International Journal of Research in Advent Technology (E-ISSN: 2321-9637) Special Issue
National Conference "NCPCI-2016", 19 March 2016
Available online at www.ijrat.org

Serial to Parallel Code Converter Tools: A Review

Dr. Amit Barve[1], Saanchi Khandelwal[2], Nujhat Khan[3], Shamali Keshatiwar[4], Swapnali Botre[5]
Computer Engineering[1, 2, 3, 4,5],VIIT,PUNE [1, 2, 3, 4,5]
Email: barve.amit@gmail.com[1], saanchik94@gmail.com [2] , khan.nujhat@gmail.com[3], shamalikeshattiwar@gmail.com[4], swapnalibotre42@gmail.com[5]

Abstract- The arrival of multi-core architecture has extremely encouraged the area of parallel computing. This multi-core model has enforced a change in the way applications are written. Multi-core processors have taken the plunge to increase the performance of an application by utilizing the benefits of parallelization. However, including parallelism in a program is not an easy task. This requires parallel programming expertise. In addition to this manual parallelization of legacy serial codes is a strenuous and time consuming job. A number of tools have been developed to automate the process of parallel compilation. These tools use the parallel programming libraries to convert serial program into parallel one. This paper addresses various issues in the implementation of serial to parallel code conversion with existing techniques used to resolves issues. The paper also presents a survey of available tools used in serial to parallel code conversion, tools are compared on the basis of their features, and a few tools are compared on the basis of their performance.
.

Index Terms- Parallel Converter Tools, Multi-Core Machines, Parallelization Techniques.

I. INTRODUCTION

Parallel architectures have been advancing over the past few decades. According to Hall et al [1], in the present decade i.e. 2010 to 2020 compiler research will play a critical role in addressing two of the major challenges facing the overall computer field:

- Security and reliability of complex software systems.
- Cost of programming multi-core processors.

The clock rate of the machine has almost reached its theoretical limit, in spite of that the machine speed is still continuing to increase impressively due to increased parallelism in the form of multi-core execution units. This trend of parallel architecture is one of the greatest challenges for the processor and software industries. Initially, only scientific researchers were in need of parallel programming. But now-a-days even general programmers are also interested in parallel programming. The invention of multi-core increased the speed of the processor. But this development of multi-core processors accompanied with it a disadvantage as most of the legacy applications or software's are written in sequential manner and therefore are unable to utilize the complete power of multi-core. This limitation urges the use of parallel programming. Also providing training in parallel programming is a difficult task if time and cost are to be considered. Therefore, there is a need of assistance tools for parallel programming. These tools take the serial code as input and generate output code with parallelization constructs. These tools help in increasing the performance and efficiency of a system.

II. ISSUES IN IMPLEMENTATION OF SERIAL TO PARALLEL CODE CONVERTER

The following issues need to be address before designing a serial to parallel code converter.

1. ***Dependency Analysis:*** Dependency analysis needs to be carried out to parallelize the application. Dependency analysis represents how different parts of the program are interdependent and how changes in one part may affect the other. Dependencies can be categories in two types: control dependency and data dependency**.** Control dependency is a situation where the execution of a part or statement in the program depends on the output of execution of another part or statement in the program. Whereas data dependency is a situation where an instruction is dependent on a previous instruction it can further be categorized into four types.

- Flow (true) data dependence: Data is written at a particular location and later read from that location.

- Anti Dependency: When data is first read at a particular location and then written to that location at a later time.
- Output Dependency: When data is first written at a location and then overwritten with a different value at a later time.
- Input Dependency: When data is read, modified and then again read at a later time.

