Automatic Parallelization Tools: A Review

Varsha. K. R
Department of CSE
RV College of Engineering, Bangalore, India

Abstract:
With the advent of multi core architectures, the concept of parallel programming is not restricted anymore to super-computers and enterprise level applications alone. Programmers are always ready to take advantage of the available potential of programs to be parallelized. The objective of the paper is to review various tools that are available for performing automatic parallelization.

Keywords: Multi-core, DOALL, Loop-level Parallelism, concurrency, data-dependence, profiling

I. INTRODUCTION
The rise of the concept of parallel or concurrent programming has led to the development of multi-core architectures on a wider range. Programmers find parallel programs executing on multiple cores or processors to be more efficient than serial ones. But it is a tedious task to write parallel programs directly or to convert all existing serial programs to parallel. This limitation can be eliminated by automating this procedure. Tools are developed which can automatically accept a serial code and generate a parallel code by inserting suitable parallel constructs.

A. PARALLEL COMPUTING ARCHITECTURES
According to Flynn’s classification, computer architectures can be classified into four classes. They are SISD (Single Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instruction Single Data) and MIMD (Multiple Instruction Multiple Data) [1]. SISD forms the normal execution procedure with single instruction executed on single data. SIMD are those which execute single or same instruction on multiple data instances. MISD is never noticed in real scenarios. MIMD executes multiple instructions on multiple data streams.

B. PARALLEL PROGRAMMING LANGUAGES
There are a wide variety of languages available to write parallel codes. OpenMP is the most widely used parallelization construct which is available in the form of APIs. OpenMP pragmas can be inserted in the available codes with little or no modification to the source codes. MPI is a framework which is used for message passing programs. For electronics applications, languages such as Verilog and VHDL are used for event-driven and hardware description applications. Elixir, Erlang are examples of languages for functional programming. For GPU applications, CUDA framework is used to write programs which are executed in parallel on GPUs. In recent times, parallel execution can also be seen in platforms such as Apache Hadoop and Apache Spark in the form of Map-Reduce and in-memory processing respectively.

II. OBJECTIVE
The objective of the paper is to review various tools that are available for automatic parallelization, the procedure employed by the tools, the source codes that the tool accepts, the parallel codes generated by the tool and other distinctive features.

III. EXISTING MODELS
The general procedure for these tools is to take a sequential code as input and generate parallel codes. Using the sequential code, a dependency matrix or graph is implemented which stores dependency information that exists between consecutive statements in a loop. The next step is to detect a code section which has potential parallelism. Finally the tools generate a parallel code by inserting parallel programming constructs/directives to potential DOALL loops. A loop can be considered DOALL if all the iterations can be executed concurrently without any conflicts. The loop cannot be DOALL if it has LCD (Loop carried dependence). Various tools are listed based on the differences in procedure employed to carry out each of these three phases.

A. SEQUENTIAL CODE PARSING
Input code is parsed to find required features in pre-defined form. There can be two kinds of tools based on the way a code is parsed to extract data from the code. The tool can be compiler based or profiling based. A compiler based tool can easily support FORTRAN. A profiling based tool is preferred for C/C++ due to pointer analysis limitation of compiler based tools. When it is not possible to keep track of flow of data owing to the usage of pointers, profiling technique analyses data dynamically in runtime.

B. DETECTION OF POTENTIAL CODE BLOCKS
Sections of code which are executed more frequently and carry no dependency among the sets executed one after the other are detected as blocks with a potential to be parallelized. While checking dependency among statements, the application has to check for RAW (read after write) or True dependency, WAW (write after write) or Output dependency and WAR (write after read) or Anti dependency.

Incremental Analysis-
In incremental method, parsing is done in incremental steps instead of parsing the entire file at once. This procedure is time saving because the parse structure is built and can be analyzed at the same time. This is equivalent to first parsing and then iterating over the file but here parsing and iterating is done in parallel so that potential blocks are detected while parsing. This
The tool was developed with a view to parallelism. It used level parallelism. It follows four steps (procedure, loops among others) because potential parallelism can be extracted even from regions which are executed less frequently. This is called Generality of the tool [25].

IV. EXISTING TOOLS AND TECHNIQUES

Alchemist: Alchemist is a novel profiling system for parallelizing C codes. During the dependency detection phase it provides facility for manual intervention to break dependencies but requires no assistance for the choice of code sections. It considers all programming constructs (procedure, loops among others) because potential parallelism can be extracted even from regions which are executed less frequently. This is called Generality of the tool [25].

Auto future: Auto future generates codes that are (a) Recognizable without much change from the original code (b) identifies set of recurring patterns with greater recognition values (c) correct or parallel codes giving similar outputs as serial codes. It provides parallelism to be used in various levels of granularity. Auto future is similar to Alchemist in that, the dependency graph generated also considers distance between statements to evaluate the effectiveness gained by parallelization [36].

