Dariusz BURAK, Marcin RADZIEWICZ, Tomasz WIERCIŃSKI
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE, KATEDRA INŻYNIERII OPROGRAMOWANIA
ul. Żołnierska 52, 71-210 Szczecin

Automatic tuning framework for parallelized programs

Ph.D. Dariusz BURAK

Received the Ph.D. degree in Computer Science in 2007 from Szczecin University of Technology. He is an assistant professor of the Computer Science at the West Pomeranian University of Technology. His current research interests are focused on cryptography and compiler optimization.

e-mail: dburak@wi.zut.edu.pl

Ph.D. Marcin RADZIEWICZ

Received the Ph.D. degree in Computer Science in 2009 from West Pomeranian University of Technology. He is an assistant professor of the Computer Science at the West Pomeranian University of Technology. His current research interests are focused on programming optimization techniques for multicore procesors and GPUs.

e-mail: mradziewicz@wi.zut.edu.pl

Abstract

Complexity of computers has grown tremendously in recent years, because, among others, multi-processor and multi-core architectures are in widespread use. Parallelized programs should run on multi-core processors to use the most of its computing power. Exploiting parallel compilers for automatic parallelization and data locality optimization of sequential programs reduces costs of software. In this paper there is described the WIZUTIC Compiler Framework developed in the Faculty of Computer Science and Information Technology of the West Pomeranian University of Technology. The application uses the source code of the PLUTO parallel compiler developed in the Ohio State University by Uday Bondhugula. The simulated annealing method and the Bees algorithm are used for finding proper transformations of the source code for given program features. The experimental study results using the Data Encryption Standard (DES) algorithm are described and the speed-ups of encryption and decryption processes are presented.

Keywords: parallelized compiler, data locality optimization, iterative compilation, Data Encryption Standard, OpenMP.

Iteracyjny kompilator zrównoleglający oraz optymalizujący lokalność danych

Streszczenie

W artykule przedstawiono autorski kompilator zrównoleglający oraz optymalizujący lokalność danych- WIZUTIC oraz jego wykorzystanie do skrócenia czasu przetwarzania algorytmu szyfrowania DES. Do utworzenia kompilatora WIZUTIC transformującego kod źródłowy zapisany w języku C ze źródła do źródła wykorzystano kody źródłowe kompilatora PLUTO autorstwa Uday'a Bondhuguli służącego do optymalizacji lokalności danych z zastosowaniem transformacji tiling oraz zrównoleglenia pętli programowych z wykorzystaniem gruboziarnistej równoległości. W procesie kompilacji wykorzystano technikę kompilacji iteracyjnej oraz dwie metody optymalizacji: symulowane wyżarzanie (SA) oraz algorytm pszczół (BA) służące do określenia odpowiedniego rozmiaru bloku transformacji tiling. Przedstawiono wyniki badań eksperymentalnych dla algorytmu DES pracującego w trybie ECB. Badania przeprowadzona z zastosowaniem maszyny 8-procesorowej Quad Core Intel Xeon Processor Model E7310, kompilatora GCC GNU z wykorzystaniem standardu OpenMP w wersji 3.0 oraz narzędzia do profilowania kodu Intel VTune.

Ph.D. Tomasz WIERCIŃSKI

Received the Ph.D. degree in Computer Science in 2007 from Szczecin University of Technology. He is an assistant professor of the Computer Science at the West Pomeranian University of Technology. His current research interests are focused compiler optimization, hardware description languages, automatic hardware synthesis (VHDL i SystemC).

e-mail: twiercinski@wi.zut.edu.pl

Słowa kluczowe: kompilator zrównoleglający, optymalizacja lokalności danych, kompilacja iteracyjna , algorytm DES, OpenMP.

1. Introduction

One of the most important functional features of modern compilers is to ensure higher performance. Therefore, it is so important to enable the use of Shared Memory Parallel Computers by exploiting iterative compilation technique. In this way there can be found appropriate parallelization and data locality optimization transformations based on profile feedback information. In order to reduce a large number of executions of the target program there are used two well-known optimization algorithms: the simulated annealing method (SA) and the Bees algorithm (BA). The Data Encryption Standard (DES) algorithm is used to verify correctness and performance of our compiler.

