

Bioinformatics on Embedded Systems: A Case Study of Computational Biology Applications on VLIW Architecture

Yue Li and Tao Li

Intelligent Design of Efficient Architectures Laboratory (IDEAL)
Department of Electrical and Computer Engineering
University of Florida, Gainesville, Florida, 32611
<http://www.ideal.ece.ufl.edu/>
yli@ece.ufl.edu, taoli@ece.ufl.edu

Abstract. Bioinformatics applications represent the increasingly important workloads. Their characteristics and implications on the underlying hardware design, however, are largely unknown. Currently, biological data processing ubiquitously relies on the high-end systems equipped with expensive, general-purpose processors. The future generation of bioinformatics requires the more flexible and cost-effective computing platforms to meet its rapidly growing market. The programmable, application-specific embedded systems appear to be an attractive solution in terms of easy of programming, design cost, power, portability and time-to-market. The first step towards such systems is to characterize bioinformatics applications on the target architecture. Such studies can help in understanding the design issues and the trade-offs in specializing hardware and software systems to meet the needs of bioinformatics market. This paper evaluates several representative bioinformatics tools on the VLIW based embedded systems. We investigate the basic characteristics of the benchmarks, impact of function units, the efficiency of VLIW execution, cache behavior and the impact of compiler optimizations. The architectural implications observed from this study can be applied to the design optimizations. To the best of our knowledge, this is one of the first such studies that have ever been attempted.

1. Introduction

The study of genetics has remarkably advanced our knowledge of the fundamental of life: in 1865, G. Mendel first discovered the phenomena of genetic inheritance, whereas now, life sciences have matured to the extent of making cloning of living beings a reality. Today, to understand biological processes and, in turn, advances in the diagnosis, treatment, and prevention of genetic diseases, researchers rely on the advanced laboratory technologies (e.g., electrophoresis and mass spectrometry, micro array transcript analysis) [1] to study all the genes as well as their activity levels and complex interactions in an organism.

As genomic science moves forward, having accessible computational tools with which to extract and analyze genomic information is essential. The field of bioinformatics, defined as the computationally handling and processing of genetic information, has experienced an explosive growth in the last decade. Since the human genome [2] has been deciphered, it has become evident that bioinformatics will become increasingly important in the future. Today, bioinformatics has become an industry and has gained acceptance among number of markets especially in pharmaceutical, biotechnology, industrial biotechnology and agricultural biotechnology. A number of recent market research reports estimate the size of the bioinformatics market is projected to grow to \$243 billion by 2010 [3].

Clearly, computer systems which provide high-performance, cost-effective genetic data processing play a vital role in the future growth of the bioinformatics market. Many major IT

companies (e.g., IBM, Microsoft, SGI, and Apple) have announced products specific to bioinformatics applications [4, 5], while dozens of start-up companies devoted to bioinformatics have arisen [6, 7]. Most of these solutions continue to address the needs of bioinformatics by developing complex and expensive high-end systems equipped with general-purpose processors. Costly and time-consuming, these approaches can also result in hardware and software architectures that are not optimized for the price, power, size and flexibility requirements of the future bioinformatics computing.

As their popularities and market continue to grow, future bioinformatics and computational biology are likely to adopt the application-specific processors and systems to win the increased competition between manufacturers. It has been widely accepted that embedded systems have become powerful enough to meet the computational challenge from many application domains [8]. On the other hand, using programmable, application-specific processors can provide much more flexible solutions than an approach based on ASICs and is much more efficient than using general-purpose processors in terms of cost, power, portability and the time-to-market.

To achieve high-performance, genetic information processing needs to exploit instruction level parallelism (ILP). General-purpose processor architectures, such as aggressive, out-of-order execution superscalar, detect parallelisms at runtime using highly complex hardware. In contrast, VLIW architectures use the compilers to detect parallelisms and reduce hardware implementation cost. Consequently, the VLIW is increasingly popular as the architecture paradigms for the programmable, application-specific embedded processors [9].

The first step towards the cost-effective genetic data processing platforms is to characterize the representative bioinformatics applications on the target architecture. Such studies can help in understanding the design issues of the new generation of programmable, application-specific processors to meet the needs of bioinformatics market as well as the software/hardware tradeoffs that can be made to fine tune the systems. This paper evaluates several representative bioinformatics software on the VLIW based embedded systems. The workloads we used include the popular DNA/protein sequence analysis, molecular phylogeny inference and protein structure prediction tools. We investigate various architectural features, such as the basic workload characteristics, impact of function units, the efficiency of VLIW execution, cache behavior and the effectiveness of compiler optimizations. The architectural implications observed from this study can be applied to the design optimizations. To the best of our knowledge, this is one of the first such studies that have ever been attempted.

