Agnieszka KAMIŃSKA, Włodzimierz BIELECKI

West Pomeranian University of Technology, Szczecin

Estimation of the execution time of coarse-grained parallel program loops

Abstract. *This paper presents results of experimental research on possibilities of estimating the execution time of coarse-grained parallel program loops based on a regression model. The intended use of the model in question is iterative compilation.*

Streszczenie. *W artykule przedstawiono wyniki badań eksperymentalnych dotyczących możliwości obliczeniowego szacowania czasu wykonania gruboziarnistych, zrównoleglonych pętli programowych w oparciu o model regresyjny. Przewidywanym obszarem zastosowania przedmiotowego modelu jest kompilacja iteracyjna. (**Obliczeniowe szacowanie czasu wykonania gruboziarnistych, zrównoleglonych pętli programowych**).*

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

doi:10.12915/pe.2014.02.23

Introduction

In many practical applications of computers, one of the most crucial problems is the duration of program execution. A common trend is that the duration of program execution has to be as short as possible.

One uses various approaches and techniques in order to achieve the purpose set by this trend. An approach that is worth noticing here is parallel computing which focuses on both software and hardware aspects related to duration of program execution.

In parallel computing, an original problem to be solved is divided into smaller ones, which are then solved concurrently ("in parallel") which in consequence results in shortening the time spent on solving the original problem.

An important characteristic of so comprehended parallelism is its granularity, i.e. the number of operations which are executed between communication or synchronization events. There are two types of granularity (hence, parallelism): coarse-grained granularity (coarse-grained parallelism) and fine-grained granularity (fine-grained parallelism). An application exhibits coarse-grained parallelism if the number of communication or synchronization events between its subtasks is low. An application exhibits fine-grained parallelism if the number of communication or synchronization events between its subtasks is high [1].

There are many ways to divide a problem into smaller problems intended for parallel solving. The only limitation is that particular ways of doing the division have to produce semantically equivalent results. In practice, this means that there has to be semantic equivalence between different versions (source codes) of a program executing a given task.

Currently, there are no theoretical means enabling one to state, at the compilation stage, the actual execution times (in a target hardware environment) of syntactically different source codes of a given program. Hence, as some practical problems and applications (e.g. scientific calculations, embedded systems, etc.) require high efficiency, one carries out so-called iterative compilation. In iterative compilation, many syntactically different source codes of a given program are created and then executed in the target hardware environment. Based on the measured execution times of particular source codes, one selects for final use the source code having the shortest measured execution time [2].

Iterative compilation is an effective yet time consuming approach. One of possible ways of shortening the duration of iterative compilation is to use a model for estimating the efficiency of parallel applications in order to statically (i.e. without executing a program in the target hardware environment) preselect these source codes of a given program that have possibly shortest execution time. Since typically used methods of modelling the efficiency of parallel applications based on the measured program execution time, i.e. Amdahl's law, extrapolation from observation and asymptotic analysis [1] are inadequate for this purpose, the authors have elaborated their own model for the estimation of the execution time of coarse-grained parallel program loops (coarse-grained parallelism exhibits a greater potential than fine-grained parallelism and for this reason has been selected for more thorough analysis).

The elaborated model is a regression model. It presents the dependence between the estimated total CPU time needed for execution of a given program loop and the following factors:

a) A specific way in which the program loop has been parallelized,

b) Specific features of an algorithm executed in the program loop,

c) Parameters of the target hardware environment in which the parallelized program loop will be executed.

The above-mentioned factors have been reflected in the model by means of independent variables, in the following way:

a) A specific way in which the program loop has been parallelized.

This factor has been expressed in the model by means of independent variables *X3* and *X4*, where: *X3* – maximum number of iterations in an iteration chunk per OpenMP thread, *X4* – number of OpenMP threads executing the program.

b) Specific features of an algorithm executed in the program loop.

This factor has been expressed in the model by means of independent variable *X2*, where *X2* – total weighted number of arithmetic operations per OpenMP thread.

c) Parameters of the target hardware environment in which the parallelized program loop will be executed.

