Automated C Code Polyhedral Parallelizer

Nida Patankar[1], Ayesha Parveen Khanbu[2], Harshala Bhoir[3], Amroz Siddiqui[4]

[1] *Student, Computer Department, Fr.CRIT-Vashi, Maharashtra, India*
[2] *Student, Computer Department, Fr.CRIT-Vashi, Maharashtra, India*
[3] *Student, Computer Department, Fr.CRIT-Vashi, Maharashtra, India*
[4] *Assistant Professor, Computer Department, Fr.CRIT-Vashi, Maharashtra, India*

Abstract - ***Optimizing compilers apply a number of interdependent optimizations, but it makes it a difficult task to decide which transformations to apply and in which order. Luckily, new infrastructures such as the polyhedral compilation framework present many kinds of transformations and facilitates efficient exploration and configuration of multiple transformation sequences. This paper aims towards providing a detailed description of the polyhedral model, its representations, tools developed so far using the polyhedral model and the idea of the system we are going to develop. The proposed system is an automatic parallelization tool based on the polyhedral model which will convert a legacy serial C code to its parallelized C code using OpenMPI. We present the idea of a completely automated parallelization tool, which transforms a program written in C language into its parallelized equivalent using OpenMPI, which is functionally equivalent version tailored for multicore architecture.***

Key Words: *C, Compiler Optimization, OpenMPI, Automatic Parallelization, Polyhedral model*

1. INTRODUCTION

Parallel processing has been a topic of study and research for a number of years. However, loop level parallelism has gained importance only in the past few years. Due to the ever growing trend of multi-core architecture, parallel programming is important as well as interesting. But, it is seen that parallel programming is a difficult chore requiring great efforts from the programmer. One conclusive elucidation to this problem is automatic parallelization.
Automatic parallelization is a mechanism of automatically converting a sequential program to a version that can directly run on multiple processing elements without changing the meaning of the program.. Automatic parallelization is typically performed in a compiler, at a high level where most of the information needed is available. Computing power can be used effectively if the programmers write only the sequential codes and leave the task of parallelization to the compiler. The output of an auto-parallelizer is a race-free deterministic program that obtains the same results as the original sequential program. This dissertation deals with compile-time automatic parallelization and primarily targets shared memory parallel architectures for which auto-parallelization is significantly easier. Currently well accepted methods of parallel programming, such as OpenMP or MPI, are essentially extensions to existing languages, like C or Fortran. On one hand, it allows reuse of an existing code base while on the other hand, it requires both the compiler and programmers to deal with languages that were not originally designed for parallelism. The major challenges involved in design and implementation of such a tool include side-effects of function calls, finding alias variables, dependency between statements, etc. Additionally the tool has to deal with the variety in coding styles, length and number of files. It is also important to take into consideration the amount of inherent parallelism the application provides.

2 POLYHEDRAL MODEL

The polyhedral model is a robust mathematical framework for automatic optimization and parallelization which is also known as the polytope model. Many scientific and engineering applications spend most of their execution time in nested loops. It is based on an algebraic representation of programs, allowing to construct and search for complex sequences of optimizations. It treats each loop iteration within nested loops as lattice points inside mathematical objects or a well defined space called the polyhedron as seen in Fig.1. Also, the most important concept in the polyhedron are the integer points, since loop iterators are integers and travel by an integral amount along the axes in the space. Depending merely on concepts of linear algebra and integer linear programming, it is possible to reason about the correctness of complex loop

transformations. It successfully performs affine transformations or more general non-affine transformations such as tiling on the polyhedron, and then converts the transformed polyhedron into equivalent, but optimized (depending on targeted optimization goal), loop nests through polyhedra scanning. The core transformation framework mainly works by finding affine transformations for efficient tiling.