The rest of the paper is organized as follows. Section 2 provides brief reviews of biology background and bioinformatics study areas. Section 3 describes the workloads, the architectures modeled, and the simulation methodology. Section 4 presents the characterization of bioinformatics benchmarks and the architectural implications. Section 5 concludes the paper.

2. Bioinformatics Background

This section provides an introductory background for biology and describes the major areas of bioinformatics.

2.1. DNA, Gene and Proteins

All living organisms use DNA (deoxyribonucleic acid) as their genetic material. The DNA is essentially a double chain of simpler molecules called nucleotides, tied together in a helical structure famously known as the double helix. There are four different kinds of nucleotides: adenine (A), guanine (G), cytosine (C) and thymine (T). Adenine (A) always bonds to thymine (T) whereas cytosine (C) always bonds to guanine (G), forming base pairs. A DNA can be specified uniquely by listing its sequence of nucleotides, or base pairs. In

bioinformatics, the DNA is abstracted as a long text over a four-letter alphabet, each representing a different nucleotide: A, C, G and T. The genome is the complete set of DNA molecules inside any cell of a living organism that is passed from one generation to its offspring.

Proteins are the molecules that accomplish most of the functions of the living cell. A protein is a linear sequence of simpler molecules called amino acids. Twenty different amino acids are commonly found in proteins, and they are identified by a letter of the alphabet or a three-letter code. Like the DNA, proteins are conveniently represented as a string of letters expressing its sequence of amino acids. A gene is a contiguous stretch of genetic code along the DNA that encodes a protein.

2.2. Bioinformatics Tasks

In this subsection, we illustrate the major interests in the bioinformatics, including sequence analysis, phylogeny inference, and protein 3D structure prediction.

2.2.1 Sequence Analysis and Alignments

Sequence analysis, the study of the relationships between the sequences of biological data (e.g., nucleotide and protein), is perhaps the most commonly performed tasks in the bioinformatics. Sequence analysis can be defined as the problem of finding which parts of the sequences are similar and which parts are different. By comparing their sequences, researchers can gain crucial understanding of the biological significance and functionality of genes and proteins: high sequence similarity usually implies significant functional or structural similarity while sequence differences hold the key information of diversity and evolution.

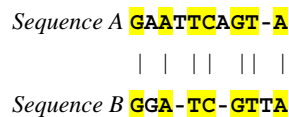


Fig. 1. Alignment of two sequences (The aligned sequences match in seven positions)

The most commonly used sequence analysis technique is sequence alignment. The idea of aligning two sequences (of possibly different sizes) is to write one on top of the other, and break them into smaller pieces by inserting gaps (“-”) in one or the other so that identical subsequences are eventually aligned in a one-to-one correspondence. In the end, the sequences end up with the same size. Figure 1 illustrates an alignment between the sequences A = “GAATTCAGGTA” and B= “GGATCGTTA”. The objective is to match identical subsequences as far as possible. In the example, the aligned sequences match in seven positions.

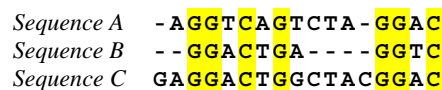


Fig. 2. Multiple DNA sequence alignment

When using a given sequence to find similar sequences in a database, one very often obtains many sequences that are significantly similar to the query sequence. Comparing each and every sequence to every other in separate processes may be possible when one has just a few sequences, but it quickly becomes impractical as the number of sequences increases. Multiple sequence alignment compares all similar sequences in one single step: all sequences are aligned on top of each other in a common coordinate system. In this coordinate system,

each row is the sequence for one DNA or protein, and each column is the same position in each sequence. Figure 2 illustrates a multiple alignment among the sequences A = “AGGTCAGTCTAGGAC”, B= “GGACTGAGGTC”, and C=“GAGGACTGGCTACGGAC”.