This factor has been expressed in the model by means of independent variable *X1*, where:

$$X1 = \frac{\left(sL1 \times aL1 + sL2 \times aL2\right)}{D_f} \tag{1}$$

where: *sL1* – data storage capacity of L1 data cache, available for a single OpenMP thread [B], *sL2* – data storage capacity of L2 cache, available for a single OpenMP thread [B], *aL1* – set associativity of L1 data

cache, *aL2* – set associativity of L2 cache, *Df* – data footprint per OpenMP thread, calculated as per [3, 9] [B].

The model is as follows:

$$Yt = X1^{a1} \times X2^{a2} \times X3^{a3} \times X4^{a4} \quad (2)$$

where: *Yt* – total CPU time, *a1, a2, a3, a4* – parameters whose values are to be estimated by means of regression analysis carried out on empirical data collected in a target hardware environment for a specially prepared sample.

Hereafter, the model specified with equation (2) will be referred to as model (2).

Detailed assumptions of model (2), the criteria used for selecting the variables of the model and its form are presented and discussed in a separate paper (see [4]).

The purpose of this paper is to present results of applying model (2) to the estimation of the execution time of parallel program loops.

Results of experimental research

Experimental research was carried out 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 assessing the performance of parallel supercomputers [5]. The software-hardware environment of the research is presented in Table 1.

Table 1. Software-hardware environment of experimental research

Processor	Intel Core 2 Quad Q6600
Number of processor cores	4
Number of processor threads	4
L1 data cache	4 x 32 KB, 8-way set associative, 64-byte line size
L2 cache	2 x 4096 KBytes, 16-way set associative, 64-byte line size
Operating system	Linux Slax 6.1.2
Compiler	gcc 4.2.4
Version of OpenMP used for parallelization	OpenMP v2.5
Compilation level optimization	None (Optimizations turned off; compilation with the –O0 option)

The first stage of the research was to determine the values of parameters *a1, a2, a3* and *a4* of model (2).

The easiest possible way of determining the values in question would be to determine them separately for each of the ten tested loops, by carrying out a regression analysis of the values of variables *X1, X2, X3, X4* and the actual duration of execution of each version of the loop in the target hardware environment. However, such an approach would be time consuming and therefore, taking into account the intended use of the model (i.e. improving iterative compilation by shortening its duration) – inadequate to fulfil the related needs.

Therefore, instead of determining the values of parameters *a1, a2, a3* and *a4* of model (2) separately for each tested loop, the values in question were determined for 2 reference loops: matmul and nonInterf. Both reference loops have carefully chosen characteristics in respect of data reuse[1] and interference[2].

[1] When data are processed in a program loop, a given sequence of operations is executed many times on varying data. The program loop may many times refer to the data coming from one and the same memory location or adjacent memory locations. These situations are referred to as, respectively, temporal data reuse and spatial data reuse [3, 6].

The criteria used for elaborating the reference loops are as follows:

1/ Presence of data reuse,

2/ Presence of interference resulting from temporal data reuse.

The characteristics of the adopted reference loops in respect of criteria 1/ and 2/ are presented in Table 2.

Table 2. Characteristics of the adopted reference loops

Loop name	Presence of data reuse	Presence of interference resulting from temporal data reuse
matmul	Yes	Yes
nonInterf	Yes	No

Source codes of the adopted reference loops:

The matmul reference loop:

```
int ma[N][N],mb[N][N],mc[N][N];

for (i = 0; i <= N-1; i++) {
  for (k=0; k <= N-1; k++) {
    r = ma[i][k];
      for (j=0; j <= N-1; j++){
        mc[i][j]= mc[i][j] + r*mb[k][j];
      } //endfor j
  } //endfor k
} //endfor i
```

The nonInterf reference loop:

```
int ma[N][N],mb[N][N],mc[N][N],md[N][N],me[N][N];

for (i = 0; i <= N-1; i++) {
  for (j= 0; j <= N-1; j++) {
    ma[i][j]=1;
    mb[i][j]=mc[i][j]+md[i][j]*me[i][j];
  } //endfor j
} //endfor i
```