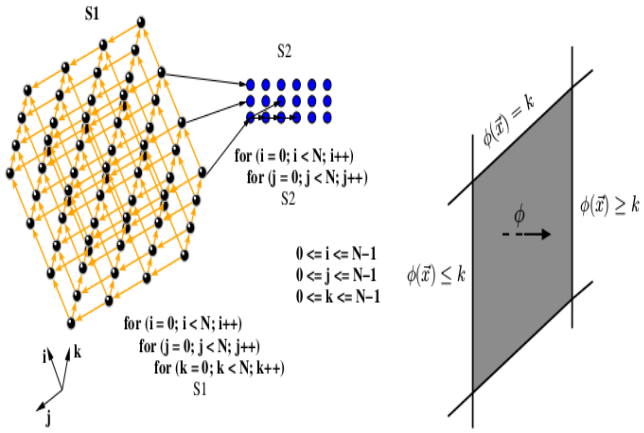

Fig -1: Hyperplane and Polyhedron[5]

The polyhedral model is readily applicable to loop nests in which the data access functions and loop bounds are affine functions (linear function with a constant) of the enclosing loop variables and parameters [4]. While a precise characterization of data dependencies is feasible for programs with static control structure and affine references and loop-bounds, codes with non-affine array access functions or code with dynamic control can also be handled, but primarily with conservative assumptions on some dependencies.

```
FOR I = 1 to n
  FOR J = 1 to n
    A[I, J] = A[I-1, J+1] + 1
```

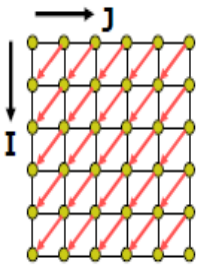

Fig -2: Representation of a simple code using Polyhedral Model

In simple words, polyhedral techniques are the symbolic counterpart, for structured loops (but without unrolling them), of compilation techniques (such as scheduling, lifetime analysis, register allocation) designed for acyclic control-flow graphs or unstructured loops [14]. Also, compared to optimizations that handle loops or arrays as a whole, polyhedral techniques can work at the granularity of their elements, i.e., at the granularity of a loop iteration and instance of a statement (operation), and at the granularity of an array element [14].

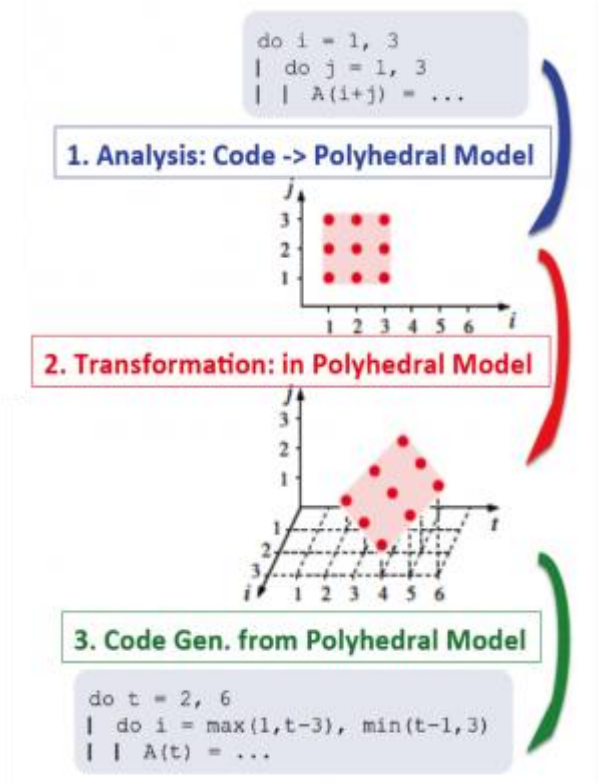

Fig -3: Stages of Polyhedral Transformation[10]

As it can be seen in Fig.3, the task of program optimization (often for parallelism and locality) in the polyhedral model may be viewed in terms of three phases:
1)Static dependence analysis of the input program.
2)Transformations in the polyhedral abstraction.
3)Generation of code for the transformed program.

