**Agnieszka KAMIŃSKA, Włodzimierz BIELECKI**

West Pomeranian University of Technology, Szczecin

doi:10.15199/48.2015.02.06

# Statistical models for the estimation of execution time of coarse-grained OpenMP programs

***Abstract***. *This paper presents a family of statistical models for the estimation of program execution time. The paper discusses the possibilities of how to apply the family to reduce iterative compilation duration and in consequence, software development duration. The discussion is supported with the results of experimental research carried out for program loops selected from the NAS Parallel Benchmarks test suite.*

***Streszczenie.*** *W artykule przedstawiono autorską koncepcję rodziny modeli statystycznych do oszacowania czasu wykonania programu oraz omówiono możliwości wykorzystania jej w celu skrócenia czasu wykonywania kompilacji iteracyjnej (a w konsekwencji czasu wytwarzania oprogramowania). Przedstawiono także wyniki przeprowadzonych badań eksperymentalnych. (**Modele statystyczne do oszacowania czasu wykonania aplikacji gruboziarnistych w standardzie OpenMP**).*

**Keywords:** program execution time, iterative compilation, statistical models.
**Słowa kluczowe:** czas wykonania programu, kompilacja iteracyjna, modele statystyczne.

## Problem statement

Limitations resulting from laws of physics impose a barrier on further miniaturization and increase of the speed of uniprocessors. On the other hand, data processing duration is crucial in many practical applications of computers. For these reasons, the use of multiprocessor computers allowing for parallel computing has become an alternative for reducing data processing time by increasing the speed of uniprocessors.

Parallel applications can be created either manually (i.e. by a developer) or automatically (i.e. by dedicated parallelizing compilers). Since the manual creation of parallel applications is very time consuming and more error prone than it is desired, the automatic creation of parallel applications is a more popular approach, especially in case of commercial software development. The idea of this approach is to transform, at the compilation stage, a sequential program into a semantically equivalent parallel program. If the semantics of a given sequential program is such that the program can be parallelized, there is always more than one way of carrying out the parallelization and the main difference between the particular ways is the execution time of the resultant executables in the target environment. This means that the parallelizing transformations selected at the compilation stage influence the execution time of the parallelized program. Within the compilation known as optimizing, one tries to select such transformations that allow us to achieve the shortest execution time of the resultant executable in the target hardware environment

In view of great complexity of the organization and architecture of modern computers, methods used in optimizing compilation do not make it possible to undoubtedly indicate which of possible versions of the source code of a given program will have the shortest execution time in a given target environment. Using these methods, it is possible to find approximate solutions; whereas iterative compilation is still the only way of finding the exact solution. Within iterative compilation, all considered and semantically equivalent source codes of a given program are executed in the target hardware environment; their execution times are compared and the source code with the shortest execution time is selected for final use. Iterative compilation can be very time consuming and thus, costly in practical applications, especially in case of commercial software development. Therefore, a potential improvement in iterative compilation is to use a mathematical model in order to select from possible source code variants of a given program the ones with shortest expected execution times and then, limit the empirical selection of the best source code to the so reduced set. In such a way, the time of software development can be reduced – since, by using estimations, one can quickly focus iterative compilation on empirical verification (execution in the target environment) of solely the most promising source code variants.

Potential practical advantages related to the proposed improvement in iterative compilation and the scientific gap found in this area have become an inspiration for the authors' solution presented in this paper and involving the elaboration of the family of iterative compilation oriented statistical models for the estimation of program execution time.

Since most of time consuming operations – calculations made within computer programs – are executed in loops, the scope of applicability of the elaborated family of statistical models has been limited to a class of parallelized loops, which is often used in practice: coarse grained loops, parallelized in the OpenMP C/C++ standard. Coarse grained granulation [1] takes place when the duration of execution of data processing related operations in the program is longer than the total duration of initializing these operations and transfer of the data needed for the execution of these operations. This type of granulation corresponds with the nested loop structure in which the outermost loop of the nest is parallelized.

Coarse grained granulation is typically used in parallelization of programs executed by currently very popular multiprocessor machines with shared memory [2].

## Family of statistical models for the estimation of program execution time

A family of statistical models for the estimation of program execution time is based on authors' general model, i.e. the general equation of a function which makes it possible to estimate the execution time of coarse grained program loops parallelized in the OpenMP C/C++ standard.

Program execution time has been assumed as the dependent variable of a general model. One has assumed that quantitative variables reflecting factors which significantly influence program execution time should be the independent variables of the general model. Apart from the dependent and independent variables, the general model comprises parameters whose values are unknown a priori.