The values of parameters *a1, a2, a3, a4* of model (2), determined for the matmul reference loop based on empirical data, are as follows:

a1=-0,298695
a2=0,623738
a3=0,014426
a4=0,962976

The values of parameters *a1, a2, a3, a4* of model (2), determined for the nonInterf reference loop based on empirical data, are as follows:

a1=-0,325431
a2=0,675172
a3=-0,082602
a4=0,981967

Next, for each tested loop of the NAS suite, one of the two reference loops was assigned as a related reference loop. The assignment was based on the similarity of characteristics of the tested loops in respect of data reuse and interference, to the same characteristics of the reference loops (see Table 3). In order to state whether there is the similarity between a given tested loop and any

[2] Interference is related to the cache replacement policy (associativity) [7] of processor cache memory. Interference takes place when a cache memory line containing data that can be reused is overwritten with new data despite the fact that there are unoccupied cache lines which could be well used for storing the new data – yet because of the applied cache replacement policy, the new data cannot be stored elsewhere than in some already occupied cache line(s) resulting from the policy in question [8].

of the reference loops, one used data reuse factors: self-temporal reuse factor and self-spatial reuse factor. Both factors are calculated based on solely the source code of a program (i.e. there is no need to execute the program in order to calculate the factors). Self-temporal reuse factors and self-spatial reuse factors are calculated separately for each reference in the source code. If even one of all the self-temporal reuse factors and self-spatial reuse factors calculated for the source code under analysis is greater than 1, then the source code exhibits data reuse. If even one of all the self-temporal reuse factors calculated for the source code under analysis is greater than 1, then the source code exhibits temporal data reuse .The approach used for the calculation of self-temporal reuse factors and self-spatial reuse factors is presented in [3, 9].

The concept of the software prepared by the authors and based on an external library, Clan [11], is presented in [9, 10]. With this software, it is possible to automatically calculate the values of data reuse factors for a given source code. This in turn opens a possibility of the automatic identification of a reference loop related to a given tested loop.

Table 3. Characteristics of the loops selected from the NAS suite

Loop name	Presence of data reuse	Presence of interference resulting from temporal data reuse	Reference loop
CG_cg_3	Yes	No	nonInterf
CG_cg_4			
FT_auxfnct_2			
LU_HP_pintgr_11			
MG_mg_3			
UA_diffuse_2			
UA_diffuse_3	Yes	Yes	matmul
UA_diffuse_4			
UA_transfer_11			
UA_transfer_16			

The next stage of the research was to examine the quality of model (2) for the loops selected from the NAS suite (see Table 3).

For each tested loop and various sizes of matrices processed in these loops, one generated several different but semantically equivalent versions of parallel source code. For each generated version of parallel source code, one measured the duration of its execution on a multi core processor and compared this duration with the corresponding duration estimated as per model (2).

Then, the measurements and the estimates as per model (2) were compared so as to verify model (2) in qualitative and quantitative aspects.

The purpose of the qualitative verification of the model was as follows. To check whether the measured durations of execution of particular versions of the loop and the corresponding estimates as per model (2) change in the same direction. For all tested cases (i.e. pairs comprising the tested loop and the size of matrices processed in the loop), the results of the verification were positive.

The purpose of the quantitative verification of the model was as follows. To check whether mean values of estimation errors for estimates as per model (2) related to the corresponding, measured durations of program execution are acceptable. The quantitative verification of the model was carried out for each tested loop and for each size of matrices processed in the loop. The resultant estimation errors are presented in Table 4.

It should be stressed here that because of the intended use of the model, its quantitative verification is of minor importance in relation to its qualitative verification. In view of the intended use of the model, it is crucial that the model should make it possible to order various source codes of a given loop by execution time, in a descending manner – however, without one's actually executing the source codes in question in the target hardware environment. If, applying the model, it is possible to do such ordering, then the model resultant estimation errors are not so relevant.

