Agnieszka KAMIŃSKA, Włodzimierz BIELECKI

West Pomeranian University of Technology, Szczecin

Model for the estimation of the execution time of parallel program loops

Abstract. *This paper presents the assumptions and proposition of the model for the estimation of the execution time of parallel program loops. The model is intended to be used in iterative compilation. The model has been elaborated with the focus on shortening the duration of iterative compilation.*

Streszczenie. *W artykule przedstawiono założenia i propozycję modelu do oszacowania czasu wykonania zrównoleglonych pętli programowych. Przewidywanym obszarem zastosowania modelu jest kompilacja iteracyjna. Model został opracowany pod kątem skrócenia czasu kompilacji iteracyjnej. (**Model do oszacowania czasu wykonania zrównoleglonych pętli programowych**).*

Keywords: iterative compilation; program execution time; performance estimation.
Słowa kluczowe: Kompilacja iteracyjna, czas wykonania programu, szacowanie wydajności.

doi:10.12915/pe.2014.02.33

Introduction

Despite continuous development and improvement of computer hardware, laws of physics impose limits on the possible increase of the clock rate of a microprocessor. Since the duration of data processing by a computer system is crucial in many practical applications of such systems, use of multiprocessor computers supporting parallel computing has become an alternative for increasing the clock rate of a microprocessor.

Parallel computing is based on the principle that a large problem can often be divided into smaller problems, which are then solved concurrently ("in parallel") and in a shorter time than it would take to solve an original large problem.

Parallel applications used in parallel computing can be created either manually or automatically. In case of the manual creation, a developer either writes a parallel application from scratch or parallelizes an already existing, sequential application. In case of the automatic creation, dedicated parallelizing compilers automatically convert sequential applications into parallel equivalents.

Because of active involvement of a human being in the manual creation of parallel applications, it is time consuming and error prone. For these reasons, the automatic creation of parallel applications is much more advantageous.

Currently, there are no theoretical means enabling one to state, at the compilation stage, how a sequence of semantically allowed transformations used for converting a sequential source code into its parallel equivalent will translate into the actual execution time of the parallel application in the target hardware environment. Therefore, in case of some practical problems and applications (e.g. embedded systems, scientific calculations, etc.) requiring high efficiency, one carries out so-called iterative compilation.

In this approach, the initial (and in this case, sequential) source code is transformed in successive steps (called iterations) into syntactically different yet semantically identical (and in this case, parallel) equivalents. Each equivalent version of the initial source code is executed in a target hardware environment; the equivalent version having the shortest measured execution time is selected for final use [1].

Iterative compilation is effective but time consuming. A potential area of improvement in iterative compilation is to use estimates in order to narrow the group of semantically allowed transformations for a given source code to transformations producing equivalent source codes of possibly shortest execution time. When estimates are used in such a way for selection purposes, it is sufficient to examine in the target hardware environment only the behaviour of the so selected, equivalent source codes. In consequence, the total duration of iterative compilation gets shortened, with no deterioration of its results [1].

Typically used approaches to estimation of program execution time

There are two alternative approaches that can be used for the estimation of program execution time: profiling (also known as "program profiling", "software profiling") and performance models.

Within profiling, detailed information (regarding e.g. usage of particular instructions, frequency and duration of function calls) on the program behaviour is collected during execution of the program and then used for the analysis of the program behaviour and for program optimization. Profiling is time consuming and as such, when applied in iterative compilation, it does not provide satisfactory shortening of the duration of iterative compilation. Therefore, from the practical point of view, the only reasonable alternative for profiling is to use performance models oriented on estimation of program execution time.

Model is a simplified view of some aspect of reality. Model should reflect, as thoroughly as possible, the issues which are relevant for the modelled perspective, at the same time partially or completely ignoring the issues which are not relevant therefor.

A good model should explain the known observations regarding the object/phenomenon under consideration and give one the possibility of making predictions regarding the object/phenomenon under consideration.

Typically used techniques of modelling performance, based on program execution time, are: Amdahl's law, extrapolation from observation and asymptotic analysis [2, 3].

According to Amdahl’s law, the maximum possible speedup to be gained from parallel execution of a program is limited by the sequential part/component of the program, i.e. this part/component of the program, which always has to be executed sequentially. Amdahl’s law provides no explicit means for calculating the fraction of operations in a computation that have to be performed sequentially [2].

Within extrapolation from observation, the performance of the application is measured in several points (e.g. for several various problem sizes and a given number of processors executing the program) and the results of such

measurements are used for predicting the behaviour of the analysed application in situations different than the ones being the base for extrapolation. Extrapolation from observation takes no account of the algorithm executed within the program, which may result in a significant estimation error for the estimations made in such a way [2].

With asymptotic analysis, it is possible to state how the program execution time changes depending on the problem size, when the said problem size and the number of processors used for execution of the program are large or very large. However, this technique provides no information about the program execution time for smaller problem sizes and a different number of processors [2].