It has been decided that the values of these parameters should be determined for a specific hardware environment, based on the regression analysis carried out for empirical data collected in this environment. In order to collect the required empirical data, we have used programs prepared specially for this purpose. These programs are hereafter referred to as pattern program loops. Each pattern program loop represents a combination of some arbitrarily assumed characteristics related to data reuse and interference. Interference takes place when a cache line containing data which can be reused in the program is overwritten with new data, despite the fact that there is sufficient unoccupied space in the cache where the new data could well be fetched – however, because of the cache organization, a specific and already occupied cache line has to be overwritten instead [3, 4, 5, 6].

In order to elaborate a family of statistical models, we have used two exemplary pattern program loops: nonInterf (representing data reuse with no interference) and matmul (representing data reuse with interference).

The source codes of the both pattern program loops are presented in Table 1.

Table 1. Pattern program loops

| Pattern program loop 1:<br>Data reuse with no interference | Pattern program loop 2:<br>Data reuse with interference |
|---|---|
| Loop nonInterf | Loop matmul |
| int ma[N][N],mb[N][N],mc[N][N],<br>md[N][N],me[N][N];<br>int i, j, N;<br><br>for (i = 0; i <= N-1; i++) {<br>  for (j = 0; j <= N-1; j++) {<br>  ma[i][j] = 1;<br>  mb[i][j] = mc[i][j] +<br>  md[i][j]*me[i][j];<br>  } //endfor j<br>} //endfor i | int ma[N][N],mb[N][N],mc[N][N];<br>int i, j, k, r, N;<br><br>for (i = 0; i <= N-1; i++) {<br>  for (k = 0; k <= N-1; k++) {<br>  r = ma[i][k];<br>  for (j = 0; j <= N-1; j++){<br>  mc[i][j] = mc[i][j] +<br>  r*mb[k][j];<br>  } //endfor j<br>  } //endfor k<br>} //endfor i |

After substituting the parameters of the general model with values, the general model becomes a specific one. The specific model specifies the general model for a particular situation, by assigning relevant values to the parameters of the general model.

Each specific model is derived from the general model for a particular pattern program loop. The specific model can be applied both to its pattern program loop and to other programs with the same data reuse type as the pattern program loop. The other programs in question are hereafter referred to as non pattern program loops.

In order to avoid the extrapolation of the specific model beyond the data range for which the model is constructed, we have elaborated assumptions regarding the scope of applicability of the specific model to non pattern program loops.

**Form of the general model**

The execution time of every program results from the interaction of many various factors. The following factors influence program execution time:

a) the structure of the parallel program and the type of parallelism exposed by the program,

b) the specificity of the problem solved in parallel,

c) parameters of the hardware environment in which the parallelized program is to be executed.

In the model, particular factors are represented by quantitative variables, in the following way:

a) The structure of the parallel program and the type of parallelism exposed by the program

In the OpenMP C/C++ standard, programs are parallelized by multithreading. The time of the execution of the parallelized program depends on the number of invoked OpenMP threads – therefore, the number of OpenMP threads executing the program has been adopted as an independent variable (*X4*) of the general model.

The duration of execution of the entire task (program loop) is determined by the execution time of the thread which has been assigned to execute the greatest number of iterations, and in particular by the size of the largest chunk of iterations assigned to this thread. Therefore, we have adopted as an independent variable (*X3*) of the general model the maximum number of iterations in a single chunk of iterations assigned to be executed by an OpenMP thread for a given assignment of iterations to OpenMP threads.

b) The specificity of the problem solved in parallel

From the low level perspective, the specificity and variety of problems solved in computer programs are reflected in the number and type of arithmetic operations to be executed by a processor. A simple yet effective way of expressing this observation quantitatively is to assign different weights to different types of arithmetic operations. The weights should be selected based on the analysis of the execution times of instructions of a given processor. Therefore, the total weighted number of arithmetic operations per single program thread has been adopted as an independent variable (*X2*) of the general model.

c) Cache and its organization

The program execution time 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 reuse of the data stored in the cache memory; the minimum data storage capacity in question can be estimated by means of data footprint (according to the methods presented in [7] and [8]); in order to calculate the data footprint for a given program, it is sufficient to know its source code; there is no need to execute this program.
3. the relation between 1. and 2.

In connection with the above, the relation between 1. and 2. has been adopted as an independent variable (*X1*) of the general model.