2. ***Indentify Loop Parallelization:*** Loops are the regions where the program spends most of its computing and execution time. These loops are needed to be recognized by the automatic compiler. The complier recognizes the loops by performing a careful program analysis. Parallelization of the loops can be carried out if and only if the iterations of a particular loop are independent of each other.
3. ***Indentify Task Parallelization:*** A program is divided into different parts called tasks. Parallelization of these tasks can be done if they are independent of each other. Analysis of the program is done to find out the independent tasks in the program.
4. ***Indentify Vectorization:*** Vectorization is a process of converting an algorithm form its scalar implementation, which does an operation for one pair of operands at a time, to a vector process. In the vector process, a single instruction can refer to a vector. Vectorization adds a form of parallelism to a program where one instruction is applied to multiple pieces of data. Benefit of vectorization is more efficient processing and improved application performance.
5. ***Partitioning*:** The design of a parallel algorithm begins with discovering as much parallelism as possible. Partitioning is the process of dividing the data and computations into small pieces. A good partitioning divides both the data and computations into small pieces. To achieve this, either a computation-centric approach or data-centric approach is taken. Domain decomposition is the approach in which data is divided into pieces and then the determinism is done to associate computations with the data. Functional decomposition is a strategy in which the computations are divided first and then the determinism is done to associate data items with the individual components. Each of the pieces formed due to decomposition is called a primitive task.
6. ***Communication*:** After portioning the next step is to determine communication pattern between the primitive tasks. There are two types of communication patterns: local and global. When a task requires input from a small no. of other tasks, channels are created from the task supplying the data to the task by inputting the data. This explains local communication. Whereas, global communication takes place when a remarkable number of primitive tasks contribute to perform a computation.
7. ***Agglomeration:*** Agglomeration is the process in parallel programming that groups tasks into larger tasks in order to simplify programming or increase performance. It is done in situations where the number of tasks exceeds the number of available processors. The main goals of agglomeration are to lower communication overhead, maintain the scalability and reduce software engineering cost.
8. ***Mapping:*** The process of assigning tasks to processors is called mapping. The two main goals of mapping are to maximize processor utilization and minimize inter-process communication. Processor utilization is maximized by distributing the load evenly on each processor, allowing all processors to begin and end execution of tasks at the same time. Inter-process communication decreases when the two communicating tasks are mapped to the same processor. These two goals are conflicting and therefore finding an optimal solution to the mapping problem is difficult.

All these issues are well described in numerous books on the subject like Gregory V. Wilson[2], Quinn Michael J.[3], Richard Gerber et al[4]. Some other issues related with parallel conversion are Division of code and Synchronization, Processor Issues, Threading, Task Distribution, Context Switching are addressed by Barve and Joshi[5].

III. TECHNIQUES FOR PARALLELIZATION

1. ***OpenMP:*** Open multi processing is an API for writing a shared memory parallel application in C, C++ and FORTRAN. Open MP is easy to implement and it also allows incremental parallelism. Open MP uses a fork and join model of parallel execution. It uses compiler directives for parallelization. These directives are embedded in the source code. It is not used to check data dependency, data conflicts, deadlocks or race conditions. OpenMP is intended to support programs that will run correctly both as parallel and serial programs [6].
2. ***CUDA:*** CUDA is a platform for parallel computing invented by NVIDIA. By using the power of Graphics Processing Unit (GPU), CUDA enables increases in computing performance. With CUDA, no assembly language is required. It allows to run C, C++ or FORTRAN code directly on GPU. The sequential part of the GPU accelerated applications are run on the CPU and the parallel part on GPU. To achieve maximum performance, the parallel part of an application should be more as compared to its serial part[7].
3. ***MPI (Message Passing Interface):*** MPI enables the execution of parallel programs across a heterogeneous collection of parallel and serial computers. It is the most popular standard of message passing library for parallel programming. The message passing library enables different processes to communicate with each other. Writing parallel programs using MPI allows us to port the programs to different parallel computers. The message passing model considers that the underlying hardware is a collection of processors, each having its own local memory, and an interconnection network that support message passing between the processors [8].
4. ***OpenACC:*** The OpenACC toolkit invented by NVIDIA offers a simple way to accelerated scientific computing without any significant programming effort. Simply insert directives in C or FORTRAN code and the OpenACC compiler runs the code on the GPU. OpenACC is simple, powerful and free. With OpenACC, the existing code is kept intact and delivers faster performance when an accelerator is available in the system. The example below shows how OpenACC extends existing serial CPU code or parallel code using approaches like OpenMP [9].
5. ***OpenCL:*** OpenCL is the cross-platform standard for parallel computing. It is an open standard for parallel programming of heterogeneous systems. It improves speed and responsiveness for a wide spectrum of applications. These applications include gaming, entertainment, medical and scientific software. OpenCL consists of two APIs first C Platform Layer API, This API is to select, query and initialize computing devices. And another is C Runtime API: this API to build and execute kernels across multiple devices. Task and data parallelism both are supported by the OpenCL standard. It utilizes a subset of ISO C99 with extensions for parallelism. It defines a configuration profile for embedded and handheld devices [10].
6. ***Click Plus:*** Intel Click Plus is a reliable, quick and easy way to improve performance. It adds simple language extensions to the C and C++ languages to express data and task parallelism. Intel Click Plus includes the following features and benefits: Keywords are a simple and powerful expression of task parallelism, array notation, reducers, SIMD-Enabled Functions [11].