1.2 Polyhedral Transformation and Dependencies

Two iterations are said to be dependent if they access the same memory location. Also, one of them should be a write operation [7]. There are different types of dependencies namely, Read-After-Write (RAW) dependencies, Write-After-Read (WAR) dependences that is,if a write operation is performed on a memory location followed by a read. Similarly, WAW and RAR are also two of them. Any reordering will only be legal if does not violate the dependencies, i.e. one is allowed to change the order in which operations are performed as long as the transformed program has the same execution order with respect to the dependent iterations. here.

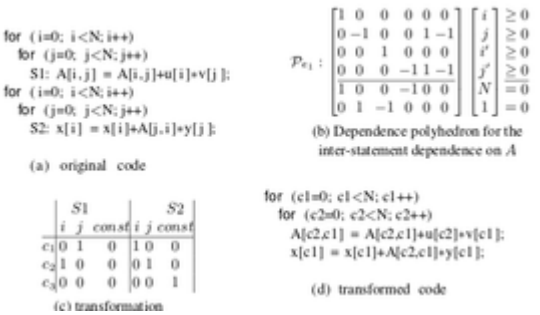

Fig -4: Polyhedral Transformation and dependencies [4]

3. EXISTING SYSTEMS

Various parallelization tools based on the Polyhedral Models were developed to deal with the problems mentioned above. Some of them are as given below:

3.1 PluTo

PLuTo[1][5][6] is a fully automatic polyhedral source-to-source program optimizer tool that takes C loop nests and generates tiled and parallelized code. It uses the polyhedral model to explicitly model tiling and to extract coarse grained parallelism and locality[6]. Since it is automatic, it follows a specific strategy in choosing transformations. Pluto's approach is closer to the latter class of partitioning-based approaches. However, it is the first to explicitly model tiling in the

transformation framework thereby enabling it to find good tiling hyperplanes for parallelism and locality. The view of tiling hyperplanes on the original domain is preserved till code generation. At the same time, input which cannot be tiled or only partially tiled is all handled, and standard transformations are captured[4]. In addition to model-based approaches, semiautomatic and search-based transformation frameworks in the polyhedral model also exist[1]. Though Pluto now is fully model-driven, some amount of empirical and iterative optimization may be required on complementary aspects, like tile size and unroll factor determination. Also, decision problems involved with fusion are good candidates for empirical search. Alternatively, more powerful cost models may be employed

once transformations in a smaller space are enumerated.

3.1 GRAPHITE

GRAPHITE [9][12] is an optimization framework for high-level optimizations that are being developed as part of the GNU Compiler Collection (GCC) now integrated to its trunk. Its emphasis is to extract polyhedral regions from programs that GCC encounters, a significantly more complex task than what research tools address[6], and to perform loop optimizations that are known to be beneficial. The design of GRAPHITE is largely borrowed from the WRaP-IT polyhedral interface to Open64 and its URUK loop nest optimizer . The CHiLL project from Chen et al. revisited the URUK approach focusing on source-to-source transformation scripting . Unlike URUK
and CHiLL, GRAPHITE aims at complete automation, possibly resorting to iterative search or statistical modeling of the profitability of program transformations. Besides, unexpected design and implementation issues have arisen[9], partly due to the design of GCC itself, but mostly due to the integration of the polyhedral representation in a general-purpose compilation flow, such as pointers, profile data, debugging information, resource usage (compilation time), pass ordering, interaction among passes, etc.

3.3 PIPS

PIPS[9] is a framework for source-to-source polyhedral optimization using interprocedural analysis. PIPS is one of the most complete loop restructuring compiler, implementing polyhedral analyses and transformations (including affine scheduling) and interprocedural analyses (array regions, alias).It uses a syntax tree extended with polyhedral annotations, but not a unified polyhedral representation. Its modular design supports prototyping of new ideas by developers. However, the end-goal is an automatic parallelizer, and little control over choices of transformations are exposed to the user.