With such a list of independent variables of the model to be formulated and assuming that the dependent variable is *Yt* representing estimated CPU time for the execution of the program loop by all program threads, expressed by the number of CPU clock cycles, we have carried out a regression analysis. Empirical data for the regression analysis have been collected for two pattern program loops (nonInterf and matmul) prepared specially for that purpose.

According to the assumptions of the linear regression, the dependency between the observed values of dependent variable *Y* and corresponding values of independent variables *X1, X2, …, Xp* is expressed by the following expression (1):

$$Y_i = a_0 + a_1 X1_i + a_2 X2_i + \ldots + a_p Xp_i + \varepsilon_i = Yt_i + \varepsilon_i \quad (1)$$

where: $i$ – $i$ = 1, 2, …, $n$ are identifiers of observations, $a_0$, …, $a_p$ – parameters of unknown exact values; the values of these parameters are estimated by means of the classical method of least squares, $X1_i$, …, $Xp_i$ – known values of independent variables *X1*, *X2*, …, *Xp*, corresponding to the value of variable *Y* observed in the $i^{th}$ observation, $Y_i$ – value of dependent variable *Y* observed in the $i^{th}$ observation, $Yt_i$ – theoretical (estimated) value of dependent variable *Y* for the $i^{th}$ observation, $\varepsilon_i$ – statistical error (disturbance, noise) for the $i^{th}$ observation.

Equation (1) can be applied when the dependency between empirical values of the dependent variable and independent variables is linear. Equation (1) can be also applied when the dependency between the variables is a nonlinear dependency represented with a nonlinear function however, after applying relevant mathematical operations (e.g. logarithms), the nonlinear function can be transformed to a linear equivalent. This kind of transformation can be carried out for the following types of nonlinear functions: power, exponential, logarithmic or hyperbolic.

Therefore, for independent variables: *X1, X2, X3, X4* and dependent variable *Yt* the general model (which is a linear regression model derived by means of the classical method of least squares) could take one of the following forms: a linear form, a power form, an exponential form, a logarithmic form or a hyperbolic form.

Taking into account the nature of variables *X1, X2, X3, X4, Yt* and their mutual relations, one could assume that the dependency between all these variables can be described with a power expression.

This assumption has been verified by the examination of the value of coefficient of determination ($R^2$) calculated for:

- variable *Yt* and all independent variables considered altogether (case 1/),
- variable *Yt* and particular independent variables considered individually (cases 2/ ÷ 5/).

The values of the coefficient of determination obtained for the both pattern program loops are presented in Table 2. For the both pattern program loops, the highest value of $R^2$ for case 1/ has been obtained for a power model. Moreover, for both loops, the power model is very well fitted for all other cases, which proves that there is a power dependency between the dependent variable and each of the independent variables of the model.

Table 2. Values of coefficient of determination, for various possible forms of the general model – for the nonInterf loop and for the matmul loop

| Form of the model | Case 1/ $R^2_{Yt.X1,X2,X3,X4}$ | Case 2/ $R^2_{Yt.X1}$ | Case 3/ $R^2_{Yt.X2}$ | Case 4/ $R^2_{Yt.X3}$ | Case 5/ $R^2_{Yt.X4}$ |
|---|---|---|---|---|---|
| Loop nonInterf | | | | | |
| linear | 0.9738 | 0.0602 | 0.9239 | 0.6125 | 0.6390 |
| power | 0.9999 | 0.8968 | 0.9957 | 0.9653 | 0.9203 |
| exponential | 0.9845 | 0.3399 | 0.7284 | 0.8848 | 0.9194 |
| logarithmic | 0.9557 | 0.4977 | 0.7366 | 0.6611 | 0.6387 |
| hyperbolic | 0.9458 | 0.9239 | 0.0602 | 0.5872 | 0.5997 |
| Loop matmul | | | | | |
| linear | 0.9506 | 0.0002 | 0.9286 | 0.3616 | 0.4771 |
| power | 0.9999 | 0.6540 | 0.9982 | 0.9119 | 0.9183 |
| exponential | 0.9645 | 0.1066 | 0.5703 | 0.4310 | 0.9170 |
| logarithmic | 0.8230 | 0.8095 | 0.5858 | 0.5074 | 0.4774 |
| hyperbolic | 0.8098 | 0.7669 | 0.0014 | 0.3219 | 0.4602 |

Based on the above presented results of the regression analysis, a regression power model with variables *Yt, X1, X2, X3, X4* and parameters *a1, a2, a3, a4* has been adopted as the general model. Hence, the final form of the general model is:

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

where: *a1, a2, a3, a4* are parameters whose values have been determined within the regression analysis on the empirical data collected in the target software-hardware environment for a specially prepared sample.