Capo: This tool generates FORTRAN codes with OpenMP directives. It supports nested level parallelism. It follows four step procedure to code analysis which combines both dependency analyses and directives insertion. The first step starts off by analyzing the outermost loop and the second step involves identification in nested regions of the scope defined in the first level. However the insertion of directives is proceeded in the reverse direction with second level insertion being done to maintain a proper state of variables before first level insertion is done on the outer loops. It is based on CAP (Computer Aided Parallelization) Tools [29].

Cetus: Cetus is considered the most powerful and easy to use source-to-source compiler for C programs. It is a very fast tool for programs with lesser iteration size of for loops and those with minimal number of instructions in the body of the loop [4]. DProf: A profiling tool giving dynamic runtime information on the dependences that exist in various memory accesses in a loop and also on the percentage of occurrences of these dependences. Parallelization is done based on the information collected during the execution [26].

EasyPar: EasyPar - EASY development of PARallel codes is a unique tool which combines code development and code parallelization. The application includes an IDE which can be used for development. Simultaneously there is concurrency analysis taking place and giving parallelization suggestions. This tool eliminates most of the problems encountered during auto parallelization which require changes to be made to the source code. Here, the tool is interactive and hence the developer can witness parallel constructs and make changes to source code accordingly [24] [42].

GRES: GRES is short for Graphical Editor. It provides visual representation of complex sections of code in GRADE environments thus providing the user an insight into the program flow of parallel process execution [40].

ICU-PFC: It is a research compiler accepting FORTRAN source codes and generating parallel FORTRAN code with OpenMP directives [2].

Intel Compiler: Intel provides C++ and Fortran compilers supporting three features: OpenMP API, auto parallelization and auto vectorization. The first two features are based on thread level parallelism and vectorization on instruction level parallelism [6].

iPAT/OMP: It shows static analysis results and performance analysis data. It enables manual parallelizing and allows the user to execute parallelizing assistance functions in a commodity editor environment [14].

Kismet: The tool identifies the potential region that can be parallelized. Along with identification, the tool determines the time reduction or speedup that can be gained by parallelizing. This estimate on the upper bound is done for a spectrum of core counts [32].

Kremelin: The tool is interactive with the programmer. When given a sequential code, the tool gives suggestions in the form of a list of code sections which can be parallelized. It considers coverage, self parallelization and time reduction occurring the process and provides user or programmer with an opportunity to decide if the section has to be modified according to requirements [31].

Loop Sampler: A loop centric profile is used to detect parallelism in programs. The tool samples various loop executions and record the time spent in each of them. This is projected in a hierarchical view to provide ease of analysis [28].

OMPI OpenMP Compiler: The compiler can be used for C and Fortran 77 programs and generates codes with OpenMP constructs. The tool supports distributed shared memory systems by using cluster enabled OpenMP [16].

OSCAR compiler: The tool was developed with a view to improve performance, cost-effectiveness and productivity of software in SMPs. It accepts Fortran programs, checks for coarse grained parallelism using static scheduling. The output has parallel program in OpenMP API [9].

Par4All: This is an open source project aiming at providing a parallelization and optimization compiler. It accepts C and FORTRAN sequential programs and supports generation of openMP, CUDA and openCL parallel programs [12].

PDT-PLDT: This is more of an analysis tool used in Eclipse environment. It supports content analysis of MPI, OpenMP and
LAPI programs. The static analyses tool and detect deadlocks with barrier analyses and concurrency analysis of the programs [39].

PLUTO: The tool is based on polyhedral model which provides abstraction to carry out advanced transformations like loop-nest. It is used for generating parallel C codes. The transformations of the code occur simultaneously for data locality and parallelization [13].

Polaris: It is a FORTRAN 77 compiler that generates parallel FORTRAN codes. Unlike most other tools, Polaris generates parallel codes in multiple-passes additional to the regular passes that other tools perform. The tool performs advanced operational tasks such as array privatization, data dependence testing, variable recognition, inter-procedural analysis, and symbolic program analysis. As the conversion is done in multiple passes, iterating over the same code, it can be time consuming [3].

PortLand’s Compilers: This set of tools introduced by PortLand Group Inc (PGI) has compilers for FORTRAN, C and C++. These tools support OpenMP 3.1 for multi-cores and OpenACC for accelerators. The C++ tool also includes support for CUDA extensions. The PortLand’s set also includes OpenMP and MPI based parallel debuggers [22].

Prospector: This is a profile-based tool which uses dynamic dependency results [27].

S2P (Serial to Parallel): It is a static analyzer tool and supports both task and loop level parallelism. An additional feature used in the tool is called ‘run ahead mechanism’ where the ‘if’ statements and else statements corresponding to the ‘if’ are executed simultaneously thus improving performance of the tool [35].

