International Journal of Recent Advances in Science and Technology, 2014; 1(1):1-5

Automatic Parallelization: Predicaments in Computationally Expensive Operations

Nahar Hashmukh Prayank*, Bhavsar, Dharmesh kumar Bhalchandra, Vishal Bhatnagar, Richa Tomer and S.K. Agarwal

Department of Computer Science, Shri Venkateshwara University, Gajraula Distt- Amroha (U.P), India

ABSTRACT

The applications developed for parallelizing are either single file based programs or algorithm structures. The commercial large scale application pose problems when performing automatic parallelization and needs to be addressed and hence thought to be implausible. This paper tries to surface the impediments to be addressed so that the parallelization techniques may be applied to these applications. Benchmark suite which are specifically developed to expose the computing requirement and are well acclaimed in the industry have been adopted. Benchmarks used are from High Performance Group of the Standard Performance rating Corporation (SPEC), both parallel and serial versions of applications are used.

Automatic parallel serial codes are compared with its parallel variants in this paper. Parallelizing compiler is employed that takes language (formula based) codes and inserts Open Multi-Processing directives around loops determined to be autonomous. Different problems were faced by an automatic parallelizing compiler when dealing with full applications procedures, optimizations based on superficial techniques, array input output and size variations ,Multilanguage hurdles, extreme inclination to library (system defined functions) and endowment accumulations. The results presented in this paper shall benefit parallelizing compilers with capabilities for handling large scale science and engineering applications.

Keywords: Subroutines, Procedures, Parallelization, Compilers.

Introduction

Any programming language, compiler, operating system and system architecture will finally have to improve upon its functionality and performance for applications that have commercial existence. In terms of line of codes commercial applications are generally voluminous. Benchmarks are available and are used in the field of systems research. Best are characterized by low execution times and availability in public arena. The runtime should be short as possible because under research a piece of code may be run number of times.

**Correspondence*
Nahar Hashmukh Prayank
Department of Computer Science, Shri Venkateshwara University, Gajraula Distt- Amroha (U.P), India

The results are only important if the program is available to everyone i.e. public arena availability so that they may be mimeographed.

This paper aims to advance automatic parallelization technology for computers systems possessing high performance. Programs with high end commercial application weightage are been adopted. Applications used are from SPEC benchmark suite. These applications are rather relevant in commercial arena and can be reproduced and shared candidly.

First application [1] was developed by ARCO beginning in 1993 to gain an accurate measure of the performance of computing systems and it relates to the seismic processing industry for procurement of new computing resources.

The second application, [2] is used to simulate molecules at the quantum level. It is a current research effort under the name of GAMESS at the Gordon Research Group of Iowa State University and is of interest to the pharmaceutical industry. Both application are extensively used to illustrate performance of high end - high performance computers.

This paper contributes in surfacing the eloquent obstacles posed from automatic parallelization to the most promising commercially accepted and driven applications. Polaris translator [3], an avant-grade parallelizing compiler to used .OpenMP was engaged and both the applications are manually parallelized. The results are thoroughly analyzed. Next Section, describes five categories of challenges faced by parallelizing compilers [4]. Sections 2.1 describe issues arising from the fact that large applications naturally have a very modular structure. Section 2.2 indicates the lingual hurdles due to excessive adoption of languages of different levels. Section 2.3 discusses about the endowment accumulation problem. Section 2.4 deals with the issue of array variations in terms of size and subroutine interactions; and Section 2.5 describes problems in the presence of input output operations. Section 4 concludes the paper.

Impediments of Automatic Parallelizing Compilers

High end applications shows little accomplishment after being acted upon by compiler tools. Hence the examples and programs shall present enhancement in present technologies of these compilers. The codes under consideration may be taken from full grown industry applications ,different techniques [5] available on parallelization of compilers performing mechanized parallelization.

Functionability

Applications performing variety of tasks and cater to different activities may be brought down or substituted with small codes called as modules. In software terminology the modularity is not only a way but also a tool. These self made modules are powerfully supported by the functions that are readily available from the library often known as library modules. Large applications have huge number of functions or modules calling and delivering at various junctures compelling both strong and little interdepencies. It becomes important for modular analysis to know in prior that at run time or under execution which modules will be called and in turn to which modules they will call.

Functions of Dynamic Application

There are two applications considered here and both of them have plentiful functions. Relevant functions are taken for executing. For instance the first application takes four steps or states namely Generation of data, Stacking of data, Migration of depth and Migration of time. The data that has been inputted decides which modules will run. Thus complier has to wait till the run time to induce about the modules that will come under scanner of execution.