IV. REVIEW OF SERIAL TO PARALLEL CONVERTER TOOLS

1. ***YUCCA:*** YUCCA (User Code Conversion Application) is an automatic serial to parallel code conversion tool introduced by KPIT Technologies Ltd. Pune. It is a user code conversion application. The tool accepts input as a C source code file. The output generated by the tool is a transformed code. This code is a multithreaded parallel code which has pthread function and OpenMP construct. The tool does both task and loop level parallelization [12].
2. ***Par4All:*** This is an open source project developed by HPC project. It is a project that aims at simplifying parallel code generation from sequential source code written in C or FORTRAN language. It aims at generating parallel code with almost no manual help required. It is based on the PIPS compiler framework. Par4all is a script for users who are not inserted in parallel programming but wish to produce parallel code. This Par4all script takes serial C or FORTRAN source files and generates OpenMP, OpenCL, CUDA output that can run on shared memory multi-core processor or GPU[13].
3. ***Cetus:*** Cetus is a parallelizing compiler for C. Cetus can run on any system that support oracle, Java runtime environment, standard edition or later. It was originally developed by Prude University. It provides an internal C parser and is written in Java. Cetus uses the basic techniques used for parallelization and currently it implements reduction variable recognition, privatization, and induction variable substitution. The most recent version of Cetus includes a GUI and also a client server model. Cetus compiles and run the sequential input and generates an output which is the C code with OpenMP constructs. It also shows the charts of speedup and efficiency [14].
4. ***PLUTO:*** Pluto is parallelization tool that uses polyhedral model. It transform serial C program into parallel programs using available parallelism. It implements loop level parallelization. Pluto includes OpenMP construct into the automatically generated parallel code. The polyhedral transformation used in Pluto earlier lacked practicability and scalability. Pluto has improved this polyhedral transformation method and also solved the previous problems by introducing a fully automatic parallelization compiler [15].
5. ***Polaris compiler:*** Polaris is an automatic parallelization tool for FORTRAN programs. The objective of Polaris compiler is automatic code parallelization and to create and preserve the optimizing compiler. The compiler includes all optimization techniques required to transform a serial program into a parallel one that will run properly and efficient on the target machine. These techniques are automatic identification for parallelism and data distribution. It uses various loop dependency tests to generate efficient parallel code. It uses some standard tests like GCD test and equality test for linear cross iteration dependency. It also uses range test for non-linear cross iteration dependency. Polaris compiler also has some advanced capabilities that can perform the tasks like variable recognition, inter procedural analysis, array privatization, data dependence testing and symbolic program analysis [16].
6. ***Intel C++ Compiler:*** Intel C++ compiler increases performance by making efficient use of multi-core. It makes building long lasting applications easy. It is compatible with famous compilers and operating systems. Intel C++ compiler includes various features like Intel Click Plus, Auto-parallelization, OpenMP, Intel Many Integrated Core Architecture (MIC), Auto-vectorization, Inter-Procedural Optimization (IPO), Profile Guided Optimization (PGO), Processor Targeting Optimization, Intel Graphics Technology (GT), C++ Compliance and Pointer Checker. It automatically generates a multithreaded form of the applications

where almost all the computations are executed in simple loops. The Guided Automatic Parallelization (GAP) feature of Intel compiler provides advice evolving better performance of the serially written applications [17].

