect matches make SAGE analysis an excellent method to benefit from a customized comparison application.

The SAGE Method

The serial analysis of gene expression (SAGE) method is a highly popular and comprehensive gene expression profiling method. Originally developed for studies of cancer and employing tags 14 nucleotides in length [Velculescu et al 1995], the SAGE method has found favorable application in the research of other organisms [Saha et al 2002; Chen et al 2002]. The SAGE method involves the identification and isolation of characteristic marker subsequences, generally concentrated in the 3' end of the region, which are associated with individual genes. The size of the SAGE sequence or tag is generally compact, involving between 10 and 30 nucleotides to uniquely characterize the specific marker. Acting as a fingerprint of expression activity, the SAGE tags provide qualitative insight into cellular processes as well as quantitative measurement based on the sampled volume of tags isolated experimentally. The SAGE method has been exploited in many instances as a method for genome annotation [Fizames et al 2004].

In this particular application, the SAGE method has been applied to research involving rice and the rice pathogen *Magnaporthe grisea* that causes the rice blast disease and negatively impacts the productivity of this important cereal crop. Two rice species, the indica species [Yu 2002] (adapted to tropical climates) and the japonica species [Goff 2002] (a variety adapted for temperate climates) were studied by looking at response of the species to attack by rice blast. Rice serves as an excellent prototypical cereal plant for study, with its relatively compact (450 million nucleotide) genome, genome synteny among cereals, and ease of genetic studies and transformations. The combined availability of the sequenced rice genome and the availability of the sequenced Magnaporthe fungus provide the necessary tools to investigate host-pathogen interactions. In the investigation that is the subject of this paper, the RL-SAGE method employing tags 21 nucleotides in size [Gowda et al 2004] was employed. This RL-SAGE method, not only provides in-depth expression pattern of all genes, but also provides early insight into novel genes.

Computational Demands for Rice SAGE Analysis

The SAGE method requires use of large numbers of identifying tags to provide the diagnostic and quantitative power. Rapid searching is imperative to keep job completion time to a minimum. Analysis of the cDNA libraries regularly generates libraries of tens of thousands of unique tags that must be compared against the available genome, cDNA and expressed sequence tag (EST) sequences available publicly. The publicly available information was obtained from multiple sources, TIGR, KOME and independent rice genome projects. In excess of 55,000 EST, chromosomal and cDNA rice Magnaporthe sequences and 600,000 SAGE tags were analyzed in the full research study, including validation in known models and analysis of experimentally derived tags for control, resistant and susceptible plants.

An important added complication in the analysis is the fact that DNA sequencing is not a perfected process, occasionally leading to mis-assignment of nucleotide identity for elements in either the laboratory isolated SAGE tags or the target genome, cDNA, and EST information. Consequently, the algorithm used for the SAGE analysis must be able to take into account these sequencing errors, providing information on candidate matches, even when a single or double nucleotide sequencing error would be present.

As mentioned previously, the number of comparisons required for an exhaustive analysis is significant. In the current study, in excess of 6.2 million virtual tags were isolated from sequence information for comparison to the 93,802 unique tags. Standard BLAST techniques, the de facto standard for inexact sequence matching, are not feasible for comparison efforts on this order.

Typically, the target sequences are much longer than the SAGE tag itself, which qualifies widely used search techniques (e.g. BLAST) for the comparison engine in this effort. However, the short size of the tags, combined with the need to explicitly allow for multiple nucleotide errors within each tag limits the true effectiveness of BLAST as a solution. A significant amount of unnecessary processing would be added to impose the required constraints, an intolerable situation with the volume of sequence comparisons required.

Computational Implementation

A new application suite was developed to optimize the SAGE specific search processing for this research project. The core comparison engine, SAGESPY was developed with significant input from the Wang research group regarding the specific application capabilities. This core application uses as input a file of SAGE tags, a file of target sequences which the SAGE tags will search, and a small control file to guide the comparison and selection of output. Both input sequence files utilize the relatively compact and widely accepted FASTA format for representing sequence information. The program creates output files that separate matched and unmatched tags and targets. Additionally, the program provides an XML file containing details of matched tags and targets, essentially, the intersection of the tag and target datasets.

The core application was developed with the CBL and PCBL libraries, primarily due to the availability of the cb_searchn() primitive which provided a high-speed near exact comparison method required for the analysis. The additional routine cb_compress() was employed to prepare the data for comparison with cb_searchn(). These libraries proved an efficient means to code and develop the application in early prototype stage.