2.2.2 Molecular Phylogeny Analysis

Biologists estimate that there are about 5 to 100 million species of organisms living on earth today. Evidence from morphological, biochemical, and gene sequence data suggests that all organisms on earth are genetically related. Molecular phylogeny is the inference of lines of ancestry for organisms based on DNA or protein sequences of those organisms. The genealogical relationships of living things can be represented by an evolutionary tree. In the evolutionary trees, the relationships among the species are represented, with the oldest common ancestor as the trunk or “root” of the tree. The real problem is that of determining just how close or distant the relationship is. Bioinformatics phylogeny analysis tools provide crucial understanding about the origins of life and the homology of various species on earth.

2.2.3 Protein Structure Prediction

A protein sequence folds in a defined three-dimensional structure, for which, in a small number of cases, the coordinates are known. The determination of the three-dimensional structures of the proteins is of great significance for many questions in biology and medicine. For example, knowing how a protein is arranged in the cell membrane helps us to understand how they work and can lead to understanding not only the cause, but also eventually to the cure for virus infections, such as the common cold. Bioinformatics protein analysis tools translate the chemical composition of proteins into their unique three-dimensional native structure.

3. Experimental Methodology

This section describes the workloads and the methodology we used in this study.

3.1. Simulation Framework

Our experimental framework is based on the Trimaran system designed for research in instruction-level parallelism [10]. Trimaran uses the IMPACT compiler [11] as its front-end. The IMPACT compiler performs C parsing, code profiling, block formation and traditional optimizations [12]. It also exploits support for speculation and predicated execution using superblock [13] and hyperblock [14] optimizations. The Trimaran back-end ELCOR performs instruction selection, register allocation and machine dependent code optimizations for the specified machine architecture. The Trimaran simulator generator generates the simulator targeted for a parameterized VLIW microprocessor architecture.

3.2. Bioinformatics Workloads

To characterize the architectural aspects of the representative bioinformatics software, we use six popular bioinformatics tools in this study. This subsection provides a brief description of the experimented workloads.

Fasta: *Fasta* [15] is a collection of popular bioinformatics searching tools for biological sequence databases. These tools perform a fast protein comparison or a fast nucleotide comparison using a protein or DNA sequence query to a protein or DNA sequence library.

Clustal W: *Clustal W* [16] is a widely used multiple sequence alignment software for nucleotides or amino acids. It produces biologically meaningful multiple sequence alignments of divergent sequences. It calculates the best match for the selected sequences, and lines them up so that the identities, similarities and differences can be seen.

Hmmer: *Hmmer* [17] employs hidden Markov models (profile HMMs) for aligning multiple sequences. Profile HMMs are statistical models of multiple sequence alignments. They capture position-specific information about how conserved each column of the alignment is, and which residues are likely.

Phylip: *Phylip* (PHYLogeny Inference Package) [18] is a package of the widely used programs for inferring phylogenies (evolutionary trees). Methods that are available in the package include parsimony, distance matrix, maximum likelihood, bootstrapping, and consensus trees. Data types that can be handled include molecular sequences, gene frequencies, restriction sites and fragments, distance matrices, and discrete characters. In this study, we use *dnapenny*, a program that performs branch and bound to find all most parsimonious trees for nucleic acid sequence.

POA: *POA* [19] is sequence alignment tool. POA uses a graph representation of a multiple sequence and can itself be aligned directly by pairwise dynamic programming, eliminating the need to reduce the multiple sequence to a profile. This enables the algorithm to guarantee that the optimal alignment of each new sequence versus each sequence in the multiple sequence alignment will be considered.

Predator: *Predator* [20] predicts the secondary structure of a protein sequence or a set of sequences based on their amino acid sequences. Based on the amino acid sequence and the spatial arrangement of these residues the program can predict regions of alpha-helix, beta-sheets and coils.

Table 1. Benchmark description

Program	Description	Input Dataset
<i>fasta</i>	compare a protein/DNA sequence to a protein/DNA database	human LDL receptor precursor protein, <i>nr</i> database (the primary database from NCBI)
<i>clustalw</i>	progressively align multiple sequences	317 <i>Ureaplasma's</i> gene sequences from the <i>NCBI Bacteria</i> genomes database
<i>hmmer</i>	align multiple proteins using profile HMMs	a profile HMM built from the alignment of 50 globin sequences, uniprot_sprot.dat from the <i>SWISS-PROT</i> database
<i>dnapenny</i>	find all most parsimonious phylogenies for nucleic acid sequences	ribosomal RNAs from bacteria and mitochondria
<i>poa</i>	sequence alignment using Partial Order Graph	317 <i>Ureaplasma's</i> gene sequences from the <i>NCBI Bacteria</i> genomes database
<i>predator</i>	predict protein secondary structure from a single sequence or a set of sequences	100 <i>Eukaryote</i> protein sequences from NCBI genomes database