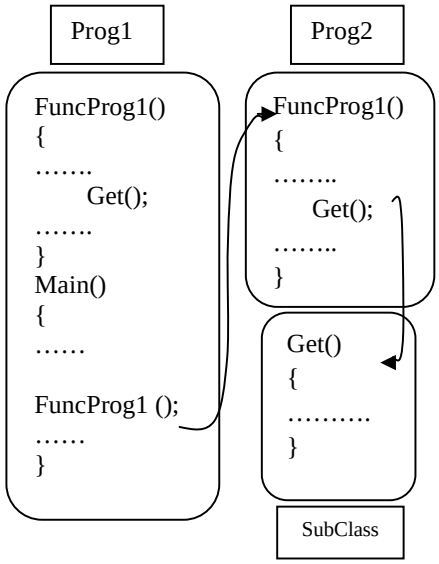


Figure 1: Modular Interdependency

In Figure-1 the progressive module calling is presented. One function call other which in turn call some other and in process some of them call themselves making it difficult to wrap up the decision of order in which they will be called due to extreme interdependencies.

Thus in order to clear the obstacle of inability of compilers to know in advance the flow of program control to different modules it requires a complete program and code knowhow's the problem in achieving automated parallelization requires the routine calling timestamps.

Extensible Libraries

The procedures or modules in these commercial large applications have a tendency for numerous schematic parameter list and different options or conditions.

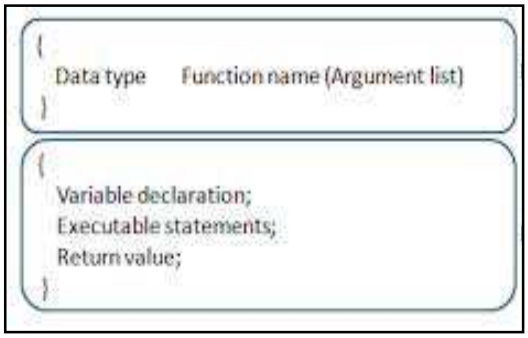


Figure 2: **Varied Arguments and Statements**

Applications of this sort are highly inclined to the definitions available in library modules. Figure-2 shows an example where the arguments list and different parts for a snippet that goes prodigious in applications under consideration.

Lingual Hurdles

The compiler transforms the higher level language into machine level code for executing on processor. Small application are generally in one language. Various languages of different flavors and importance have evolved notably the languages on objected oriented concepts.

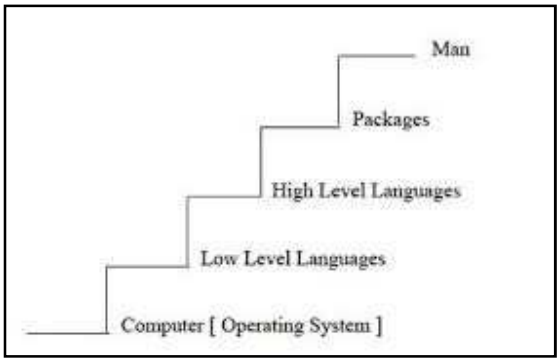


Figure 3: Multi-Lingual Components

In large application the modules and codes will be found in different languages (Figure-3). Hence, it becomes mandatory in order to achieve automatic parallelization the capability of interacting with multiple language exists.

Endowment Accumulation

The commercial application of SPEC under consideration uses different low level mathematical and scientific codes such as matrix multiplications, discrete Fourier transformations and others. Though these codes are polished in terms of performance but still poses difficulties in optimizations to the compiler.

Erstwhile generations for computer systems with big performances have designed options to hold the transformation difficulties in codes of endowments. These solutions may not be favorable under present circumstances. Specseis includes numerous lower level FFT routines mentioned in IEEE Press book of 1979.

The compiler can do its own endowments only when the endowments prior may be cancelled or retracted. The other solution can be that the compiler enhances its ability to negotiate with the modules and the intricacies posed by the accumulated endowments.

Array Variations and Type Change

For programs and applications those are highly procedural, interprocedural approaches holds a high value of importance.
The compiler under consideration accomplish the similar eventuality through subroutine inline extensions. Now the complication both the applications of SPEC under consideration face is related to arrays that may take any size, pattern and configuration for caller as well as called codes.