7. ***iPat/OMP:*** iPat is an interactive tool. It is a parallelizing tool for OpenMP. The tool helps the user in parallelizing serial codes into parallelized version using OpenMP. Emacs editor is required for the implementation of the tool. It uses Omni OpenMP compiler. iPat/OMP has four types of capabilities:
   1. Parallelism analysis: check the dependencies accordingly decide which section can be parallelized.
   2. Directive creation: Including the OpenMP directives for the section of code that can be parallelized.
   3. Program restructuring: Restructuring of program or sections in order to increase parallelism.
   4. Execution time analysis: Including calls that measure execution time.
   The details about this tools can be found in [18].
8. ***Vienna FORTRAN Compiler (VFC):*** VFC is a parallelizing compiler for HPF (High Performance FORTRAN) VFC translates FORTRAN 95 or HPF+ programs to FORTRAN 90 SPMD message passing (MPI) programs. VFC execute most characteristics of HPF+ and it also provides features that manage irregular programs. VFC consist of a front end that performs scanning and parsing of the input program and it also outputs an intermediate representation of the input program. The analysis module of VFC builds different abstractions of the program such as the flow graph, dependence graph and the call graph for every program unit. The parallelization module of VFC converts the input program to a parallel SPMD program [19].
9. ***SUIF Compiler:*** SUIF (Stanford University Intermediate Format) is a framework for research on optimizing and parallelizing compilers. SUIF has been developed as a platform on which research on compiler techniques is done for high-performance machines. It is flexible, modular, powerful and complete enough to compile benchmark programs. SUIF has been successfully used to perform research on various concepts including loop transformation, array data dependence analysis, software pre-fetching, scalar optimizations and instruction scheduling. The SUIF system consists of a parallelizer that automatically searches for parallel loops and generate a corresponding parallel code. To support parallelization the system supplies many features such as reduction variable recognition and data dependence analysis [20].
10. ***Omni OpenMP Compilers:*** Omni compiler consists of a collection of a libraries and programs which allows user to build parallel code. The compiler is used to translate FORTRAN and C programs with OpenMP directives into parallel code. This parallel code is then appropriate for compiling with a native compiler which is linked with the runtime library with Omni OpenMP [21].
11. ***TRACO:*** TRACO is a loop parallelization compiler. It is based on the iteration space slicing framework (ISSF). It is a source to source transformation of the C code. The output generated by Traco is a parallel code with OpenMP construct Traco is developed in python. It is a command line tool s[22].
12. ***CAPO:*** CAPO (Computer-Aided Parallelizer and Optimizer) is a tool that automatically inserts compiler directives to enable parallel processing on shared memory parallel machines. CAPO is developed at NASA Ames Research Center. The current version of CAPO takes in serial parallel FORTRAN codes or programs, performs inter-procedural data dependency analysis and generates a code with OpenMP directives. The success of CAPO depends on accuracy of interprocedural data dependency

information. CAPO generates the compiler directives in following three stages:

1) Identifying parallel loops
2) Constructing parallel regions around parallel loops and optimization of these parallel regions
3) Automatically identifying private, reduction, induction and shared variables to insert directives

Although CAPO generates directives automatically, user interaction is still important for producing parallel codes. A graphical user interface is therefore included in the CAPO tool so that a user can interact with the parallelization process [23].