The major purpose of this paper is to present the WIZUTIC Compiler Framework along with the description of automatic parallelization and data locality optimization of the DES algorithm. The paper is organized as follows. In Section 2, the PLUTO parallel compiler is briefly described. Section 3 contains detailed description of the WIZUTIC Compiler Framework. Section 4 shows optimization methods exploited in the WIZUTIC Compiler Framework. In Section 5, the Data Encryption Standard (DES) algorithm is briefly described. Section 6 shows experimental results regarding the optimized DES algorithm. Conclusion remarks are given in Section 7.

2. PLUTO compiler

The PLUTO compiler [1,2] is an automatic parallelization and data locality optimization tool that is based on the polyhedral model. The polyhedral model for compiler optimization is a high-level representation for programs that utilize machinery from Linear Algebra and Linear Programming for analysis and transformations (such as statement domains, dependences, array access functions- and affine program transformations). Pluto transforms C programs from source to source for coarse-grained parallelism and locality simultaneously. The core transformation framework mainly works by finding affine transformations for efficient tiling and fusion. The key elements of the PLUTO compiler are as follows:

- scanner, parser and dependence tester (from the LooPo infrastructure [3]);
- Affine Transformation Framework (for parallelization and locality optimization transformations);
- CLooG [4] for code generation;
- GNU GCC / ICC compiler [5, 6].

The PLUTO compiler has same disadvantages. One of the most important disadvantage of using the PLUTO compiler is acceptance only a very small subset of C language.

3. WIZUTIC Compiler Framework

Fig. 1 shows the WIZUTIC Compiler Framework for automatically tuned programs.

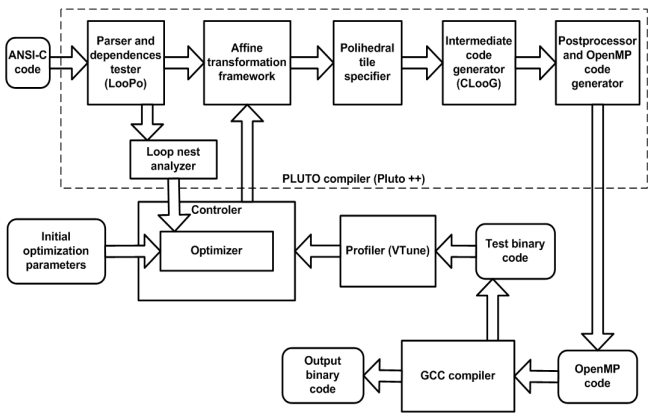

Fig. 1. Overview of the WIZUTIC Compiler Framework
Rys. 1. Schemat budowy kompilatora WIZUTIC

The WIZUTIC Compiler Framework

The most important module of the WIZUTIC Compiler Framework is a Controller, which calls other modules for each iteration of the compilation process. The Controller contains an additional module - Optimizer. The Optimizer reads initial parameters of the compilation process (and also identifies their allowed limits) and then executes the following iterations of compilation using well-known optimizing methods: the simulated annealing technique (SA) and the Bees algorithm (BA) (parameters of compilation process are selected from their allowed limits). The following modules are running for each iteration of the compilation process:

- PLUTO compiler,
- GNU GCC,
- Intel VTune profiler.

The PLUTO executes program in accordance with the OpenMP standard [7] based on values of parameters of compilation process. As the PLUTO data output there is received an output file containing the transformed source code (with accordance with the OpenMP 3.0 standard). In the next step this file is executed using GNU GCC to binary code for actual hardware/software architecture. The executed program metrics are read using the Intel VTune profiler [8]. This measurement concerns executing time of our program (it depends on both parallelization and data locality level). In the next step the metrics of program execution are analyzed using the Optimizer. The iterative compilation process ends when the speed-up values are satisfactory for a user.