Table 1 summarizes the experimented workloads and their input data sets. We use the highly popular biological databases, including *nr* (the primary database from The National Center for Biotechnology Information (NCBI) [21]) and *SWISS-PROT* (an annotated biological sequence database from the European Bioinformatics Institute (EBI) [22]). The two multiple sequences alignment tools (*clustalw* and *POA*) use the same input data set: the 317 *Ureaplasma's* gene sequences from the *NCBI Bacteria* genomes database [23]. The input for the protein structure prediction tool predator is the 100 *Eukaryote* protein sequences from the NCBI genomes database.

3.3. Machine Configuration

The simulated machine architecture comprises a VLIW microprocessor core and a two-level memory hierarchy. The VLIW processor exploits instruction level parallelism with the help of compiler to achieve higher instruction throughput with minimal hardware. The core of the CPU consists of 64 general purpose registers, 64 floating point registers, 64 predicate registers, 64 control registers and 16 branch registers. There is no support for register renaming like in a superscalar architecture. Predicate registers are special 1-bit registers that specify a true or false value. Comparison operations use predicate registers as their target register. The core can execute up to eight operations every cycle, one each for the eight functional units it has. There are 4 integer units, 2 floating point units, 1 memory unit and 1 branch unit. The memory unit performs load/store operations. The branch unit performs branch, call and comparison operations. The level-one (L1) memory is organized as separate instruction and data caches. The processor's level-two (L2) cache is unified. Table 2 summarizes the parameters used for the processor and memory subsystems.

Table 2. Machine configuration

VLIW Core	
Issue Width	8
General Purpose Registers	64, 32-bit
Floating-Point Registers	64, 64-bit
Predicate Registers	64, 1-bit (used to store the Boolean values of instructions using predication)
Control Registers	64, 32-bit (containing the internal state of the processor)
Branch Target Registers	16, 64-bit (containing target address and static predictions of branches)
Number of Integer Units	4, most integer arithmetic operations: 1 cycle, integer multiply 3 cycles, integer divide 8 cycles
Number of Floating Point Units	2, floating point multiply 3 cycles, floating point divide 8 cycles
Number of Memory Units	1
Number of Branch Units	1, 1 cycle latency
Memory Hierarchy	
L1 I-Cache	8KB, direct map, 32 byte/line, cache hit 1 cycle
L1 D-Cache	8KB, 2-way, 32 byte/line, cache hit 1 cycle
L2 Cache	64KB, 4-way, 64 byte/line, L2 hit 5 cycles, 35 cycles external memory latency

4. Results

This section presents a detailed characterization of the VLIW processor running the bioinformatics software. Unless specified, all the applications are compiled with the IMPACT compiler with the maximum `-O4` option to produce optimized code for the VLIW processor. All benchmarks are run to the completion of 1 billion instructions. We examine benchmark basic characteristics, the efficiency of VLIW execution, the impact of function units, cache behavior and the impact of compiler optimizations.

4.1. Benchmark Basic Characteristics

Figure 3 shows the dynamic operations mix of the examined bioinformatics tools. The dynamic operations are broken down into seven categories: branches, loads, stores, integer (ialu) and floating point (falu) operations, compare-to-predicate (cmp) operations and prepare-to-branch (pbr) operations. Prepare-to-branch operations are used to specify the target address and the static prediction for a branch ahead of the branch point, allowing a prefetch of instructions from the target address. Compare-to-predicate operations are used to compute branch conditions, which are stored in predicate registers. Branch operations test predicates and perform the actual transfer of control.

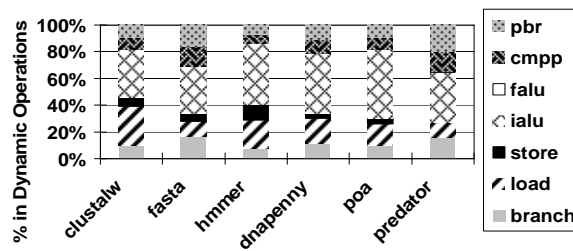


Fig. 3. Dynamic Operations Mix

As can be seen, load and integer operations dominate the dynamic operations in the studied bioinformatics programs. Overall, loads and integer computations account for more than 60% of dynamic operations. Only 5% of operations executed are stores. Stores occur only when updating the dynamic data structures such as HMM (e.g. on *hmmer*) and alignment score matrices (e.g. on *clustalw*, *fasta* and *POA*). Branches constitute 12% of operations executed. Additionally, there are 11% compare-to-predicate and 13% of prepare-to-branch operations in the dynamics operations. The experimented workloads all contain negligible (less than 0.1%) floating point operations, suggesting that floating point units are under-utilized. Therefore, the bioinformatics embedded processors may remove the costly and power-hungry floating point units and use software emulation for the floating point execution.

