C CODE PARALLELIZATION WITH PARAGRAPHER

Abstract

Parallel programming has been attracting attention of programmers and researchers for many years. Parallelization is a very difficult task and may cause many runtime errors so some methods and tools facilitating this process are necessary. Parallel code can be prepared by a programmer or automatically by some compilers. As both approaches have advantages and disadvantages, it seems that a tool enabling manual and automatic parallelization can be very useful in the production of fast programs. Such a tool, called ParaGraph, was designed and implemented at the Institute of Computer Science, Warsaw University of Technology. ParaGraph is platform independent and plug-in in Eclipse IDE. In this paper an experiment comparing the effectiveness of manual and automatic parallelization of a program with ParaGraph is described.

1. INTRODUCTION

Parallel programming has been attracting attention of programmers and researchers for many years. At the beginning parallel code was simultaneously executed on processors of a supercomputer and currently many cores are present even in PCs so they can be used to run parallel code as well. Sequential program code can be parallelized and simultaneously executed on multiple cores. Parallelization is a very difficult task and may cause many runtime errors so some methods and tools facilitating this process are necessary. Parallel code can be prepared by a programmer or automatically by special compilers. As both approaches have advantages and disadvantages, it seems that a tool enabling manual and automatic parallelization can be very useful in the production of fast programs. Many tools have been built to support automatic or/and manual code parallelization. SUIF Explorer [1] combines static and dynamic compiler analysis with indications given by programmer. In compiler of Fortran 77 Polaris [2] directives introduced by a user are used in the process of translating code into parallel dialect of Fortran. ParaWise [3] is able to generate parallel code for distributed or for multiprocessor systems. HTGviz [4] enables manual and compiler parallelization of Fortran programs; the result is also a Fortran program with inserted OpenMP directives. Additionally, a visualization tool shows the static control flow graph of the program with data dependences.
At the Institute of Computer Science, Warsaw University of Technology, a tool called ParaGraph \([5,6]\), was designed and implemented. In ParaGraph, code in the C programming language can be manually, as well as automatically, parallelized and presented in a graphical form. To our best knowledge there are no similar tools dedicated to the C programming language. Usually programmers have difficulties writing code which can be highly parallelizable so code visualization may be very useful in teaching how to built easily parallelizable programs. ParaGraph is platform-independent and plug-in in Eclipse IDE.

The organization of this paper is as follows. The main approaches to parallelization are briefly presented in section 2. Section 2 contains also a very brief description of ParaGraph; more details can be found in \([5,6]\). In section 3 an example showing the speed-up of a program parallelized by our tool is described. The comparison of execution times of a program manually and automatically parallelized on two and sixteen processors is given. Section 4 contains some conclusions.

2. CODE PARALLELIZATION AND PARAGRAPH

Either code of a program can be parallelized automatically by a parallelizing compiler or a programmer can identify all parallelizable fragments and implement the parallel code. OpenMP \([7]\) is an environment in which a programmer is able to decide where to insert parallelizing directives indicating the parallel code. The compiler then uses platform specific mechanisms to make the program executable in parallel on multiple processors. A programmer, knowing the implemented algorithm, can precisely indicate the code suitable for parallelization and accelerate execution of the algorithm. On the other hand an inexperienced programmer can slow down execution of the program or make it unstable.

The parallelizing compiler works automatically, is not driven by a user, so parallelization is very limited. Compilers usually perform only static analysis and are able to efficiently parallelize loops in which there are no data dependences between loop iterations. Loops which are parallelized automatically usually contain only a few instructions, so the parallelization granularity is fine and the speed-up in execution time is not very convincing. Finding data dependence constraints is the basic step in detecting loop level parallelism in a program \([8]\). The data dependence test \([9]\), which determines whether two accesses to an array within a nest of loops may reference to the same element of that array, can be applied only to a pair of affine (linear) accesses. If a programmer implements a data structure to which the accesses in a loop cannot be affine, the compiler will not be able to compute the dependences and will have to assume that the conflict exists. Viitanen and Hamalainen in \([10]\) show that data dependence tests may fail to find all dependences in a program. Despite the limitations, automatic parallelization can speed up many programs but some effort is needed to be able to write programs suitable for the dependence test.

Giordano and Funari proposed in \([4]\) a tool called HTGviz which combines two methods of parallelization for Fortran programs. The sequential code is parallelized automatically by a compiler. The result of the compilation is also a Fortran program with inserted OpenMP directives. A visualization tool shows the static control flow graph of the program, with data dependences that were found by the compiler.

We proposed a tool, ParaGraph \([5,6]\), also taking advantages of both parallelization approaches and useful in studying how to prepare a highly parallelizable program. ParaGraph is dedicated to code in the C programming language and works as an Eclipse plug-in, precisely it is an additional element of the well-known plug-in CDT (C/C++
C code parallelization with Paragraph

Development Tool) [11]. The idea of ParaGraph operation is presented in Fig. 1. An external compiler is used to generate the parallelized code by injecting the OpenMP directives before suitable "for" loops. It also creates a file containing the control flow graph of the compiled program. Then the graph is shown in a graphical editor, with information concerning the source code. A direct link from blocks on the graph to the source code after parallelization helps a programmer to access the fragment of code which seems interesting to her/him. The feedback from the compiler is also given in a textual form. A user can modify the source code of the original project, then recompile it and observe whether the changes made the program more suitable for parallelization. The code generated by the parallelizing compiler can also be modified.