V. COMPARISON OF SERIAL TO PARALLEL CONVERTER TOOLS

The tools presented in section III are compared on the basis of language support for parallelism, various techniques used in tools, support to platform and other various features. Table 1 gives comparison of various tools on the basis of language support for parallelism, various techniques used in tools, support to platform. And table 2 provides a comparison of the tools on the basis of their properties. The meanings of the columns in the table are as follows:

- F.I = Function Inlining
- A.P = Array Privatization
- R.V.R = Reduction Variable Recognition
- I.V.S = Induction Variable Substitution
- G.T = GCD Tests
- S.P = Scalar Privatization
- T.P = Task Parallelization
- L.P = Loop Parallelization
- P.M = Polyhedral Model
- V = Vectorization

Table 1: Comparison of tools based on language, techniques and platform

Sr. No	**Tools**	**Language Support**	**Techniques Used**	**Available Platform**
1	YUCCA	C	OpenMP, pthreads	Linux
2	Cetus	C	OpenMP	Supports all Linux environments
3	Par4All	C, FORTRAN	OpenMP, OpenCL, Cuda	For Debian and Ubuntu
4	Pluto	C	OpenMP	Linux/Unix
5	Polaris Compiler	FORTRAN77	OpenMP	Linux
6	Intel C++ Compiler	C++	OpenMP, Guided Auto Parallelism, Intrinsic, Click Plus	Windows, Linux (not for all variants), Mac OS, Android
7	SUIF	FORTRAN, C	SPMD	Linux
8	VFC	HPF+	SPMD	Linux
9	Traco	C, C++	OpenMP/OpenACC	Linux
10	iPat	C, FORTRAN	OpenMP	Windows
11	Omni	C, FORTRAN77	OpenACC	Ubuntu, Fedora, Mac OS

Table 2: Comparison of Tools Based on Available Features

Sr. No	**Tools**	**F.I**	**A.P**	**R.V.R**	**I.V.S**	**G.T**	**S.P**	**T.P**	**L.P**	**P.M**	**V**
1	YUCCA							✓	✓		
2	Cetus	✓	✓	✓	✓		✓		✓		
3	Par4All		✓	✓	✓				✓		
4	Pluto					✓			✓	✓	
5	Polaris Compiler					✓			✓		
6	Intel C++ compiler							✓	✓		✓
7	SUIF								✓		
8	VFC								✓		
9	Traco								✓		
10	iPat								✓		
11	Omni								✓		

VI. PERFORMANCE COMPARISON OF PLUTO AND PAR4ALL

The experiment for the testing of Pluto and Par4All was carried out on Ubuntu 14.04 LTS on HP Core i7 Machines with 8MB on chip cache and 4 GB RAM and processor speed 3.40 GHz having 8 cores in total. For testing, matrix multiplication program was used which generates random matrix of NxM and computes their multiplication. The performance of both the tools is shown in figure 1 and 2 respectively

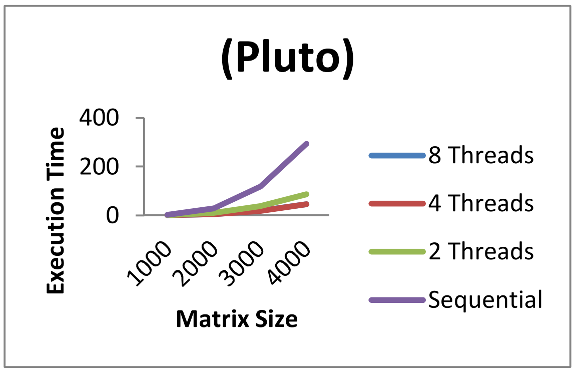

Figure 1. Time Taken by Pluto for parallel execution of various matrix multiplication using 2-8 CPUs