4.2. ILP

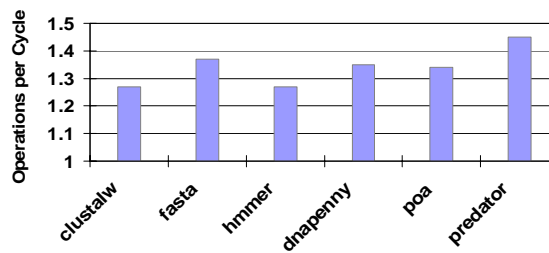


Fig. 4. Baseline ILP

Figure 4 shows the number of dynamic operations completed each cycles on the VLIW machine. To quantify the baseline ILP performance, we use the classic compiler optimizations and assume the perfect caches. As can be seen, although the VLIW machine can support 8 operations every cycle, on the average, only 1.3 operations are completed per cycle. The processor baseline OPC (operations per cycle) ranges from 1.27 (*clustalw*) to 1.45 (*predator*).

This indicates that control and data dependencies between operations limit the available ILP. Using conventional code optimizations and scheduling methods, VLIW processors can not attain the targeted ILP performance on the studied bioinformatics software.

4.3. Impact of Function Units

On the VLIW processors, the number and type of function units affects the available resources for the compiler to schedule the operations. The presence of several instance of certain function unit allows the compiler to schedule several operations using that unit at the same time. Figure 3 shows that the integer and memory operations dominate the bioinformatics software execution. We investigate the impact of the integer and memory units on the benchmark performance in this subsection.

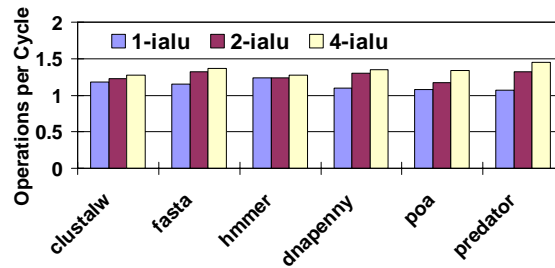


Fig. 5. Impact of Integer Units

We vary the number of integer units from 1 to 4 while keeping other architectural parameters at their default values. Figure 5 shows the impact of integer units on the ILP performance. As can be seen, increasing the number of integer units provides a consistent performance boost on the integer computation intensive benchmarks, since it permits greater exploitation of ILP by providing larger schedulable resources. When the number of integer units increase from 1 to 4, the processor ILP performance increases from 2.4% (*hmmer*) to 35.5% (*predator*), with an average of 16%.

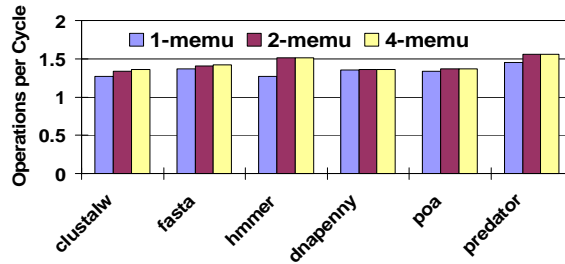


Fig. 6. Impact of Memory Units

We also perform experiments varying the number of memory units. Similarly, when we vary the memory units, the setting of other machine parameters are fixed. Figure 6 shows the impact of the memory units on the performance. We find that adding more memory units does not effectively improve performance on the majority of workloads. Increasing the memory units from 1 to 2 provides a performance gain ranging from 1% (*dnapenny*) to 19% (*hmmer*). Further increasing the memory units from 2 to 4 yields negligible performance improvement. The traditional compiler optimizations lack the capability of scheduling the instructions across the basic block boundaries, making the added memory units underutilized.