**Estimation of the values of parameters for specific models**

It has been assumed that the values of parameters for specific models should be determined in a methodical way that could be applied to any environment. Therefore, it has been decided to determine the values of parameters *a1, a2, a3, a4* for a given hardware environment by means of the statistical analysis of empirical data collected in this environment.

To determine the values of parameters *a1, a2, a3, a4* we have used pattern program loops. Each pattern program loop represents a different type of data reuse in the program. We have used two pattern program loops: nonInterf and matmul.

Each of the pattern program loops represents some arbitrarily selected characteristics related to data reuse and interference.

Taking into account data reuse and interference, program loops can be classified as follows:

1. Loops with no data reuse – in practice, very rarely used and therefore, not included in the model
2. Loops with data reuse

2a. Without interference – pattern program loop: nonInterf

2b. With interference – pattern program loop: matmul

The nonInterf loop exposes data reuse but no interference.

The matmul loop exposes data reuse and interference.

The source codes of the loops nonInterf and matmul are presented in Table 1.

It should be stressed here that pattern program loops nonInterf and matmul are exemplary pattern program loops with the characteristics as indicated in Table 1. These loops have been adopted in order to determine exemplary specific models using the general model (2). This realization of pattern program loops (i.e. by the nonInterf and matmul loops) is one of many possible realizations. Assuming some other realization of pattern program loop 1 and pattern program loop 2, one could derive specific models with domains different from the domains of specific models derived from pattern program loops nonInterf and matmul. This in turn means that the proposed approach is highly universal, as it provides the possibility of changing the domain of a specific model simply by modifying the pattern program loop for the model.

In order to obtain empirical data that are representative for the environment under analysis, it has been assumed that for each pattern program loop the following is true:

1. the total size of the data processed in the loop does not exceed the size of the L2 processor cache,
2. the relative difference between the mean and maximum number of iteration chunks per single OpenMP thread for a given assignment of iterations to OpenMP threads does not exceed 50 % (the value assumed a priori).

For assumptions 1. and 2., the exemplary pattern program loops and hardware environment of empirical research, one has derived the following specific models:

- for the nonInterf pattern program loop:

$$Yt = X1^{-0.325431} \times X2^{0.675172} \times X3^{-0.082602} \times X4^{0.981967} \quad (3)$$

- for the matmul pattern program loop:

$$Yt = X1^{-0.298695} \times X2^{0.623738} \times X3^{0.014426} \times X4^{0.962976} \quad (4)$$

The results of regression analysis (i.e. the resultant regression model) should not be extrapolated outside the data range for which the regression model has been constructed, since the character of the dependency

between values of independent and dependent variables is unknown outside the data range in question.

To avoid the risk of such an extrapolation while applying specific models to non pattern program loops, we have formulated the following, detailed assumptions regarding the scope of applicability of specific models.

1. The total size of the data processed in the loop does not exceed the size of the L2 processor cache. Moreover, the total size of the data processed in a non pattern program loop is not less than and not greater than the size of the data processed in the pattern program loop for which the specific model has been derived.
2. The relative difference between the mean and maximum number of iteration chunks per single OpenMP thread for a given assignment of iterations to OpenMP threads does not exceed 50 %.
3. The actual duration of execution of a non pattern program loop in the target environment is not less than and not greater than the measured, empirical duration of execution of the pattern program loop for which the specific model has been derived.

**Results of experimental research**

In order to prove that the proposed model is indeed useful in iterative compilation, we have used the NAS Parallel Benchmarks (NPB) test suite [9, 10]. NPB has been used as it is a test suite dedicated for the assessment of the performance of parallel computers and consists of a great number of very various loops.

8 NPB program loops were selected for the experimental research. The selected program loops were different from the pattern program loops, but had the same type of data reuse as the pattern program loops. By means of the exemplary specific models, one estimated execution times of various source code variants of the 8 selected program loops. In total, one estimated execution times for 133 various source codes.