The high-velocity comparison engine has been developed in Fortran and employs the OpenMP standard (http://www.openmp.org) to enable parallelism on shared memory computer architectures. The choice of Fortran for the implementation was driven by three factors, 1) the natural affinity of the CBL for a Fortran application; and 2) the need for efficient access to multi-dimensional arrays involved in the analysis; and 3) the generally more effective support for OpenMP parallelism in Fortran programming environments. Fortran 90 support for dynamic memory allocation was additionally critical for enabling throughput in a batched managed environment as exists at OSC.

The core application is supported by Java-based applications developed to perform necessary preprocessing efforts to extract virtual tags for comparison, identify non-redundant tag sets, and to conduct post-processing analysis of the data to characterize locality of the matching tags in the gene and to prepare for database importation.

Performance Results – Success, Bottlenecks and Breakthroughs

The OpenMP parallel implementation proved very effective in the Cray X1 and ultimately Cray SV1 SMP environment. Following the resolution of an issue in the Cray SV1 implementation relating to an overflow of thread stack size that eliminated all parallel performance improvement, both the Cray SV1 and Cray X1 systems demonstrated high throughput capacity. The application demonstrates extremely favorable scalability characteristics of 15.9 out of a theoretical maximum 16 in the latest version evaluated.

The high degree of parallel efficiency is only achievable with certain conditions. First, the application had not only been designed to read full datasets into memory before processing, but also to store all intermediate results during the search and comparison operation. The fastest I/O operation is that which is not done. By storing intermediate results, all I/O operations in the comparative phase are deferred, and a minimal contention design to access the shared area enables the results to be collected across threads as they are created. The impact on total memory requirements due to reading the entire datasets into memory were comparatively minimal. The large memory of the Cray systems is convenient storing intermediate results, although in the case of SAGE comparisons, the number of intermediate results grows only as the number of tags examined.

Second, efficient I/O to load the information into the application, and write information upon completion is mandatory. The earliest implementation of the SAGESPY program did not employ the cb_read_fasta() methods available in the PCBL and CBL libraries, but relied on Fortran-based I/O operations. This initial design choice was made to minimize the memory profile for the application and assure portability of the application across multiple systems. Ultimately, this approach proved to impose a significant performance penalty on the application when compared with the available CBL/PCBL method cb_read_fasta().

The choice to employ cb_read_fasta() proved very effective at reducing I/O overhead in the application. However, the choice did have an impact on the application portability as written in Fortran and in application memory requirements. The impact on portability stems from a mild inconsistency in the PCBL/CBL implementations for cb_read_fasta() when called from Fortran environments. As a result, portability was not prioritized to be maintained and execution became exclusive for Cray environments using the CBL libraries. Overall, the choice to use cb_read_fasta() proved an important factor in achieving the goal of high-velocity SAGE tag and sequence scanning.

A new effort originating in part from the portability concerns in PCBL/CBL APIs is the creation of the Ohio Bioscience Library (OBL). While the CBL (and the PCBL by consequence) are focused on mid to low level bioinformatics application primitives, the

OBL is a set of extensions to the CBL/PCBL libraries which, while building on the foundation provided by the CBL and PCBL methods, provide a higher-level set of APIs and applications. These APIs and applications are suited for conducting high-performance, high-throughput bioinformatics sequence analysis in multiple high-performance computing architectures and environments. The ultimate goal of the OBL is to provide a portable set of bioinformatics APIs, frameworks and applications which can be readily employed for doing significant sequence analysis and comparisons in today's and tomorrow's high-performance computing environments. The first release of the OBL is planned for summer 2004, which will include the SAGESPY application.

Despite the high-speed capabilities present in the cb_searchn() method, the large quantity of SAGE tags and target sequences to be compared still made for daunting run-time requirements. Consequently, after further discussion with the Wang group, the efforts were focused on improving the performance for the exact comparison and search, the most critical need for the research application. This change in relative importance of comparison resulted in a significant new approach to be employed.

Methods for doing exact string or sequence searching are well studied efforts, with many successful algorithms from which to choose. Individually, methods such as Rabin-Karp [Rabin and Karp 1981] and Knuth-Morris-Pratt [Knuth et al 1977] provide efficient means to complete a specific substring search. Even the cb_searchn() method works quite well for exact substring searches when the threshold for allowable mismatch is set to 0. Unfortunately, order analysis of the current implementation indicated that even with a very efficient exact substring search, the computational magnitude would remain O(nm) where n is the number of SAGE tags and m is the number of target sequences to be searched. A better approach was needed.

