ISSN:0975-9646

FINE TUNING OF PHYLIP ON INTEL XEON ARCHITECTURE

Prinkesh Sharma[1], Kondorpa Kumar Borchetia[2], Shivam Saxena[3]

[1,2,3]*Dept. of Computer Science & Engineering,*

National Institute of Technology Silchar-788010, India

Sumeet Singh Bhambrah[4], Anneswa Ghosh[5] andJoydeep Chakraborty[6]

[4,5,6]*Dept. of Computer Science & Engineering,*

National Institute of Technology Silchar-788010, India

Abstract **-Computational Science is witnessing an exceptional growth over the years, but we are still lacking in efficient programming to effectively optimize these computations. In today's modern world, computations need to be done and results delivered in the least possible time. Porting, optimization, scaling and tuning of existing High Performance Computing (HPC) Applications on hybrid architectures is the norm for reaping the benefits of extreme scale computing. That being said, we must remember that the real gist in optimizing computations lies in properly tuning the core source code running on a single processor or a shared memory model within a node. This paper gauges the performance of PHYLIP application on Intel Xeon Processor.**

Keywords - High Performance Computing,Parallel Programming,Optimization,Phylogenetic

I. INTRODUCTION

HPC systems are becoming challenging in terms of speedup and scalability. This ever increasing complexity demands well-organized and flexible numerical algorithms to achieve high performance computing. The size of compute intensive problems also increases as the computing technology advances. In order to effectively gain performance, the basic serial code running on a single processor or shared memory model must be properly tuned using efficient optimization techniques.

Increases in the quantity of molecular sequence data available for analysis, with over 1,000 complete genomes available at NCBI's Genome Project Resource, has brought with it both the ability and need to perform phylogenetic analyses on larger, more complex data sets. PHYLIP package is intended for phylogenetic studies of protein and DNA sequences which provide unique and valuable insights into the molecular and genetic basis for important medical and epidemiological problems [1,2] as well as important questions about the origins and development of physiological features in present day organisms [3,4].

In this paper, we have explored the possibilities of optimizing the proml program in the PHYLIP package when executed on an Intel Xeon processor. Optimization of proml was taken more into consideration because of large time required by this code for large datasets. [5]

A. About PHYLIP

PHYLIP is a comprehensive phylogenetic analysis package created by Joseph Felsenstein at the University of Washington. The PHYLIP package is one of the most comprehensive sets of tools freely available for use in phylogenetic studies [6]. This package can do many of the phylogenetic analyses available in today's world. The package consists of methods like parsimony and distance matrix as well as likelihood methods. Molecular sequences, restriction sites, gene frequencies, distance matrices and 0/1(binary) discrete characters are the data types that can be handled by PHYLIP.

1) Installation: PHYLIP is freely available from:http://evolution.genetics.washington.edu/phylip.html.It ships with a comprehensive manual covering the usage of different programs. The manual can be read from a web browser as the manual is written in HTML format .For Windows users, installation is simple.One must download the three zip-files (phylip.exe, phylipwx.exe, phylipwy.exe), and extract them to a preferred folder. The subfolder exe contains all the programs. Manual can be found from the subfolder doc.

For Macintosh OS X you may download the packaged disk image (Phylip3.66.dmg). It is compressed, so you need to expand it, and copy the resulting folder to a desired location. Alternatively, you may compile the programs from their sources as outlined in the UNIX installation below. There are source codes and readymade compilations available for older Macintosh systems.

Installation for UNIX systems is also quite straight-forward. These instructions apply for RedHat-based Linux systems. Installation on main frame can require tweaking of the Makefile. Download the source code and documentation package (phylip-3.66.tar.gz) into a suitable folder. Unzip the package with gzip utility (gzip –d phylip-3.66.tar.gz) and expand the tar ball (tar xvf phylip-3.66.tar). Move to the newly formed folder containing the source codes (cd phylip3.6/src). The folder contains a file called Makefile. Installation of the PHYLIP programs is done simply by typing make install. The default Makefile usually works fine.

2) User Interface: PHYLIP has a menu driven user interface, which allows the user to set the options and start computation. The data are fed into the program from a text file, which can be prepared using any word processor or text editor (but it is crucial that the text file is not in the special format of that word processor – instead, it should be in flat ASCII or Text Only format). ClustalX and T-Coffee are some sequence alignment programs that can write data files in the PHYLIP format. Most of the programs search for the data in a file called infile. In case this file is not found, then, they then ask the user to type in the file name of the data file. Output is written onto special files like outfile and outtree. Trees written onto outtree are in the Newick format, an informal standard agreed to in 1986 by authors of a number of major phylogeny packages (Felsenstein, PHYLIP documentation).