SD3: The major problems occurring with profiling techniques are runtime and memory overheads. This scalable dynamic data-dependence profiling tool addresses both these issues. Runtime overhead is reduced by parallelizing the profiling step. Memory overhead is reduced by minimizing memory accesses using stride patterns [34].

SequenceL: A tool to parallelize c++ source codes with no explicit manual instruction from the user [19].

SGI Origin 2000 Compilers: The tool is used for Fortran source codes. It is designed to parallelize codes with simple structures and loops using MPI statements with minimal code modification [23].

SUIF (Stanford University Intermediate Format): The tool includes loop level parallelize and locality optimizer. The tool can be used for C and Fortran languages. It proceeds by maximizing the granularity and minimizing communications and synchronizations [5]. A limitation of the tool is that generation happens in multiple passes making the procedure time-consuming [7].

Timing Architects Compiler: It is simulation based approach to improve task allocation and also task parallelization. It is used for Embedded C programs [17].

TRACO: The tool is implemented in python for C/C++ applications. It uses the concepts of Iteration Space Slicing and Free Schedule Framework. Here, loop dependencies are represented using relations and Omega calculator for analysis [18].

Vector Fabrics: Gives insight information about the program which is crucial for writing highly optimized parallel code. The tool performs data-dependency analysis; convert it to parallel code for a particular architecture [37].

VISO: The occum based tool uses the concepts of incremental parsing, incremental dependency analysis, incremental profiling, incremental flow graph analysis [41].

YUCCA: YUCCA (User code parallelization application) is developed by KPI Technologies Ltd. [11]

V. CHALLENGES

A few tools require manual intervention to break dependencies and to choose parallelization parameters like Alchemist. Another important factor being the field of view of the tool, i.e. the constructs that the tool detects as potential blocks. Tools range from those detecting only one kind of loop (e.g.) to those which detect all programming constructs (e.g. Alchemist) exist. The complexity of tool depends on this range of detection.

VI. CONCLUSION:

Around 40 tools are reviewed and various methods employed by the tools are studied. It is observed that the procedure followed by most of the tools is as explained with flowchart. The most important phase is the detection of potential blocks which is also most time consuming. It can be observed that most tools are developed for FORTRAN language as it is easier to understand the operational flow and the objective of the code. It can also be noticed that the most widely used parallel construct is openMP because it requires very less or no modification in the existing serial code and also insertion of few statements are carried out. There are wide variety of tools in the market which can be used for auto parallelization. The tool has to be chosen wisely by considering the programming language of the code to be parallelized and the requirement of parallelization model.

VII. REFERENCES


[12]. Par4All: Auto-Parallelizing C and Fortran for the CUDA Architecture

[13]. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer UdayBondhugula, Albert Hartono J, Ramanujam, P. Sadayappan

[14]. Interactive Parallelizing Assistance Tool for OpenMP: iPat/OMP Makoto Ishihara (*1), Hiroki Honda (*1), ToshitsuguYuba(*1), Mitsuhashi Sato(*2) Fifth European Workshop on OpenMP EWOMP ’03 Aachen University, Germany September 22-26, 2003 *1: The Graduate School of Information Systems, University of Electro-Communications *2: Institute of Information Sciences and Electronics, University of Tsukuba.

[15]. VFC: The Vienna Fortran Compiler-Siegfried Benkner-Institute for Software Technology and Parallel Systems, University of Vienna, Liechtenstein Strasse 22, A- 1090 Vienna, Austria

[16]. Performance Evaluation of the Omni OpenMP Compiler Kazuhiro Kusano, Shigehisa Satoh and Mitsuhashi Sato RWCP Tsukuba Research Center, Real World Computing Partnership 1-6-1, Takezono, Tsukuba-shi, Ibaraki, 305-0032, JAPAN

[17]. Memory Architecture Exploration for Programmable Embedded Systems By Peter Grun, Nikil D. Dutt, Alexandru Nicolau

[18]. TRACO Parallelizing Compiler - Marek Palkowski

[19]. A PARALLEL COMPILER FOR SequenceL by PER ANDERSEN, B.E., M.S. A DISSERTATION IN COMPUTER SCIENCE Submitted to the Graduate Faculty of Texas Tech University in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY


[24]. A Review of Parallelization Tools and Introduction to Easypar SudhakarSah Symbiosis Institute of Research & Innovation (SIRI) Pune, India Vinay G. Vaidya Symbiosis Institute of Research & Innovation (SIRI) Pune, India


[33]. Prism: an analysis exploration and verification environment for software implementation and optimization on multicore architectures from CriticalBlue. http://www.criticalblue.com


[37]. vfAnalyst: Analyze your sequential C code to create an optimized parallel implementation from VectorFabrics, http://www.vectorfabrics.com/