PLUTO++ compiler
(Extensions to the PLUTO compiler)

To integrate the PLUTO compiler and WIZUTIC Compiler Framework we have to introduce some changes in order to:

a) know how many program loops are included in the source code (this information is indispensable for the Optimizer to generate the proper block size for tiling transformation);
b) know how many nested loops are included in each outer loop (this information is also indispensable for the Optimizer to generate the proper block size for tiling transformation);
c) know what are the initial and final values of iteration variables for each program loops (both outer and nested) (this information is also indispensable for the Optimizer to generate proper block size for tiling transformation) (in the case of parameterized loops, we use default settings- 32);
d) execute the source code ignoring parsing and data dependency analyzing for each iteration of compilation (except on the first one) in order to reduce the compilation process time (we omit some parts of compilation process- the same for each iteration of compilation process) (this optimization is optional (and can be switch on using special parameter)).

PLUTO++ have the following modes of operation:

a) normal- modifications are switched off;
b) informational- information about program loops are generated (and writing to an output file);
c) iterative- PLUTO++ compiler sends information to the WIZUTIC Framework Compiler and waits for further instructions after applying affine transformations; after receiving this information the PLUTO++ continues work in accordance with the used parameters for tiling transformation; next, it ones more sends information to the WIZUTIC Framework Compiler and passes on the waiting mode; in this way we can ignore the source code analyzing phase, data dependences generation phase and affine transformations applying phase; in this way we reduce compilation process time (communication between PLUTO++ compiler and WIZUTIC Compiler Framework is implemented using the message queues);
d) front-end- only main executing file is executed;
e) back-end- only final scripts are executed (vectorization, loop unrolling and generation of parallel programs in accordance with OpenMP standard); this mode of operation is used by the WIZUTIC Compiler Framework; after receiving information about new output files generation (from the PLUTO++ compiler) it is necessary to end compilation process by executing PLUTO++ back-end.

4. Optimization methods

In order to find the correct and effective version of consecutive automatic transformations (a transformation chain) there were used two well-known optimization methods: the simulated annealing (SA) and the Bees algorithm (BA).

Simulated annealing

Simulated Annealing is a technique proposed by Kirkpatrick, Gelatt and Vecchi in 1983 for finding the global minimum of a cost function that may possess several local minimums. It is an optimization procedure based on the physical process of annealing. Annealing is a process in which organized crystals are formed. During this process a physical substance is melted by raising to very high temperature, then cooled down slowly so that a long time is spent at each temperature drop, which allows the molecules of substance to reach the equilibrium state. If this is not done, and substance is allowed to get out of equilibrium, the resulting crystal will have many defects, or the substance may form a glass, with no crystalline order and only metastable, locally optimal structure will be formed. The more detailed description of simulated annealing (SA) technique is presented in [9].

Bees algorithm

The Bees Algorithm (BA) is a novel populace-based search algorithm. The algorithm imitates the food foraging behaviour of honeybees' colony. In its basic version, the algorithm executes a kind of vicinity search combined with random search and can be used for optimization. The more detailed description of the Bees algorithm (BA) is presented in [10].

5. The Data Encryption Standard (DES) algorithm

The Data Encryption Standard (DES), developed by IBM in the 1970s by the National Bureau of Standards with the help of the National Security Agency, adopted by the U.S. government in 1977, officially described in FIPS PUB 46 is still a widely used algorithm for data encryption for many applications. The DES algorithm uses a 56-bit secret key to turn 64 bit blocks of a plaintext (input) into 64 bit output blocks of the ciphertext. The DES encrypts a block of data by first passing it through the initial permutation (IP), then iterating the round function 16 times (each with a different sub-key), and finally passing it through the inverse of the initial permutation (IP-1) to create one output data block according to the Feistel's scheme. For more information concerning the DES transform, please refer to the DES specification (FIPS PUB 46-2)[11].

6. Experimental results

In order to parallelize the Data Encryption Standard (DES) algorithm in the ECB mode, there was used a sequential algorithm written in ANSI C language presented in [12].

In order to study the efficiency of the parallel code, the computer with the following features was used:
- 8 x Quad Core Intel Xeon Processor Model E7310, Speed 1.6GHz, L2 Cache memory: 2 x 2 MB, RAM memory: 32 GB,
- the openSuse 11.1 operating system,
- the GNU GCC ver.4.3.2 with compilation option -O3 (that supports the OpenMP ver.3.0).

The results received for the plaintext of the size about 20 megabytes are shown in Table 1.