An example of code is shown in listing 1. A programmer wanted to accelerate the loop operations. Unfortunately, he/she did not notice, that the subscript in line 6 contains a data dependence which forbids parallelization with the #pragma omp for directive. After compilation, the programmer can see a graph in ParaGraph (Fig. 2) with information why this loop was not parallelized. Such information can help the programmer in realizing that he/she made a mistake and that the program will not produce a correct result. It can be seen that the loop cannot be parallelized in the way the programmer suggested. Additionally, ParaGraph parallelized the first loop, which was not indicated for parallelization by the programmer.

```c
int arr[10];                   //1
for (int i = 0; i < 10; i++)   //2
  arr[i] = i;                  //3
#pragma omp parallel for       //4
  for (int i = 1; i < 10; i++)  //5
    arr[i] = arr[i - 1];        //6
  printf(“arr[%d]=%d”,i,arr[i]);  //8
```

List. 1. Loop with data dependence [13]

ParaGraph is able to work with any compiler, which can generate code with OpenMP and a control-flow graph. Currently, it uses an open-source compiler (Fig. 1) named Cetus [12], performing source-to-source processing. Cetus currently supports ANSI C and is able to parallelize certain types of loops and insert "omp parallel for" directives. More details concerning the Paragraph design and operation can be found in [5,6].

3. EXPERIMENTS

ParaGraph, a tool for manual and automatic parallelization, was used in several experiments. The results of some experiments are presented in [5, 6]. In [5] OpenMP Micro Benchmark Suite, donlp2 solver for general continuous nonlinear programming problems and two specially written programs were examined. In [6] we describe how ParaGraph helps revealing errors in parallelization made by an inexperienced programmer. Below, a comparison of the effectiveness of manual and automatic parallelization is presented. Experiments were performed on a PC (Ubuntu Linux 9.04 with gcc 4.3.2 and GOMP
library for OpenMP), and on a machine with 16 processors (Windows Vista 32 and 4.3.3-tdm-1 5.1.4, a specific, build of MinGW, compiler supporting OpenMP).

Below the measurements obtained for an ANSI C program repeatedly performing various operations on matrices (matrix addition, scalar multiplication and addition, matrix multiplication) are presented. Approximately 50% of time is used in this program for matrix multiplication. We tested four versions of this program:

1. A sequential version,
2. A manually parallelized version – a programmer inserted some OpenMP directives,
3. An automatically parallelized version - the compiler injected some OpenMP directives,
4. A manually and automatically parallelized version, combining the two above methods. ParaGraph returned feedback information about the non-affine subscripts used in the loops to access the matrix data structure. After refactoring, which included changing the data structure to make it suitable for parallelization (i.e. the accesses to the matrix was made affine), the compiler successfully parallelized loops. The feedback from the compiler helped the programmer to prepare his/her code for parallelization.

The results of automatically parallelized programs are shown in Figures 3-5. Fig. 3 presents the speed-up of parallelization depending on the increasing number of matrix dimension for three versions of the same program. Fig. 4 illustrates the growth of acceleration when the number of processors is increasing. In Fig. 5 the speed-up on two and sixteen processors are shown. The bars in Fig. 3 show that automatic parallelization was successful only on large problems. The reason is that the overhead introduced by the thread creation and destruction is noticeable when a program is short-running. In matrix programs the threshold, at which parallelization accelerates the program enough to overcome, the overhead is around the dimension of 300. The situation when a parallelized program
C code parallelization with Paragraph

consumes more time than the sequential one, for small problems is frequent, especially when the parallelization grain is small. Automatic parallelization obtained in ParaGraph is of a very small grain, for matrix implementations a parallelized loop contained one or two instructions.

Fig. 3. Speed-up of parallelized program "matrix1" on 16 processors

In Fig. 4 it can be seen that although the increase of the number of processors is exponential the speed-up of parallelized programs does not grow up dramatically. The ratio of the number of parallelized instructions to the total number of instructions was only 3%. In well manually parallelized programs the ratio is usually much greater.

Fig. 4. Speed-up of parallelized program "matrix1" on growing number of processors

In Fig. 5 the comparison of the performance of matrix1 program parallelized using the combined method on two and sixteen processors is presented. It can be seen that parallelization accelerated the program even on two processors. It confirms that parallelization can be applied even to programs which are to be executed on personal computers with two cores only.

Fig. 5. Comparison of parallelization of "matrix1" program on 2 and 16 processors
4. CONCLUSIONS

In this paper the results of an experiment performed with ParaGraph, an Eclipse plug-in, supporting automatic and manual parallelization of C code are presented. The results, given in section 3, show that the best results can be obtained using both approaches to the code parallelization.

The visualization of the C code after parallelization, available in Paragraph, is very helpful in understanding OpenMP directives and preparing a program suitable for parallelization. An example described in [6] shows that our tool helps revealing some parallelization errors and improves the speed-up of parallelization. The results of experiments, presented in section 3, show, that parallelization can accelerate the execution of programs even on a PC with a double core processing unit. To our best knowledge there are no similar tools dedicated to the C programming language and available in the Eclipse environment.

REFERENCE


ZRÓWNOLEGLANIE C KODU ZA POMOCĄ PARAGRAPH

Streszczenie