Table 4. Estimation errors for estimates as per model (2)

Loop name	Size of the matrix processed in the loop	Mean for the absolute value of the relative estimation error [%] for the estimate as per model (2)	Reference loop
CG_cg_3	75 000	14.27	nonInterf
CG_cg_3	118 000	13.73	nonInterf
CG_cg_3	160 000	12.48	nonInterf
CG_cg_4	100 000	10.48	nonInterf
CG_cg_4	215 000	11.66	nonInterf
CG_cg_4	330 000	13.66	nonInterf
FT_auxfnct_2	30	53.34	nonInterf
FT_auxfnct_2	38	51.54	nonInterf
FT_auxfnct_2	45	53.05	nonInterf
LU_HP_pintgr_11	200	16.42	nonInterf
LU_HP_pintgr_11	265	16.47	nonInterf
LU_HP_pintgr_11	330	14.84	nonInterf
MG_mg_3	26 000	25.96	nonInterf
MG_mg_3	57 444	29.21	nonInterf
MG_mg_3	88 888	31.04	nonInterf
UA_diffuse_2	80 000	6.80	nonInterf
UA_diffuse_2	173 333	5.12	nonInterf
UA_diffuse_2	266 666	6.44	nonInterf
UA_diffuse_3	30	31.60	matmul
UA_diffuse_3	50	16.88	matmul
UA_diffuse_3	71	27.84	matmul
UA_diffuse_4	30	28.55	matmul
UA_diffuse_4	50	12.70	matmul
UA_diffuse_4	71	26.49	matmul
UA_transfer_11	100	10.49	matmul
UA_transfer_11	267	11.30	matmul
UA_transfer_11	433	14.86	matmul
UA_transfer_16	100	12.27	matmul
UA_transfer_16	267	11.10	matmul
UA_transfer_16	433	14.86	matmul

There are large differences in the estimation errors obtained for particular tested loops. The differences in question result from large diversity of the tested loops in respect of:

- Type of reuse of the data processed in particular loops,
- Presence or absence of interference,
- Number of threads executing particular loops,
- Mapping of loop iterations to threads.

However, in view of positive results of the qualitative verification of the model, large differences in the estimation errors obtained for particular tested loops are of minor importance.

Conclusions

The presence of data reuse (in case of the tested loops for which nonInterf was the reference loop) or the presence of data reuse and interference (in case of the tested loops for which matmul was the reference loop) was what the tested loops (see Table 3) had in common. As far as the criteria described by the independent variables of model (2) are considered, there was large variety among the tested loops.

In view of the above and taking into account the intended use of model (2), estimation errors obtained for

particular tested loops (see Table 4) can be regarded as acceptable. Because of large variety of the loops used for verification of model (2), based on the achieved results it can be expected that also for other loops meeting the assumptions of model (2), the model shall produce estimates of acceptable accuracy.

REFERENCES

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

[2] 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, pp. 247-270

[3] Wolfe M., High Performance Compilers for Parallel Computing, Addison-Wesley, 1996

[4] Kamińska A., Bielecki W., Model for the estimation of the execution time of parallel program loops, SoftSec 2013

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

[6] Wolf M.E., Lam M.S., A Data Locality Optimizing Algorithm, *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, Toronto, Ontario, Canada, 1991

[7] Stallings W., Computer Organization and Architecture. Designing for Performance. Sixth Edition, *Pearson Education Inc.,* 2003

[8] Coleman S., McKinley K.S.: Tile Size Selection Using Cache Organization and Data Layout, *Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation*, La Jolla, California, USA, 1995

[9] Kamińska A., Bielecki W., Obliczeniowe szacowanie lokalności danych na poziomie pamięci podręcznej, *Metody Informatyki Stosowanej,* nr 4/2011 (29), pp 33-44

[10] Kraska K., Wierciński T., Kamińska A., Obliczeniowe szacowanie lokalności danych dla programów ANSI-C, *PAK*, nr 08/2011, pp 951-953

[11] Bastoul C., Cohen A., Girbal S., Sharma S., Temam O,. Putting Polyhedral Loop Transformations to Work, *LCPC'16 International Workshop on Languages and Compilers for Parallel Computers LNCS 2958*, 2003

***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*