4.4. Cache Performance

This section studies the cache behavior of bioinformatics applications. Figure 7 shows the variation in the cache miss rates with cache associativity. We vary the cache associativity from 1 to 4, keeping the sizes of the L1 I-cache, L1 D-cache and L2 cache fixed at their default values. Cache misses are further broken down into conflict, capacity and compulsory misses. As can be seen, direct map instruction caches yield high miss rates on nearly all of the studied benchmarks. The conflict misses due to the lack of associativity dominate the cache misses. The instruction cache miss rates drop significantly with the increased associativity: the 4-way set associative, 8KB L1 I-cache shows a miss rate of less than 1%. This indicates that bioinformatics applications usually have small code footprints. A small, highly associative instruction cache can attain good performance on the bioinformatics applications.

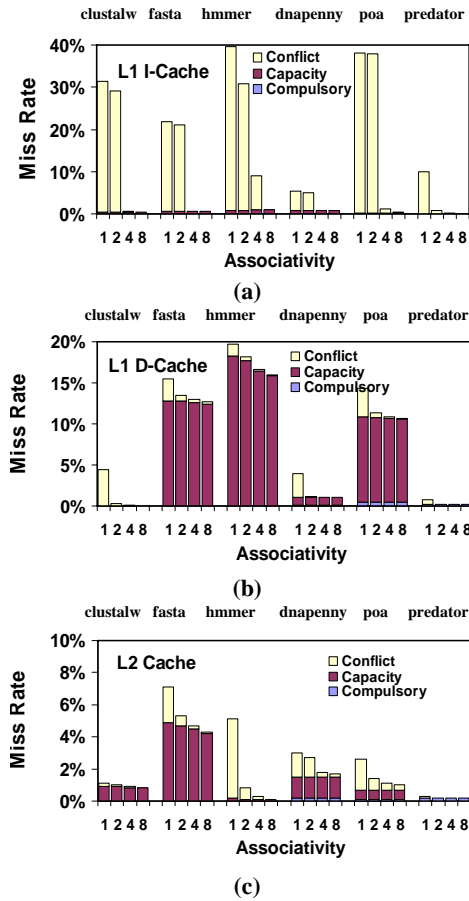


Fig. 7. Impact of Cache Associativity

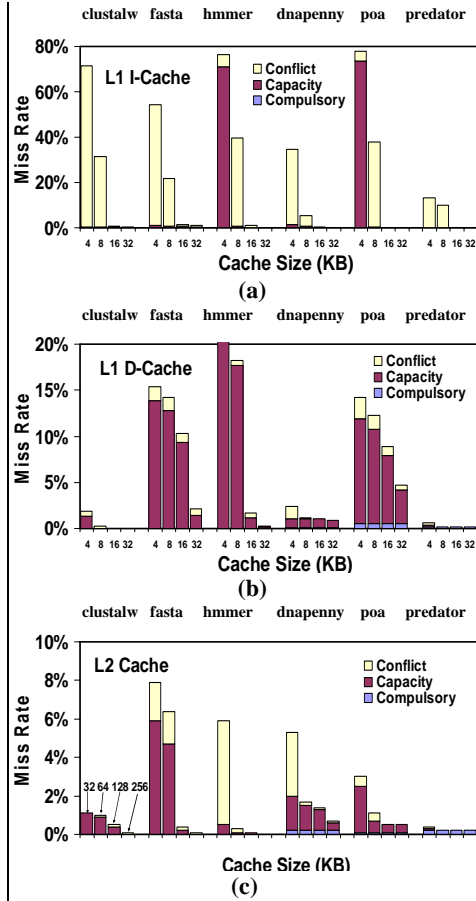


Fig. 8. Impact of Cache Size

Compared with instruction misses, data misses are difficult to absorb, even with the high cache associativity. Figure 7 (b) shows that on the 8-way L1 data cache, the miss ratios exceed 12% on benchmarks *fasta*, *hmmer* and *POA*. Unlike the instruction misses, the data cache misses are dominated by the capacity misses. This is because sequence alignment programs

normally work on large data sets with little data reuse. Figure 7 (c) shows that the L2 misses on five out of six programs are dominated by the capacity misses, suggesting that most of the L2 misses are caused by data references. Increasing the L2 cache associativity does not seem to be very helpful.

We also perform experiments varying the size of the caches. When we vary the L1 instruction cache, the sizes of L1 data cache and L2 cache are fixed. The associativity of the L1 I-cache, L1 D-cache and L2 cache are set to be direct-map, 2-way and 4-way. Figure 8 plots the cache miss rates for a range of varied cache sizes. As can be seen, on a direct-map instruction cache, increasing the cache sizes from 8K to 16K can nearly eliminates all the cache misses. For data references, a 32K L1 cache can achieve good hit ratios across all the benchmarks. The large working sets of *fasta*, *hmmer* and *POA* cause substantial traffic to the L2 cache. The entire working sets can be captured by a L2 cache with a size of 256K Byte. Larger input data would increase the working set requiring larger caches.