Table 3. Quality assessment of estimates calculated according to specific models (3) and (4)

| Loop (specific model) | Size of the problem solved in the loop | Number of various source code variants subjected to the estimation of execution time | Resultant mean for $\delta_{Yt(per\_thread)}$ | Resultant maximum for $\delta_{Yt(per\_thread)}$ |
|---|---|---|---|---|
| CG_cg_4 (3) | 215 000 | 8 | 11.66 | 24.43 |
| CG_cg_4 (3) | 330 000 | 8 | 13.66 | 27.06 |
| FT_auxfnct_2 (3) | 30 | 8 | 53.34 | 60.43 |
| FT_auxfnct_2 (3) | 38 | 9 | 51.54 | 60.75 |
| LU_HP_pintgr_11 (3) | 265 | 8 | 16.47 | 32.85 |
| LU_HP_pintgr_11 (3) | 330 | 8 | 14.84 | 28.76 |
| MG_mg_3 (3) | 26 000 | 6 | 25.96 | 34.41 |
| MG_mg_3 (3) | 88 888 | 6 | 31.04 | 40.51 |
| UA_diffuse_3 (4) | 30 | 9 | 31.60 | 38.46 |
| UA_diffuse_3 (4) | 50 | 9 | 16.88 | 24.02 |
| UA_diffuse_4 (4) | 30 | 9 | 28.55 | 35.80 |
| UA_diffuse_4 (4) | 50 | 9 | 12.70 | 20.18 |
| UA_transfer_11 (4) | 100 | 9 | 10.49 | 20.44 |
| UA_transfer_11 (4) | 267 | 9 | 11.30 | 29.81 |
| UA_transfer_16 (4) | 267 | 9 | 11.10 | 29.05 |
| UA_transfer_16 (4) | 433 | 9 | 14.86 | 34.97 |

For each of the selected NPB loops, the trend of changes in the measured execution times of particular variants of a given loop has matched the trend of changes in the corresponding estimations calculated according to the relevant specific model.

The mean and maximum relative estimation error calculated in relation to execution times measured empirically for all source code variants adopted for a given loop and size of the problem solved in the loop have not exceeded respectively 55 and 65 per cent points (detailed results are presented in Table 3).

For each of the selected NPB loops, we have also estimated the shortening of iterative compilation duration (and in consequence, the related software development duration) that could be achieved by applying specific models in accordance with the authors' procedure on how to support iterative compilation with such models. The meaning of the symbols used is as follows:

$t$ – number of all various input source code variants for a given loop,

$k$ $(0 < k < t)$ – value given by the user and denoting the assumed number of source code variants with shortest estimated execution times for a given loop,

$k_{min}$ – minimum value of $k$ which guarantees that one selects for final use the source code variant with the shortest execution time measured in the hardware environment.

The achieved results are summarized in Table 4.

Table 4. Shortening of iterative compilation duration after applying specific models (3) and (4)

| Loop (specific model) | Size of the problem solved in the loop | $t$ | $k_{min}$ | Iterative compilation duration ($T$) for | | Shortening of iterative compilation duration = ($Tt$/$Tk_{min}$) |
|---|---|---|---|---|---|---|
| | | | | $t$ source code variants | $k_{min}$ source code variants | |
| CG_cg_4 (3) | 215 000 | 8 | 1 | 4.33E+03 | 3.70E+02 | 11.73 |
| CG_cg_4 (3) | 330 000 | 8 | 1 | 6.63E+03 | 5.61E+02 | 11.83 |
| FT_auxfnct_2 (3) | 30 | 8 | 1 | 1.97E+03 | 1.71E+02 | 11.53 |
| FT_auxfnct_2 (3) | 38 | 9 | 3 | 4.26E+03 | 1.12E+03 | 3.82 |
| LU_HP_pintgr_11 (3) | 265 | 8 | 1 | 4.19E+03 | 3.58E+02 | 11.71 |
| LU_HP_pintgr_11 (3) | 330 | 8 | 1 | 6.46E+03 | 5.48E+02 | 11.80 |
| MG_mg_3 (3) | 26 000 | 6 | 1 | 1.42E+03 | 1.68E+02 | 8.41 |
| MG_mg_3 (3) | 88 888 | 6 | 2 | 4.72E+03 | 1.11E+03 | 4.27 |
| UA_diffuse_3 (4) | 30 | 9 | 2 | 4.75E+04 | 7.43E+03 | 6.39 |
| UA_diffuse_3 (4) | 50 | 9 | 1 | 3.67E+05 | 2.87E+04 | 12.81 |
| UA_diffuse_4 (4) | 30 | 9 | 1 | 4.55E+04 | 3.55E+03 | 12.80 |
| UA_diffuse_4 (4) | 50 | 9 | 1 | 3.49E+05 | 2.73E+04 | 12.80 |
| UA_transfer_11 (4) | 100 | 9 | 1 | 3.94E+04 | 3.08E+03 | 12.82 |
| UA_transfer_11 (4) | 267 | 9 | 1 | 7.54E+05 | 5.89E+04 | 12.81 |
| UA_transfer_16 (4) | 267 | 9 | 1 | 7.53E+05 | 5.88E+04 | 12.81 |
| UA_transfer_16 (4) | 433 | 9 | 1 | 3.22E+06 | 2.51E+05 | 12.80 |