3) Running Phylip Programs: The programs are used sequentially. The output from the first program is fed as an input to the next program. The trick lies in knowing how to use the programs in suitable combinations. Most PHYLIP programs run in similar manner. The input for a program is taken from the infile. If the program is unable to find the infile, it then requests the user to type in the file name of the data file. The outputs are written in a file called outfile. Some programs may produce both: outfile and a file called out tree or plot file. Because most programs use default names for input and output files, the user needs to be sure to rename the files which have to be saved before proceeding, else there is a risk losing the results.

The following data flow charts describe some basic analyses:

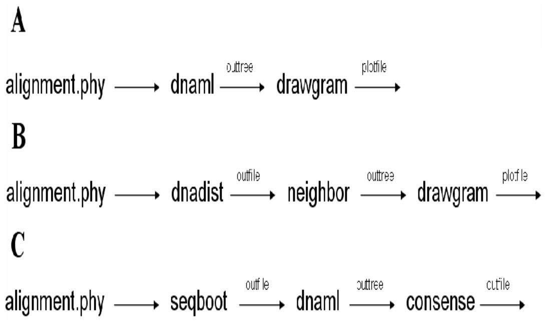

Fig 1 Program sequence

II. Experiment in Detail

Maximum likelihood analysis for DNA sequences is depicted in flow chart A.
Analysis using Neighbour joining method for DNA sequences is depicted in flow chart B.
Bootstrapping analysis for DNA sequences using maximum likelihood as the analysis method is depicted in flow chart C.

4) Methods Used for Optimization: The code is analysed for finding out time and space consuming hotspots using profilers (gprof,etc) and Intel Vtune Amplifier XE. Once profiling is done and hotspots identified, we use different optimization techniques in these regions and try to maximize the performance of the program.

Uniprocessor optimization can be done by implementing techniques like Loop Unrolling, better Cache Hit, Loop Fusion, Loop Splitting, Inlining functions.Thread level parallelism can be implemented by using OpenMP. OpenMP is an Application Program Interface (API) that maybe used to explicitly direct multi-threaded, shared-memory parallelism.

5) Hardware and Software Configuration:

TABLEI
Hardware Configuration

Sr. No.	Cluster Parameter	Host Node Configuration
1	CPU	Intel(R) Xeon(R) E5-2650 v2
2	RAM	64 GB
3	Cores	16
4	OS	GNU/Linux 2.6.32-358.el6.x86_64
5	Thread/Core	1
6	Nodes	1

TABLE II
Software Configuration

Sr. No.	Name	Description
1	Intel Compiler	composer_xe_2013_sp1.2.144
2	Intel MPI Library	4.1.3.048
3	Intel VtuneAnalyzer	2013.1.046
4	PHYLIP	Ver 3.695

We have focused our work mostly on the proml program.Proml is a compute intensive protein maximum likelihood program [5], a part of PHYLIP package. To maximize the scalability and performance of this code on multi-processor systems and hybrid systems, it is required to optimize and maximize its performance on uniprocessor system or shared memory model system.

A. Dataset

The test datasets (containing 1,000 bootstrap replicates generated with the PHYLIP seqboot program) used to illustrate the performance of the parallel codes are: 1) a 375 residue, single-gene alignment of cytochrome B from the mitochondrial genome from thirteen species of plasmodium (1/13); 2) a 375 residue, single-gene alignment of cytochrome B from the mitochondrial genome from twenty five species of plasmodium (1/25); 3) a 1139 residue, three-gene alignment consisting of the cytochrome B, cytochrome oxydase 1, and cytochrome oxydase 3 genes from the mitochondrial genome from twenty five species of plasmodium (3/25); 4) a 389 residue single gene alignment of 60 ABCG transporters from a variety of species (1/60); and 5) a 356 residue, single gene alignment from a set of 121 g-protein alpha inhibitory subunits from fungi (1/121)

B. Phases of Experiment

1) *Profiling and Analysis of Code:* We analysed the proml code using Intel Vtune Amplifier XE.

We obtain the hotspots, which are regions of significant activity, and are more time consuming.

2) *Uniprocessor optimization:* Uniprocessor optimizations were carried on the hotspur regions obtained in phase 1. The hotspots were optimized to access memory in a cache optimized manner hence increasing cache hit resulting into better performance of code. Complex Mathematical formulas, which previously were calculated using loops, were modified such that now they were calculated using less nesting of loops hence increasing the performance further.