4.5. Compiler Optimizations

A compiler for VLIW processors must expose sufficient instruction-level parallelism (ILP) to effectively utilize the parallel hardware. This subsection examines the impact of compiler optimizations on the bioinformatics software execution.

We first investigate the impact of basic compiler optimizations on the benchmark performance. The IMPACT compiler provides a set of classic optimizations such as constant propagation, copy propagation, constant folding, and strength reduction. These optimizations do not necessitate any additional microarchitectural support. On the IMPACT compiler, level 0 option does not contain any optimization. Level 1 option contains local optimizations. Level 2 option contains local and global optimizations. Level 3 option contains local, global and jump optimizations. Level 4 option contains local, global, jump and loop optimizations.

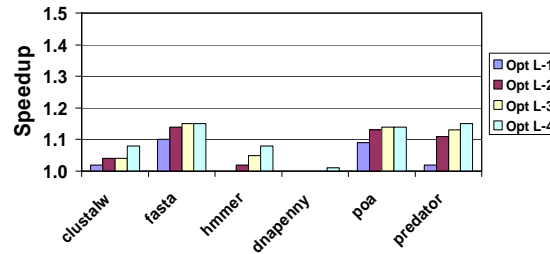


Fig. 9. Impact of Basic Compiler Optimizations

Figure 9 shows the effectiveness of using classic compiler optimizations. The program execution time (presented in terms of speedup) is normalized to that with no compiler optimizations. As can be seen, the basic compiler optimizations provide a speedup ranging from 1.0X to 1.15X. The classic compiler optimizations yield limited ILP performance improvement.

More aggressive compiler optimization technique can be used to further exploiting ILP. The IMPACT compiler provides two types of such optimizations: superblock and hyperblock optimizations. The superblock optimizations [13] form superblocks, add loop unrolling and compiler controlled speculation, in addition to the basic block optimizations. Compiler controlled speculation allows greater code motion beyond basic block boundaries, by moving instructions past conditional branches. Hyperblock optimizations [14] add predicated execution (conditional execution/if-conversion) to superblock optimizations. Predicated execution can eliminate all non-loop backward branches from a program.

Figure 10 shows the speedups of program execution due to superblock and hyperblock optimizations. The data is normalized to that of using the basic compiler optimization. Figure 10 shows that compared to the basic block optimizations, the superblock optimizations further yield speedups ranging from 1.1X to 1.8X. The hyperblock optimization results in speedups ranging from 1.1X to 2.0X. On the average, superblock and hyperblock optimizations improve performance by a factor of 1.3X and 1.5X. These speedups present an opportunity for improving the efficiency of VLIW execution on the bioinformatics applications.

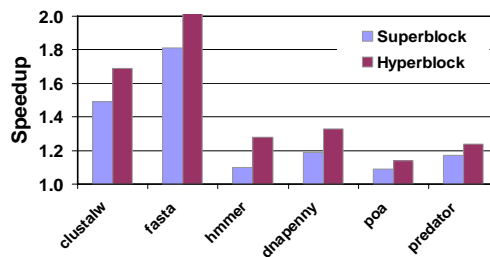


Fig. 10. Impact of Aggressive Compiler Optimizations

5. Conclusions

In the near future, bioinformatics and computational biology are expected to become one of the most important computing workloads. Bioinformatics applications usually run on the high-end systems with general purpose processors like superscalar. The rapidly growing market and the increasingly intensive competition between manufactures require the cost-effective bioinformatics computing platforms. The programmable, application-specific embedded processors and systems appear to be an attractive solution in terms of cost, power, size, time-to-market and easy of programming.

In order to design the complexity/cost effective processors and specialize hardware and software for the genetic information processing needs, a detailed study of the representative bioinformatics workloads on the embedded architecture is needed. This paper studies how the VLIW and compiler perform to extract the instruction level parallelism on these emerging workloads. The workloads we used include the popular DNA/protein sequence analysis, molecular phylogeny inference and protein structure prediction tools. Characteristics including operation frequencies, impact of function units, cache behavior, and compiler optimizations are examined for the purposes of defining the architectural resources necessary for programmable bioinformatics processors.