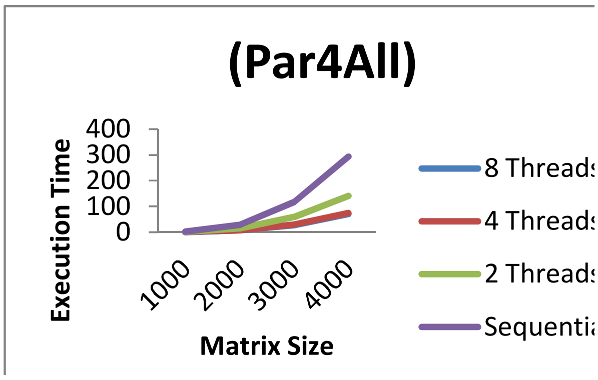

Figure 2. Time Taken by Par4All for parallel execution of various matrix multiplication using 2-8 CPUs

The performance of two tools Pluto and Par4All is compared. As per the performance graphs, it is clear that Pluto generated parallel code takes less time to execute as compare to parallel code generated by Par4All.

VII. CONCLUSION

The paper discusses the need of parallel programming and also stresses on the need of tools for parallel programming in recent era of computing. The paper also presents a brief introduction of the available techniques and tools for automatic parallelization along with implementation issues. A brief comparison of serial to parallel code converter tools has been presented on the basis of techniques, platform and features.

VIII. ACKNOWLEDGMENT

It gives us great pleasure in presenting the paper on “A Serial To Parallel C++ Source Code Converter”.

We would like to take this opportunity to thank my internal guide Dr. A. K. Barve for giving us all the help and guidance we needed. We are really grateful to them for their kin support .Their valuable suggestions were very helpful.

We are also grateful to Prof. Sachin Sakhare, Head of Computer Engineering Department, Vishwakarma Institute Of Information Technology for his indispensable support, suggestions.

In the end our special thanks to Dr. A. K. Barve for providing varies resources such as laboratory with all needed software platforms, continuous internet connection, for our project.

REFERENCES

[1]. Marry Hall, David Padua, Keshav Pingali; “*Compiler Research: The Next 50 years*”; Communication of the ACM, Vol. 52, No. 2, pp. 60-67, February 2009.

[2]. Gregory V. Wilson; “Practical Parallel Programming”; Prentice Hall of India, New Delhi 2002.

[3]. Michael J. Quinn; “*Paralle Programming in C with MPI and OpenMP*”; Tata McGraw-Hill Publication, New Delhi 2003.

[4]. Richard Gerber Aart J.C. Bik Kevin B. Smith Xinmin Tian; “The Software Optimization Cookbook High-Performance Recipes for IA-32 Platforms” Second Edition; Intel Press 2006.

[5]. Barve, Amit and Joshi, Brijendra Kumar; “*Issues in Implementation of Parallel Parsing on Multi-Core Machines*”; International Journal of Computer Science, Engineering and Information Technology; pp. 51-57, Vol 4, No. 5, October 2014.

[6]. www.**openmp**.org/

[7]. http://www.nvidia.com/object/cuda_home_new.html

[8]. http://www.mpi-forum.org/

[9]. https://developer.nvidia.com/openacc

[10]. https://www.khronos.org/opencl/

[11]. https://software.intel.com/en-us/intel-cilk-plus

[12]. Priti Ranadive & Pranjali Modak, “*Best To Be or Not To Be... A Driver*”, TechTalk@KPIT, Volume 7, Issue 1, 2014.

[13]. http://www.par4all.org/
[14]. http://cetus.ecn.purdue.edu/
[15]. http://www.ece.lsu.edu/jxr/pluto/
[16]. http://polaris.cs.uiuc.edu/polaris/polaris-old.html/
[17]. https://software.intel.com/en-us/intel-compilers
[18]. http://sourceforge.net/projects/ipat/
[19]. http://www.hindawi.com/journals/sp/1999/304639/abs/
[20]. http://suif.stanford.edu/suif/suif2/
[21]. www.hpcs.cs.tsukuba.ac.jp/omni-compiler/
[22]. http://issf.sourceforge.net/
[23]. http://people.nas.nasa.gov/~hjin/CAPO/