TABLE III

Benchmarking of the Proml Code on Intel(R) Xeon(R) E5-2650 v2 CPU of Frequency 2.60GHz, ie. Ivy Bridge Architecture without HT (Hyper Thread Enabling)

Code Version	**Execution times (seconds)**				
	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**	**Dataset5**
Without optimization	976	334.33	7.846	33.718	42.582
After Uniprocessor Optimization	573.538	203.927	5.501	22.579	25.731

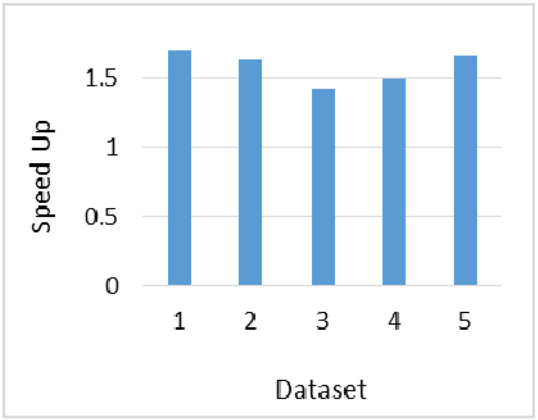

Fig. 2 Speed up on uniprocessor optimization

The code performance of proml programs run with default parameters is summarised in Figure 2.We can infer that an average speed-up of 1.5 was obtained when the proml code was executed for all the test data.

3) *Thread Level Parallelism Using OpenMP:* After doing the uniprocessor optimizations, our goal was to introduce parallelism wherever we could without affecting the output of the code. Parallelism was implemented using OpenMP wherever it was suitable and efficient. The proml code was executed for all the datasets varying the number of threads from 1 to 16.

TABLE IV

Benchmarking of the Proml Code on Intel(R) Xeon(R) E5-2650 v2 CPU of Frequency 2.60GHz, ie. Ivy Bridge Architecture without HT (Hyper Thread Enabling) by Varying Number of Threads.

		Code Version																
		Original Code	OpenMP and Uniprocessor optimizations implemented and executed taking "n" number of threads (Approximate time)															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Execution time (seconds)	Dataset 1	976	576	446	346	324	301	297	285	282	279	279	282	279	282	278	282	284
	Dataset 2	334.33	204	153	123	112	107	107	100	101	98	99	99	98	98	98	101	101
	Dataset 3	7.846	6	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	Dataset 4	33.718	23	19	15	13	13	13	12	13	12	12	12	12	12	12	12	12
	Dataset 5	42.582	26	20	18	17	17	17	17	17	18	17	17	18	18	19	19	18

Figure 3 shows that the performance of code is enhanced up to 233 % when the number of threads is increased to 4-8 and further increase in number of threads doesn't lead to any significant performance enhancement or degradation. Thus, optimal usage of CPU is obtained when the code is used with 4-8 numbers of threads.

Speed-up of optimized code w.r.t Original Code can be defined as the ratio of Execution time of original code to the Execution time of optimized code run on "n" threads.