A faster approach for SAGE tag association had been employed by [Blake Meyers, University of Delaware, personal communication] for work on Arabidopsis thaliana employing MPSS tag isolation from each target sequence prior to comparison. In this approach, candidate match tags are isolated from the target sequences before comparison to the actual SAGE tags. Implemented within the context of a database system, this general approach proved reasonably effective for tag search and comparison for relatively modest quantities of SAGE tags and target sequences. The required comparison effort required in the SAGE analysis of rice, was much more significant due to the larger genome and availability of candidate SAGE tags.

Building on the method to isolate tags prior to comparison, a much more efficient exact comparison approach has been implemented SAGESPY application. The current fast path of execution through the application relies on minimizing the number of overall comparisons required to discern equality of two sequences. A highly efficient means to compare two similar lists is to first sort each list using common criteria, and then sequentially process the elements looking for similarities. Analogous to the process employed in the merge phase of a merge sort, this approach dramatically reduces the computational effort required to match the tags and targets. Assuming achieving O(n lg n) computational complexity measure for an efficient sort (e.g. heap sort, quick sort and merge sort) is reasonable, the order of computational complexity in the revised comparative approach becomes O(m lg m + n lg n). The enhanced tag extraction approach dramatically reduced the run time for the comparison in terms of computational order. In the case of a 64,000 set of target sequences and tags, the speedup can be as much as 4,000 fold. This new approach has been implemented in a generalized form in the SAGESPY application employing characteristic signatures for each of the tags or sequences involved in the comparative search.

While the new algorithmic approach provides a tremendous boost for comparing tags for exact matches, it does have limitations. It does not address the need to handle inexact matches when it is necessary to determine near matches when a single or double nucleotide mismatch exists, nor does a general signature based approach support the searching for sequence reverse complements (as would be generated with cb_revcomp()) in the absence of special pre-processing. The use of an integer logic-based primitive, Exclusive OR (XOR) holds the most promise for addressing these limitations while providing significant performance improvements. In contrast to generalized sequence signatures, the XOR operation enables an estimation of the degree of mismatch between two like sized entries while doing it very quickly.

In the case of SAGE tag analysis, conveniently the tags remain relatively small enabling a binary translation for nucleotide sequences into single word (64-bit) integer representations. The cb_compress() method performs this translation and in the case of nucleotide sequence, a SAGE tag as long as 32 nucleotides can be readily converted for representation in a 64-bit word unit. The equality of the sequences is readily determined by the XOR operation, with an exact match yielding no one bits in the result – an integer zero. The magnitude of mismatch is also determined by examining the popcount() of the answer. The popcount() operation (all non-zero bits in a word0 establishes a minimum bound to the number of mismatching elements between two compared sequences.

The use of the XOR is functionally equivalent to a replacement for the cb_searchn() method when the threshold is 0 and the compared nucleotide sequence lengths are less than 32. In the remaining cases of limited size word comparisons, the XOR method provides a low-cost accelerator, providing a quick pretest to determine whether the compared elements have a possibility of satisfying the conditions of acceptable mismatch. Preliminary timings are available in Tables 1 and 2 showing relative performance of the XOR operation compared to the cb_searchn() primitive.

Conveniently, the XOR implementation is not limited to accelerating near match comparisons, nor required to utilize the O(nm) algorithm for analysis. With the working constraints of exact comparison and no more than 32 nucleotides in length, the XOR logic functions as an equality test for the same comparative O(n lg n + m lg m) approach. While not appreciably faster than the use of sequence signatures to create sorted lists, the XOR comparison is exhaustive and eliminates the remote possibility of two different sequences possibly being evaluated as equal.

A second speed enhancement tested for the application reduces the number of individual calls to the cb_searchn() method. Instead of making a call to cb_searchn() for each individual tag and target comparisons, the target sequences are combined into a long single sequence that is then scanned in one call to the cb_searchn() method. Even with the additional overhead to discount inappropriate matches that happen in the concatenated target sequence, the target union approach also promises significant performance increases. Tables 1 – 3 provide a look at comparative performance of the

Cray SV1 and Cray X1 for the three exhaustive approaches, one target-tag comparison per cb_searchn() call, one cb_searchn() call per target (target fusion), and one target-tag comparison using an XOR based comparison. Table 4 provides performance information for the complete processing of the datasets utilizing the XOR method on the Cray SV1 and Cray X1.