3.4 MARS COMPILER

The MARS compiler[9] unifies classical dependence-based loop transformations with data storage optimizations. However, the MARS intermediate representation only captures part of the loop information (domains and access functions): it lacks the characterization of iteration orderings through multidimensional affine schedules.

3.5 PETIT TOOL

The first thorough application of the polyhedral representation was the Petit tool[9] , based on the Omega library . It provides space-time mappings for iteration reordering, and it shares our emphasis on per-statement transformations, but it is intended as a research tool for small kernels only.

3.5 ClooG

CLooG[11] stands for Chunky Loop Generator: it is a part of the Chunky project, a research tool for data locality improvement. It is designed to be also the back-end of automatic parallelizers like LooPo. Thus it is very 'compliable code oriented and provides powerful program transformation facilities. Mainly, it allows the user to specify very general schedules, e.g. where unimodularity or invertibility doesn't matter. CLooG is a free software and library to generate code for scanning Z-polyhedra. That is, it finds a code (e.g. in C,FORTRAN...) that reaches each integral point of one or more parameterized polyhedra. CLooG has been originally written to solve the code generation problem for optimizing compilers based on the polytope model. Nevertheless it is used now in various area e.g. to build control automata for high-level synthesis or to find the best polynomial approximation of a function. CLooG may help in any situation where scanning polyhedra matters. While the user has full control on generated code quality, CLooG is designed to avoid control overhead and to produce a very effective code.

3.6 LooPo

LooPo[11] is a project of the Chair for Programming at the Department of Informatics and Mathematics of the University of Nassau. Its purpose has been to develop prototype implementation of loop parallelization methods based on the polyhedral model. It is a prototypical source-to-source polyhedral compiler. The distribution contains the complete source code of LooPo,written in C++, with a front-end in Tcl/Tk and a new frontend in Java (in development). LooPo uses only freely available software to enable a wide-spread use. LooPo is known to run on Linux, Solaris and FreeBSD (and probably other *nix operating systems). LooPo is being provided as is and has been put under the GNU General Public License. It contains the CLooG code generator (byCedriccBastoul) and relies on externally installed PIPlib (required), PolyLib (required), Barvinok(optional) and Omega (optional).

3.7 Polyhedral Compiler Collections

Polyhedral Compiler Collections (PoCC) is another framework for source-to-source program optimizations, designed to combine multiple tools that utilize the polyhedral model[13]. POCC seeks to provide a framework for developing tools like Pluto, and other automatic parallelizers. However, their focus is oriented towards automatic optimization of C codes, and they do not explore memory (re)-allocation[12].

4. PROPOSED SYSTEM

The proposed system is an automatic parallelization tool based on the polyhedral model which will convert a legacy serial C code to its paralleled C code using OpenMPI. One approach would be to design the software in such a way that it uses the underlying hardware to its fullest. To reduce the manual analysis burden, time and effort, we present a completely automated parallelization tool, which converts a program written in C language into its paralleled equivalent using OpenMPI, which is functionally equivalent version which is tailored for multiform architecture. This tool would be able to convert serial code into a parallel code to execute on parallel multiprocessor architectures. It is based on the polyhedral representation. Given a program, each dynamic instance of a statement S, is defined by its iteration vector i which contains values for the indices of the loops surrounding S, from outermost to innermost[8]. Whenever the loop bounds are linear combinations of outer loop indicesand program parameters (typically, symbolic constants representing problem sizes), the set of iteration vectors belonging to a statement define a polytope[8]. Our system aims at providing precise performance models and profitability prediction heuristics. The implementation idea is taken from Pluto[8] which was developed to work with OpenMP ie., in a multi-threaded fashion. Its applications include automatic parallelization, factorization and static code analysis. We will implement our system to transform C code completely automatically. Fig.5 shows the entire tool-chain. We will use the scanner, parser and dependence tester from the LooPo infrastructure. LooPo is a polyhedral source-to-source transformation system that includes implementations of various polyhedral analyses and transformations from the literature. We will use PipLib as the ILP solver and CLooG 0.14.1 (with 64 bits) for code

generation[8]. Our tool will take as input dependence polyhedra from LooPo's dependence tester. Flow, anti and output dependences are considered for legality as well as the bounding function, while input dependences can optionally be considered for the bounding objective. We will also integrate the annotation-based transformation system of Norris et al. to perform some syntactic transformations on the code generated from CLooG as a post-processing; these include register tiling and unrolling and scalar replacement.