Array Variation

Different variables and datastructures are places in stack or heap depending on their declarations and usage in execution.

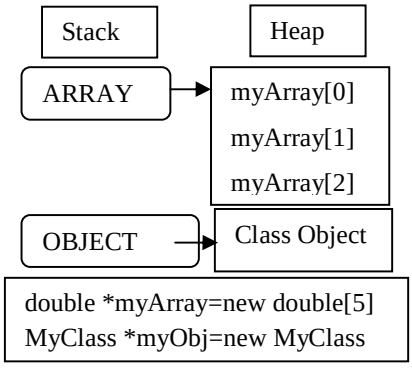
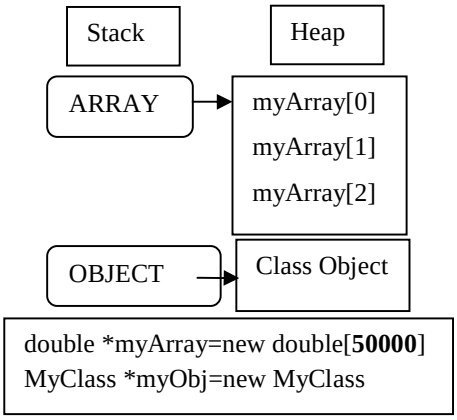


Figure 4: Array in Memory

The problem arises when the array as shown in Figure-4 is sent to different modules as reference and since the returned values may have a different value, the modules requiring the original value will be affected by this parallelism effect.



The other situation as in Figure-5 is when the array size is abnormally high and it is passed to several modules simultaneously. Modules with less running time may present result and the others will take a considerable amount of time and the results in either case will be ambiguous.

Transformations of Array Type

One more problem that is comprehend is some routines are using array that is defined as double and others are using integer defined arrays. The data type and the storage concern has also evolved to be a addressable complication.

Exits from Loop and IO

The large applications contains various functions, block of codes and statements which involves Input-Output statements such as Exit, Break, Continue, Abort, Goto. These statements will run never or under certain conditions. When various parts of application automatically parallelize the repercussions on the results on certain condition may prove fatal to the other modules or sometimes to the complete program.

Large-scale Applications Issues

A commercially large package shall produce considerable issues with the compiler which has been indicated in this paper. The problems and concerns raised are important to humongous applications but are also important to the applications with light nature and considerable small execution time.

Apart from the facts pointed in earlier sections the other relentless players that make the parallelization difficult to achieve are declarations, constants, pointers, memory allocations and reallocations.

Conclusion

Today till date the parallelization is still not completely automatic. But there are fine steps and approaches that may be employed in automatic parallelizing compilers to generate efficacious parallel code.
This paper surfaces the options where the compiler community may enact that enables the automatic parallelization constructive and valuable for large scale applications.

First concern is the extreme procedural nature of the languages and immense interdependence of these modules on the results of one another.
The second complication is the banking upon of the program and code on the library modules and functions.
The third issue is the present scenario of different programming languages functioning unitedly in close coupled nature.

Input/output operations are another problem to successful parallelization. The compiler has to have a recognizable application that certain I/O statements are called and run not so often or only under certain erroneous conditions.

The study presented in this paper is only a small step in the direction of understanding the characteristics of large scale applications. Analyzing such applications takes significant effort. Many more, similar studies

will be necessary to assist the current generation of compilers to become truly useful tools for the user of real world commercial applications and attaining fully automatic parallelization.

References

1. C. C. Mosher and S. Hassanzadeh. ARCO seismic processing performance rating suite, user s guide. Technical report, ARCO, Plano, TX , 1993.
2. Michael W. Schmidt et. Al. General atomic and molecular electronic structure system. Journal of Computational Chemistry, 1993;14(11):1347-1363.
3. W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. HoeYinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu. Parallel programming with Polaris IEEE Computer, 1996;29(12):78-82.
4. William Blume and Rudolf Eigenmann. An Overview of Symbolic Analysis Techniques Needed for the Effective Parallelization of the Perfect Benchmarks. Proceedings of the International Conference on Parallel Processing, pages II 233 II 238, August, 1994.
5. Rudolf Eigenmann, Insung Park, and Michael J. Voss. Are parallel workstations the right target for parallelizing compilers? In Lecture Notes in Computer Science, No. 1239: Languages and Compilers for Parallel Computing, pages 300 314, March 97.

Source of Support: NIL
Conflict of Interest: None