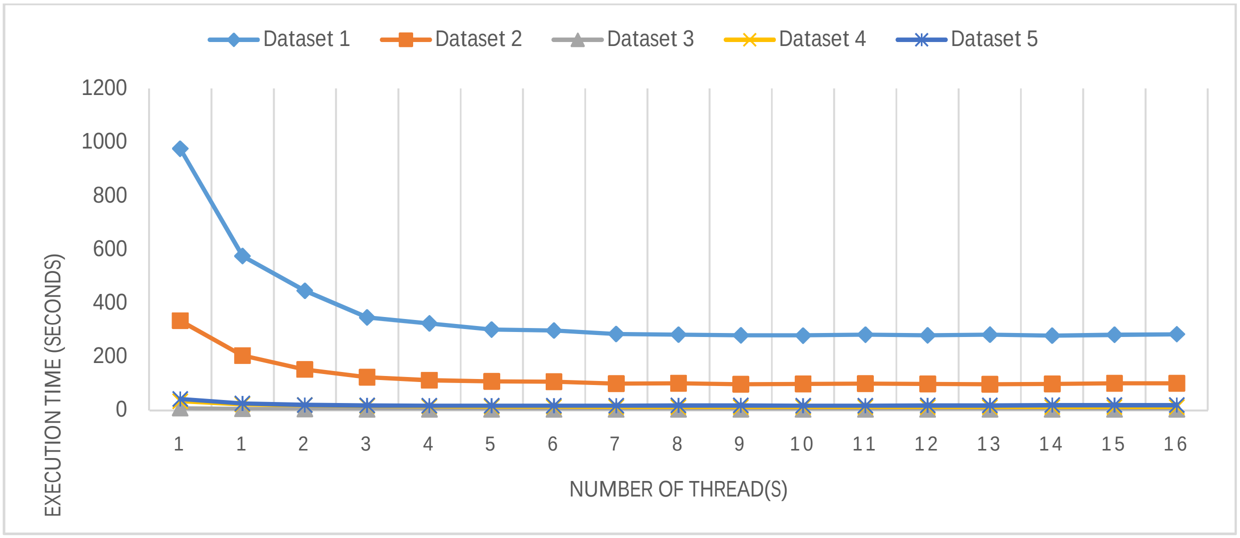

Fig 3 Analysis of execution time on all datasets w.r.t increase in number of threads

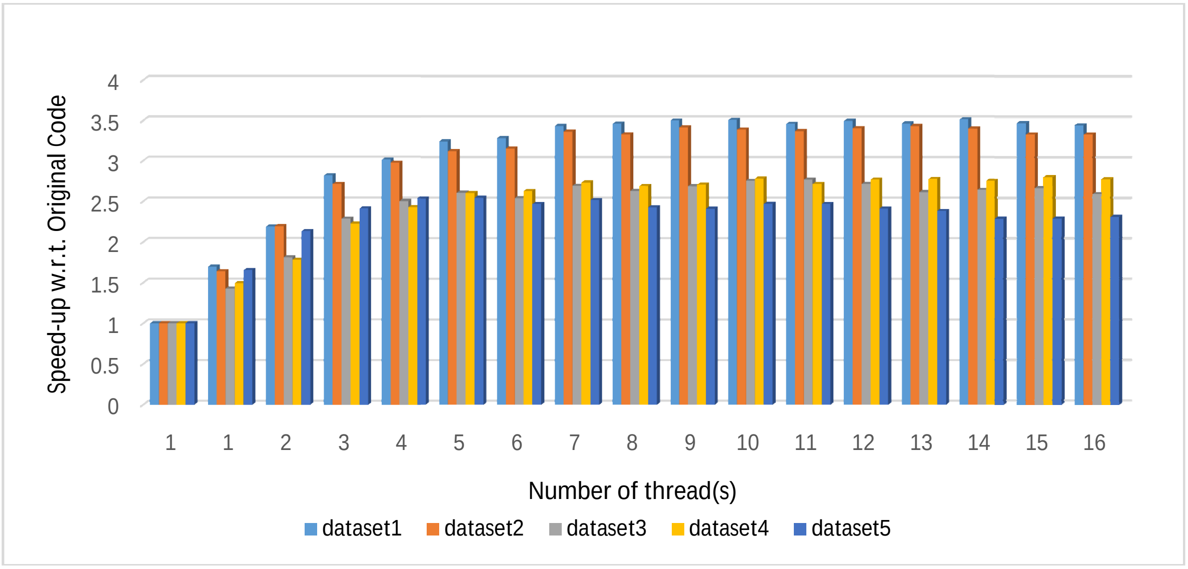

Fig. 4 Speed up w.r.t original code

The fig 4 shows the speed-up of optimized code with respect to the original code. For example, for the dataset1, a speed-up of 3.5 is obtained when executed with 10 threads means that it will take 3.5 times lesser time than the execution time of original code.

After analysing Figure 4, we can infer that with increasing number of threads the speed-up gradually increases and then attains a stable value in the range 2.3 to 3.5. Thus a better performance is achieved as compared to performance of Uniprocessor optimization (Refer: Fig 2). Speed-up obtained by execution on "n" threads w.r.t execution time on 1 thread can be defined as the ratio of Execution time on 1 thread to the Execution time on n threads.

The fig 5 shows the speed-up of execution of optimized code when executed on "n" threads with respect to the execution of optimized code executed on single thread.For example, for the dataset1, a speed-up of 2 is obtained when executed with 10 threads means that it will take 2 times lesser time than the execution time of optimized code executed on single thread.

After analysing Fig 5, we can infer that the code is scalable up to 200% and the speed up in performance becomes stable when the number of threads is increased to 8-9 and further increase in threads results is no significant change.