The above description shows that the discussed techniques of modelling performance, based on program execution time, are inadequate for the proposed improvement of iterative compilation i.e. using estimates in order to narrow the group of semantically allowed transformations for a given source code to the transformations producing equivalent source codes of possibly shortest execution time in a given hardware environment. Therefore, the proposed improvement requires and involves elaboration of a completely new performance model, dedicated entirely for this improvement.

Description of the proposed model

In order to elaborate the model for estimation of execution time of parallel applications, adequate to be used in iterative compilation as the compilation stage selector of source code versions/transformations of possibly shortest execution time, one has to in the first place formulate the assumptions and limitations of the model. Only when these assumptions and limitations are met, the estimates made with the elaborated model are of satisfactory accuracy.

The assumptions and limitations in question regard the following aspects:

- Scope of applicability of the model,
- Dependent and independent variables of the model,
- Type and form of the model.

Defining the scope of applicability of the model is the key and first problem to solve, as it is not possible to formulate a universal and at the same time, sufficiently precise, model adequate for all possible cases.

Since the majority of time consuming calculations in the program is usually executed in program loops, therefore in parallel applications the focus is put on parallelization of program loops. An important issue to take into account in parallelization of program loops is their granularity, i.e. the number of operations done between communication or synchronization events. There are two types of granularity: coarse-grained granularity and fine-grained granularity. Coarse-grained granularity takes place when the number of operations done between communication or synchronization events is large; this type of granularity corresponds with parallelization of a nested loop done at the level of its outermost loop. Fine-grained granularity takes place when the number of operations done between communication or synchronization events is small; this type of granularity corresponds with parallelization of a nested loop done at the level of one of its inner loops [2, 3].

The scope of applicability of the model proposed herein is coarse-grained parallel program loops, parallelized in the OpenMP standard. The selection of OpenMP for parallelization purposes results from the popularity of this standard.

As the model is intended for estimation of program execution time, the dependent variable of the model should reflect this time. It is proposed herein to express this time by total CPU time (i.e. the sum of CPU time consumed by all of the CPUs utilized by the program) since it reflects the time spent solely on processing instructions of the program.

Independent variables of the model should reflect crucial factors having impact on program execution time. In view of the assumptions made so far, there are the three factors to be reflected in independent variables:

- Parameters of the target hardware environment in which the program loop will be executed,
- An algorithm executed in the program loop,
- A specific way in which the program loop has been parallelized.

Taking into account the intended use of the model, for each of the factors listed above there has to be a reasonable trade-off between the precision of reflecting the factor in independent variables of the model and the complexity of mathematical apparatus used for quantifying this reflection.

As far as the target hardware environment is considered, it is essential to remember that nowadays, there is a large disproportion between processor speed and memory access time [4]. Because of this disproportion, it is the memory – and especially, the quickly accessible processor cache memory – that is one of these components of the hardware environment, which determine the duration of program execution.

Ideally, all the data needed by the processor during execution of a program should be available in the processor cache memory at the moment they are requested, instead of being just then fetched from the main memory to the processor cache memory. The more data to be processed in the program are available in the processor cache memory at the moment they are requested, the shorter the duration of program execution.

On the other hand, the data storage capacity of the processor cache memory and its replacement policy (associativity) determine what fraction of the data processed in the program will be available in the processor cache memory right at the moment they are requested.

This means that the duration of program execution depends on:

1/ the actual data storage capacity of the processor cache memory in a given computer system and its replacement policy (associativity),

2/ the minimum data storage capacity of direct-mapped cache, which is necessary in order to contain all the data processed in the program, assuming the full temporal and spatial reuse of the data stored in the cache memory. The minimum data storage capacity in question can be estimated by means of data footprint. In order to calculate the data footprint for a given program, it is sufficient to know the source code of a program; there is no need to execute this program [5].

3/ the relation between 1/ and 2/.

In connection with the above, one has introduced into a model variable *X1* (calculated as per (2)) which expresses the relation between 1/ and 2/.

Regarding the algorithm executed in the program loop, one should reflect in the model the fact that, at the most low level, various algorithms differ most significantly in the type and number of arithmetic operations to be executed. A simple yet effective way of quantifying the above observation is to assign different weighting factors to different arithmetic operations, based on the analysis of instruction timings for a given processor. With this approach, arithmetic operations of various types (e.g. addition and multiplication) are comparable with one another. For this reason, one has introduced into the model

variable *X2* expressing the total weighted number of arithmetic operations per single program thread.

As far as the way of parallelization of a program loop is considered, a model should reflect the parallelization approach adopted in the OpenMP standard, i.e. multithreading [6].

In connection with the above, one has introduced into the model variables expressing the number of threads executing the program (variable *X4*) and the maximum number of iterations in an iteration chunk per thread (variable *X3*).

Taking into account large variety of issues to be reflected in a model as well as complex, significant and highly unformalizable relations between these issues and the hardware environment, the easiest way of combining them in a formula producing satisfactorily precise estimates is to apply a statistical model built on a power function.