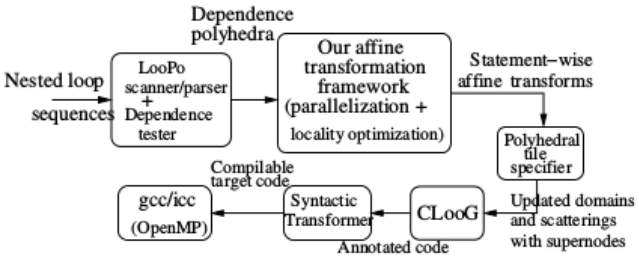

Fig -4: Source to source-to-source transformation system [8]

5. CONCLUSIONS

We have presented the idea of a fully automatic polyhedral source-to-source program optimizer that can optimize sequences of arbitrarily nested loops for parallelism. Through this work, we will be showing the practicality of automatic transformation using the polyhedral model in the multi-processor architecture. Performance wise, the automatically parallelized code generated using our system will almost be as optimal as the one which is manually developed. The transformation framework is planned to make it work with the C language which can be further optimized and extended to work with other languages as well. Such as for example, it could be applied to very high-level languages like MATLAB or domain-specific languages to generate high-performance parallel code.

REFERENCES

[1] Uday Bondhugula, Albert Hartono, J. Ramanujam, P. Sadayappan.A Practical Automatic Polyhedral Parallelizer and Locality Optimizer. U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelization and locality optimizer. In ACM SIG-PLAN Conf. on Programming Languages Design and Implementation (PLDI'08) , Tucson, AZ, USA, June 2008.

[2] Sven Verdoolaege, Tobias Grosser.Polyhedral Extraction Tool.

[3] Louis-Noel Pouchet, Cedric Bastoul , Albert Cohen, John Cavazos.Iterative Optimization in the Polyhedral Model:Part II, Multidimensional Time.

[4] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model.

[5] PLUTO,URL:http://www.ece.lsu.edu/jxr/pluto/index.html, Retrieved on:21 June, 2015.

[6] Tomofumi Yuki.Beyond shared memory loop parallelism in the polyhedral model.

[7] Uday Kumar Reddy Bondhugula.Effective automatic parallelization and locality optimization using the polyhedral model.

[8] Uday Bondhugula, Albert Hartono, J. Ramanujam, P. Sadayappan.PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System.

[9] Konrad Trifunovic, Albert Cohen, David Edelsohn, Li Feng, Tobias Grosser, Harsha Jagasia, Razya Ladelsky, Sebastian Pop, Jan Sjodi, Ramakrishna Upadrasta. GRAPHITE Two Years After First Lessons Learned From Real-World Polyhedral Compilation.

[10] D-TEC, URL:https://xstackwiki.modelado.org

[11] CLooG, URL:http://www.cloog.org

[12] Tomofumi Yuki, Gautam Gupta, DaeGon Kim, Tanveer Pathan, Sanjay Rajopadhye, AlphaZ: A System for Design Space Exploration in the Polyhedral Model*

[13] Pouchet, L.N., Bondhugula, U., Bastoul, C., Cohen, A., Ramanujam, J., Sadayap- pan, P.: Hybrid iterative and model-driven optimization in the polyhedral model.
[14] Polyhedral Compilation, URL:http://polyhedral.info
[15] A. Cohen, S. Girbal, D. Parello, M. Sigler, O. Temam, and N. Vasilache. Facilitating the search for compositions of program transformations. In ACM ICS , pages 151160, June 2005.