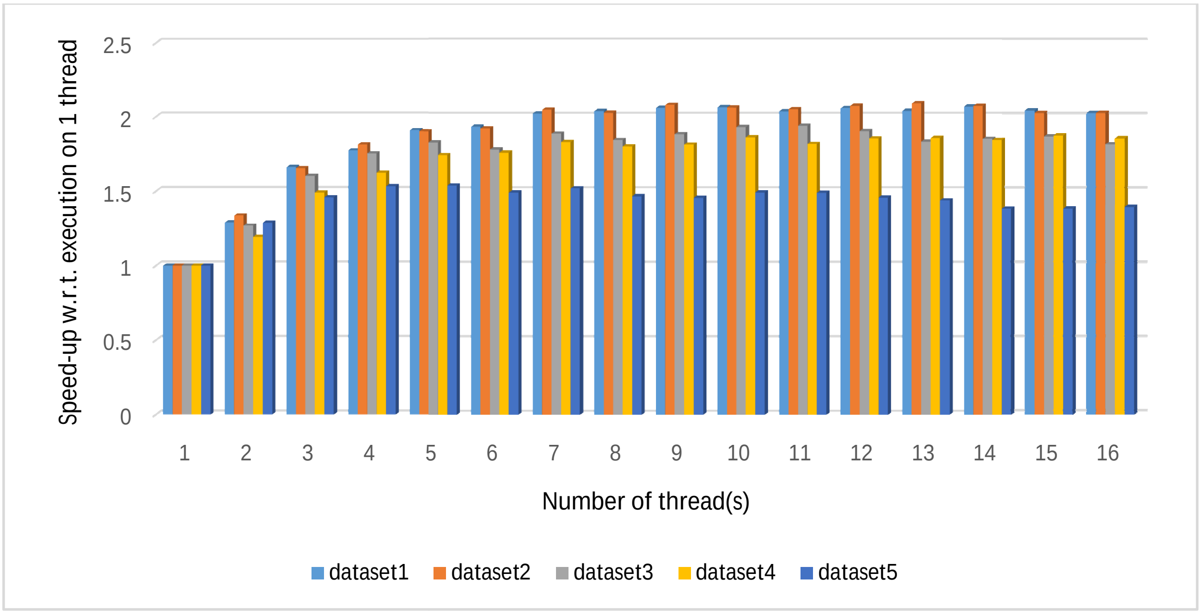

Fig 5 Speed up w.r.t to uniprocessor optimized code

III. Conclusions

From the above experiments it is concluded that a significant improvement of more than 200% is in achieved in performance of the the proml program of the PHYLIP package and the performance can be enhanced further by increasing the number of threads upto a certain point after which the performance enhancement more or less remains the same.

IV. Future work

The current optimization of PHYLIP application is limited to promlonly; whereas it has more than 30 other programs for DNA sequences, discrete characters, tree plotting, gene frequencies etc. The application can be ported onto Intel Xeon Phi MIC coprocessor and benchmark the performance of the PHYLIP application on Xeon Phi.There is also a possibility of taking the input data directly from the memory rather than reading the data file each time, which will affect the run time of the application in a very crucial manner. As observed, the output file is much larger than the input file given. Sometimes the output file has become 10 to 100 times larger than the given input file. Some compression algorithm may be applied to these output files in order to make them less memory consuming, as the compression can be obtained as the cost of CPU time.

ACKNOWLEDGEMENTS

We would like to extend our gratitude to Dr P.K. Sinha (Sr. Director Strategy, R&D, Centre for Development of Advanced Computing, Pune, India), Mr Abhishek Das,Mr.ManjunathaValmiki, Mrs.Shweta Das and the C-DAC members at NIT Silchar for their cordial support, valuable information and guidance provided by them, which helped us in moving ahead with this task through various stages. We are also very grateful for their cooperation during the period of our research. We would also like to thank Mr.B.Dey sir for his valuable advices and support.

References

[1] Sheps JA, Ralph S, Zhao Z, Baillie DL, Ling V (2004) The ABC transporter gene family of Caenorhabditiselegans has implications for the evolutionary dynamics of multidrug resistance in eukaryotes. Genome Biol 5: R15.

[2] Bridgham JT, Carroll SM, Thornton JW (2006) Evolution of HormoneReceptor Complexity by Molecular Exploitation. Science 312: 97–101.

[3] Durand D, Hoberman R (2006) Diagnosing duplications–can it be done? Trends Genet 22: 156–164.

[4] Darling AE, Miklo´ s I, Ragan MA (2008) Dynamics of Genome Rearrangement in Bacterial Populations. PLoS Genet 4: e1000128. doi:10.1371/journal. pgen.1000128.

[5] Alexander J. Ropelewskimail,Hugh B. Nicholas Jr,RicardoR.Gonzalez Mendez (2010, 15 Nov) MPI-PHYLIP: Parallelizing Computationally Intensive Phylogenetic Analysis Routines for the Analysis of Large Protein Families

[6] Felsenstein, J. (1981). "Evolutionary trees from DNA sequences: A maximum likelihood approach”.