The major observations are summarized as follows: Loads and integer operations dominate bioinformatics applications execution. Floating point unit is underutilized. The baseline ILP performance is limited on the studied bioinformatics applications due to the data and control dependences in the instruction flow. A small, set-associative instruction cache can handle instruction footprints of bioinformatics applications efficiently, suggesting that bioinformatics applications have good locality and small instruction footprints. For the L1 data cache, capacity misses dominate the cache miss, suggesting that the bioinformatics applications have poor data locality. Therefore, in the L1 data cache design, increasing capacity is more efficient than increasing associativity. Classic compiler optimizations provide a factor of 1.0X to 1.15X performance improvement. More aggressive compiler optimizations such as superblock and hyperblock optimizations provide additional 1.1X to 2.0X performance enhancement, suggesting that they are important for the VLIW machine to sustain the desirable performance on the bioinformatics applications.

In the future, we plan to explore new architectural and compiler techniques for VLIW processors to support bioinformatics workloads. We also plan to expand our study to include other bioinformatics applications such as molecular dynamics, gene identification, protein function assignment, and microarray data analysis.

References

- [1] D. E. Krane and M. L. Raymer, *Fundamental Concepts of Bioinformatics*, ISBN: 0-8053-4633-3, Benjamin Cummings, 2003.
- [2] The Human Genome Project Information, http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml
- [3] Bioinformatics Market Study for Washington Technology Center, Alta Biomedical Group LLC, www.altabiomedical.com, June 2003.
- [4] SGI Bioinformatics Performance Report, <http://www.sgi.com/industries/sciences/chembio/pdf/bioperf01.pdf>
- [5] The Apple Workgroup Cluster for Bioinformatics, http://images.apple.com/xserve/cluster/pdf/Workgroup_Cluster_PO_021104.pdf
- [6] BioSpace, <http://www.biospace.com/>
- [7] Genomeweb Daily News, <http://www.genomeweb.com/>
- [8] A. S. Berger, *Embedded Systems Design - An Introduction to Processes, Tools, & Techniques*, ISBN 1-57820-073-3 CMP Books, 2002
- [9] P. Faraboschi, J. Fisher, G. Brown, G. Desoli, F. Homewood, Lx: A Technology Platform for Customizable VLIW Embedded Processing, In the Proceedings of the International Symposium on Computer Architecture, 2000
- [10] The Trimaran Compiler Infrastructure, <http://www.trimaran.org>
- [11] W.W. Hwu et.al. The IMPACT project, <http://www.crhc.uiuc.edu/IMPACT>
- [12] A.V. Aho, R. Sethi, J.D. Ullman. *Compilers: Principles, Techniques and Tools*, Pearson Education Pte. Ltd., 2001.
- [13] W.W. Hwu and S. A. Mahlke, The Superblock: An Effective Technique for VLIW and Superscalar Compilation, In the *Journal of Supercomputing*, page 224-233, May 1993.
- [14] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, R. A. Bringmann, Effective Compiler Support for Predicated Execution Using the Hyperblock, In the *International Symposium on Microarchitecture*, 1994.
- [15] D. J. Lipman and W. R. Pearson, Rapid and Sensitive Protein Similarity Searches, *Science*, vol. 227, no. 4693, pages 1435-1441, 1985.
- [16] J. D. Thompson, D.G. Higgins, and T.J. Gibson, Clustal W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice, *Nucleic Acids Research*, vol. 22, no. 22, pages 4673-4680, 1994.
- [17] S. R. Eddy, Profile Hidden Markov Models, *Bioinformatics Review*, vol. 14, no. 9, page 755-763, 1998.
- [18] J. Felsenstein, PHYLIP - Phylogeny Inference Package (version 3.2), *Cladistics*, 5: 164-166, 1989.
- [19] C. Lee, C. Grasso and M. F. Sharlow, Multiple Sequence Alignment using Partial Order Graphs, *Bioinformatics*, vol. 18, no. 3, pages 452-464, 2002.
- [20] D. Frishman and P. Argos, 75% Accuracy in Protein Secondary Structure Prediction, *Proteins*, vol. 27, page 329-335, 1997.
- [21] NCBI, <http://www.ncbi.nlm.nih.gov/>
- [22] The UniProt/Swiss-Prot Database, <http://www.ebi.ac.uk/swissprot/>
- [23] The NCBI Bacteria genomes database <ftp://ftp.ncbi.nih.gov/genomes/Bacteria/>.