Table 1. Relative performance of one target-tag per comparison approach. A set of 100 tags were utilized. All SAGE tags were 21 nucleotides long. A mismatch threshold of 2 nucleotides was used.

	Number of target tags	Individual SV1 time (seconds)	Individual X1 time (seconds)	Number of Tag Hits
Chrom1	634952	716.42	1372.61	29
Chrom2	566094	638.72	1235.54	30
Chrom3	602420	679.71	1301.10	35
Chrom4	472536	533.17	1026.72	34
Chrom5	410074	462.69	890.75	24
Chrom6	449616	507.30	973.52	21
Chrom7	417020	470.52	902.95	29
Chrom8	403882	455.70	871.40	12
Chrom9	331252	373.75	714.06	16
Chrom10	284020	320.46.	618.16	8
Chrom11	364548	411.32	792.54	32
Chrom12	360950	407.26	782.21	21
TIGR(EST)	470924	531.35	1015.72	333
KOME(cDNA)	475998	537.07	1026.53	262

Table 2. Single target fusion search analysis. A set of 100 tags were utilized. All SAGE tags were 21 nucleotides long. A mismatch threshold of 2 nucleotides was used.

	Number of target tags	Target Union SV1 time (seconds)	Target Union X1 time (seconds)	Number of Tag Hits
Chrom1	634952	33.57	27.80	29
Chrom2	566094	30.22	25.11	30
Chrom3	602420	32.93	27.40	35
Chrom4	472536	25.82	21.54	34
Chrom5	410074	21.98	18.11	24
Chrom6	449616	23.90	19.78	21
Chrom7	417020	22.24	18.39	29
Chrom8	403882	21.50	17.94	12
Chrom9	331252	17.71	15.18	16
Chrom10	284020	15.77	13.19	8
Chrom11	364548	20.20	16.89	32
Chrom12	360950	20.01	16.73	21
TIGR(EST)	470924	40.29	33.87	333
KOME(cDNA)	475998	32.17	27.11	262

Table 3. Short tag XOR search analysis. A set of 100 tags were utilized. All SAGE tags were 21 nucleotides long. A mismatch threshold of 2 nucleotides was used.

	Number of target tags	XOR SV1 time (seconds)	XOR X1 time (seconds)	Number of Tag Hits
Chrom1	634952	1.35	0.36	29
Chrom2	566094	1.20	0.32	30
Chrom3	602420	1.29	0.34	35
Chrom4	472536	1.00	0.27	34
Chrom5	410074	0.87	0.24	24
Chrom6	449616	0.96	0.26	21
Chrom7	417020	0.89	0.25	29
Chrom8	403882	0.86	0.23	12
Chrom9	331252	0.71	0.19	16
Chrom10	284020	0.61	0.16	8
Chrom11	364548	0.77	0.20	32
Chrom12	360950	0.77	0.21	21
TIGR(EST)	470924	1.00	0.26	333
KOME(cDNA)	475998	1.01	0.27	262

Table 4. Total CPU time (in seconds) for XOR-based evaluation of sequence similarity using an equality threshold of two nucleotide mismatches. 93802 unique SAGE tags were compared against chromosomal, EST and cDNA sequences.

	Number of Virtual Target Tags for Comparison	XOR SV1 time (seconds)	XOR X1 time (seconds)	Total Matching Tag Count
Chrom1	634952	1264.56	353.62	55286
Chrom2	566094	1127.77	299.71	48170
Chrom3	602420	1199.82	317.83	53976
Chrom4	472536	941.15	259.90	38985
Chrom5	410074	816.94	228.04	34742
Chrom6	449616	898.57	244.27	38520
Chrom7	417020	830.79	226.19	35660
Chrom8	403882	804.49	219.08	33581
Chrom9	331252	659.79	181.24	26791
Chrom10	284020	565.75	151.00	23509
Chrom11	364548	726.15	194.18	29217
Chrom12	360950	719.03	199.47	29744
TIGR (EST)	470924	938.23	258.20	100102
KOME (cDNA)	475998	948.36	259.75	99399

The performance timings shown above clearly identify a significant performance penalty is imposed using individual invocations of cb_searchn() for each tag and target comparison. A factor of 40x speed improvement can be obtained by converting the algorithm to employ a single target union when searching for tag matches. This alteration can be accomplished without loss of generality in the application.