In statistical models, the values of model parameters are estimated by means of regression analysis carried out on the observations collected for the adopted sample. With this approach, it is possible to achieve high goodness of fit of a statistical model and keep it highly responsive to even very minor changes in the values of its independent variables.

Moreover, the derivative of a power function is much more responsive to even very minor changes in the values of its arguments than the derivative of a linear function.

For these reasons, while constructing the model to be used for the discussed herein improvement of iterative compilation, a power function statistical model with parameters a1, a2, a3, a4 has been applied.

Ultimately, the model, meeting all the assumptions and limitations specified above, is as follows:

$$Yt = X1^{a1} \times X2^{a2} \times X3^{a3} \times X4^{a4} \tag{1}$$

where: *Yt* – total CPU time, *X2* –total weighted number of arithmetic operations per OpenMP thread, *X3* –maximum number of iterations in an iteration chunk per OpenMP thread, *X4* – number of OpenMP threads executing the program, *a1, a2, a3, a4* – parameters the values of which are estimated by means of regression analysis carried out on the empirical data collected in the target hardware environment for a specially prepared sample.

$$X1 = \frac{\left(sL1 \times aL1 + sL2 \times aL2\right)}{D_f} \tag{2}$$

where: *sL1* – data storage capacity of the L1 data cache, available for a single OpenMP thread [B], *sL2* – data storage capacity of the L2 cache, available for a single OpenMP thread [B], *aL1* – set associativity of the L1 data cache, *aL2* – set associativity of the L2 cache, *Df* – data footprint per OpenMP thread, calculated as per [5] [B].

It should be emphasized here that variables *X1, X2* and *X3* are related to one another. The size of the problem solved within the program is the binding element here.

Verification of the model and conclusions

Model (1) has been verified for 10 program loops (CG_cg_3, CG_cg_4, FT_auxfnct_2, LU_HP_pintgr_11, MG_mg_3, UA_diffuse_2, UA_diffuse_3, UA_diffuse_4, UA_transfer_11, UA_transfer_16) selected from the NAS suite that has been elaborated by the NASA Advanced Supercomputing (NAS) Division as software means for evaluating the performance of parallel supercomputers [7].

For each tested program loop, several different but semantically equivalent, parallel versions of a source code were generated. For each version, the time of its execution on a multi core processor was measured and compared with the corresponding estimate produced by model (1).

Based on this comparison, the quality of model (1) was assessed. The assessment criteria were as follows:

1/ For each tested program loop, the actual execution times and the corresponding estimates as per model (1) should change in the same direction.

2/ For each tested program loop, the mean of the relative estimation error for particular parallel versions of the source code should be as low as possible.

Criterion 1/ has been met for all tested cases. Regarding criterion 2/, results are summarized in Table 1.

Table 1. Estimation errors for the tested program loops

Loop name	Mean of the absolute values of relative errors of estimates calculated as per (1)
CG_cg_3	13.49
CG_cg_4	11.93
FT_auxfnct_2	52.64
LU_HP_pintgr_11	15.91
MG_mg_3	28.74
UA_diffuse_2	6.12
UA_diffuse_3	25.44
UA_diffuse_4	22.58
UA_transfer_11	12.22
UA_transfer_16	12.74

The achieved results indicate that model (1) offers a real possibility of improving iterative compilation.

A more thorough discussion of the achieved results is the object of separate papers.

REFERENCES

[1] Kisuki T., Knijnenburg P.M.W., Gallivan K., O'Boyle M.F.P., The effect of cache models on iterative compilation for combined tiling and unrolling, *Concurrency and Computation: Practice and Experience*, 16 (2004), Issue 2-3, 247-270

[2] Bielecki W., *Przetwarzanie równoległe i rozproszone. Część 1. Metody zrównoleglenia algorytmów i tworzenia aplikacji,* Wydawnictwo Uczelnianie Politechniki Szczecińskiej, 2007

[3] Bielecki W., *Essentials of Parallel and Distributed Computing,* Informa, 2002

[4] Stallings W., *Organizacja i architektura systemu komputerowego. Projektowanie systemu a jego wydajność,* Wydawnictwa Naukowo-Techniczne, 2004

[5] Wolfe M., *High Performance Compilers for Parallel Computing,* Addison Wesley, 1996

[6] OpenMP Specifications, http://openmp.org/wp/openmp-specifications/

[7] NAS Parallel Benchmarks, http://www.nas.nasa.gov/publications/npb.html

***Authors**: mgr inż. Agnieszka Kamińska, Zachodniopomorski Uniwersytet Technologiczny w Szczecinie, Katedra Inżynierii Oprogramowania, ul. Żołnierska 49, 71-210 Szczecin, E-mail: agkaminska@wi.zut.edu.pl; prof. dr hab. inż. Włodzimierz Bielecki, Zachodniopomorski Uniwersytet Technologiczny w Szczecinie, Katedra Inżynierii Oprogramowania, ul. Żołnierska 49, 71-210 Szczecin, E-mail: wbielecki@wi.zut.edu.pl*