The total running time of the Data Encryption Standard algorithm consists of the following time-consuming operations:
- data reading from an input file,
- data encryption,
- data decryption,
- data writing to an output file (both encrypted and decrypted text).

We optimized two the most time-consuming functions: the des_enc() and the des_dec() using the WIZUTIC Compiler Framework with the following PLUTO compiler options: --tile –noprevector.

Tab. 1. Experimental result studies of the DES algorithm in the ECB mode
Tab. 1. Wyniki badań eksperymentalnych z zastosowaniem algorytmu DES w trybie pracy ECB

Encryption Algorithm	Execution time of the des_enc() function [s]	Execution time of the des_dec() function [s]	Speed-up of the des_enc() function	Speed-up of the des_dec() function	Data size [MB]
Sequential DES	6,04	6,02	1	1	20
Optimized DES (PLUTO with default options)	1,21	1,2	4,99	5,02	20
Optimized DES (WIZUTIC with SA)	1,01	1	5,98	6,02	20
Optimized DES (WIZUTIC with BA)	0,9	0,89	6,71	6,76	20

We transformed outer 'for' loops included in these functions using tiling transformation. In order to select suitable block sizes, we applied two optimizing algorithms: the Simulated Annealing and the Bees Algorithm.

The following experimental results (Table 1) were obtained using:
- PLUTO compiler with default options,
- WIZUTIC Compiler Framework with the Simulated Annealing (SA) (initial temperature (t_0)- 200, temperature reduction function (α)- 0.9, number of iterations (i)- 25),
- WIZUTIC Compiler Framework with the Bees Algorithm (BA) (number of scout bees (n)- 20, number of scout bees (m)- 5, number of best sites out of m selected sites (e)- 2, number of bees recruited for best e sites (nep)- 2, number of bees recruited for the other(m-e) selected sites (nsp)- 4, initial size of patches (ngh)- 5, number of algorithm steps repetitions ($imax$)- 10).

7. Conclusion

The WIZUTIC Compiler Framework was applied to data locality optimization of the Data Encryption Standard algorithm using tiling transformation. We measured the executing-time of encryption and decryption processes, both for sequential and optimized with use of the WIZUTIC Compiler Framework versions (using both the Simulated Annealing and the Bees Algorithm). We obtained the satisfactory speed ups (presented in Table 1). The experimental results confirm legitimacy of iterative compilation for optimization of the Data Encryption Standard algorithm.

8. References

[1] PLUTO - An automatic parallelizer and locality optimizer for multicores. http://pluto-compiler.sourceforge.net.
[2] Bondhugula U., Sadayappan P.: Effective automatic parallelization and locality optimization using the polyhedral model, Ohio State University, Columbus, OH, 2008.
[3] The Polyhedral Loop Parallelizer: LooPo. http://www.infosun.fim.uni-passau.de/cl/loopo.
[4] The CLooG Code Generator in the Polyhedral Model's. http://www.cloog.org.
[5] GCC, the GNU Compiler Collection. http://gcc.gnu.org.
[6] Intel® Compilers. http://software.intel.com/en-us/intel-compilers.
[7] The OpenMP API specification for parallel programming. http://openmp.org/wp.
[8] Intel® Software Network.http://software.intel.com/en-us/intel-vtune.
[9] Kirkpatrick S., Gelatt C. D., Jr. and Vecchi M. P.: Optimization by Simulated Annealing, Science, vol. 220, pp. 671-680, May 1983.
[10] Pham D.T., Ghanbarzadeh A., Koc E., Otri S., Rahim S. and Zaidi M.: The Bees Algorithm, A Novel Tool for Complex Optimisation Problems. Proc 2nd Int Virtual Conf on Intelligent Production Machines and Systems (IPROMS 2006). Oxford: Elsevier, pp. 454-459.
[11] NIST Federal Information Processing Standards (FIPS) PUB 46-2 Data Encryption Standard, Dec. 1993.(http://www.itl.nist.gov/fipspubs/fip46-2.htm)
[12] Schneier B.: Applied Cryptography. 2nd Edition, John Wiley and Sons, New York, 1996

otrzymano / received: 21.09.2010
przyjęto do druku / accepted: 01.11.2010 *artykuł recenzowany*