A second and more dramatic improvement in the algorithm will be obtained when employing the XOR comparison instead of invoking the cb_searchn() method. In this case a factor of 500x speed improvement can be expected for the application. This second improvement imparts a loss of generality, imposing the restriction that elements to be compared be of equal length and less than 32 nucleotides in length. This method has been implemented in an API within the OBL called obl_short_searchn().

Both of the above efforts provide tremendous improvement in the most critical remaining bottleneck, identifying tags matching within one or two nucleotides. The dramatic improvements seen in both of the above methods serve to underscore the importance of continued optimization efforts. They also serve to highlight that even naïve brute-force parallel implementations can be improved, and optimization can provide performance gains exceeding those of a parallel implementation.

Future Directions

Several improvements remain in progress for SAGESPY prior to availability later in 2004. These include incorporating truly portable implementations using OBL API's derived from and relying on PCBL and CBL libraries or implementations, introducing the XOR comparative step to accelerate inexact tag comparisons, and final quality assurance validation. SAGESPY will be released in conjunction with the OBL software release. Contact the Ohio Supercomputer Center for advance versions of the OBL application library prior to general availability.

Conclusions

The CBL (and corresponding PCBL) provide a highly efficient means and capable foundation from which to develop bioinformatics searching and comparison algorithms. A significant improvement can be obtained using the cb_read_fasta() method for I/O when compared to normal Fortran I/O, although presently at some expense of application portability. The performance of the resulting applications, while capable, can still take significant time when faced with genome size comparative efforts. Algorithmic improvements extending the original capabilities of the CBL/PCBL can provide tremendous improvements in overall runtime. As a result, higher level APIs are valuable to incorporate the algorithmic improvements to common comparative operations. These APIs and applications are being developed and delivered through an Ohio Bioscience Library coordinated through OSC.

Acknowledgements

The authors wish to acknowledge Cray Inc. for providing advance access to Cray X1 systems for performance benchmarking. Doak and Stahlberg wish to acknowledge Bill Long of Cray and Jim Long of the Arctic Regional Supercomputer Center for discussions and insight provided when utilizing the CBL and PCBL respectively. The SAGE rice and rice blast project in Wang lab is supported by the National Science Foundation – Plant Genome Research Program (#115642).

References

Velculescu VE, Zhang L, Vogelstein B and Kinzler KW (1995). Serial Analysis of Gene Expression. Science 270: 484-487.

Saha S, Sparks AB, Rago C, Akmaev V, Wang CJ, Vogelstein B, Kinzler KW (2002) Using the transcriptome to annotate the genome. Nat Biotechnol 20: 508-512

Chen J, Sun M, Lee S, Zhou G, Rowley JD, Wang SM (2002) Identifying novel transcripts and novel genes in the human genome by using novel SAGE tags. Proc Natl Acad Sci USA 99: 12257-12262.

Fizames C, Munos S, Cazettes C, Nacry P, Boucherez J, Gaymard F, Piquemal D, Delorme V, Commes T, Doumas P, Cooke R, Marti J, Sentenac H, and Gojon A (2004) The Arabidopsis Root Transcriptome by Serial Analysis of Gene Expression. Gene Identification Using the Genome Sequence. Plant Physiology, 134(1): 67 – 80.

Yu J, Hu S, Wang J, Wong GK, Li S, Liu B, Deng Y, Dai L, Zhou Y, Zhang et al. (2002) A draft sequence of the rice genome (Oryza sativa L. ssp. indica). Science 296: 79-92

Goff SA, Ricke D, Lan TH, Presting G, Wang R, Dunn M, Glazebrook J, Sessions A, Oeller P, Varma H et al. (2002) A draft sequence of the rice genome (Oryza sativa L. ssp. japonica). Science 296: 92-100

Gowda M, Jantasuriyarat C, Dean RA, Wang GL (2004) Robust-LongSAGE (RL-SAGE): A Substantially Improved LongSAGE Method for Gene Discovery and Transcriptome Analysis. Plant Physiology 134(3):890-7

Karp RM and Rabin MO (1981), Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University

Knuth DE, Morris JH Jr., and Pratt VR (1977). Fast pattern matching in strings. SIAM Journal on Computing, 6(2):323-350