**Related work**

Within the research on optimizing compilation and program execution time, various methods for forecasting program execution time [11], estimating program execution time [12] or selecting the program source code with the shortest expected execution time [13, 14, 15] have been proposed.

The approaches, proposed in [11, 12, 13, 14, 15], provide good tools for the estimation of program execution time, yet the process of the construction of the related mathematical models is time consuming and the scope of the applicability of the resultant models is narrow (the models are dedicated for specific programs or optimizing transformations). In view of these characteristics, the approaches in question are not adequate for carrying out the proposed improvement of iterative compilation, which involves analytical selection from possible source code variants of a given program the ones with shortest expected execution times in order to limit the empirical selection of the best source code thereto.

In contrast to the aforementioned methods, applying the authors' solution presented herein it is possible to elaborate models for the estimation of program execution time, which are adequate both for the pattern program loops for which the models have been derived and for completely different (non pattern) program loops which have only the data reuse type in common with the pattern program loops.

**Summary**

This paper presents the authors' family of statistical models for the estimation of program execution time. The family consists of a general model and specific models. The family has been elaborated based on the empirical data collected for pattern program loops representing some arbitrarily selected features related to the program structure and the specificity of program execution environment.

The elaborated general model (and hence, each specific model derived from the general model) is dedicated to programs parallelized in the OpenMP C/C++ standard and in its current form, i.e. with its current independent variables, cannot be applied to programs parallelized in any other standard. However, the idea of pattern program loops and creating specific models based on pattern program loops could be used for elaborating models for programs parallelized in different standards than OpenMP C/C++ – yet, in such a case, one would have to elaborate a new list of independent variables dedicated to the used parallelization standard.

Exemplary specific models belonging to the family have been used to estimate execution times of non pattern program loops. The accuracy of the estimations is satisfactory.

We have also estimated the shortening of iterative compilation duration (and in consequence, the related software development duration) which could be achieved by applying specific models in accordance with the proposed procedure of supporting iterative compilation with such models. For the exemplary loops presented in the paper and coming from the NPB test suite, the duration of iterative compilation has been shortened from approximately 4 to approximately 13 times (detailed results are presented in Table 4).

The achieved results show that the authors' solution presented in the paper is adequate for use in iterative compilation for optimization purposes and at the same time gives a possibility of shortening the duration of software development.

REFERENCES


[1] Ishizaka K., Obata M., Kasahara H., Coarse grain task parallel processing with cache optimization on shared memory multiprocessor, *Languages and Compilers for Parallel Computing*, 2003, 352-365

[2] Kazi I.H., Lilja D.J., Coarse-grained Speculative Execution in Shared-memory Multiprocessors, *Proceedings of the 1998 International Conference on Supercomputing – ICS '98*, 1998, 93-100

[3] Aho A.V., Lam M.S., Sethi R., Ullman J.D., Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley, 2006

[4] Coleman S., McKinley K.S., Tile Size Selection Using Cache Organization and Data Layout, *ACM SIGPLAN Notices*, 30 (1995), Issue 6, 279-290

[5] Esseghir K., Improving data locality for caches. Master's thesis, Department of Computer Science, Rice University, 1993

[6] Temam O., Fricker C., Jalby W., Cache interference phenomena, *ACM SIGMETRICS Performance Evaluation Review*, 22 (1994), Issue 1, 261-271

[7] Lam M.S., Rothberg E.E., Wolf M.E., The Cache Performance and Optimization of Blocked Algorithms, *ACM SIGARCH Computer Architecture News*, 19 (1991), No. 2, 63-74

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

[9] Haoqiang J., Frumkin M., Yan J., The OpenMP implementation of NAS parallel benchmarks and its performance. Technical Report NAS-99-011, NASA Ames Research Center, 1999

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

[11] Berlińska J., Metody tworzenia modeli statystycznych charakteryzujących aplikacje równoległe i rozproszone. Rozprawa doktorska. Politechnika Szczecińska, 2005

[12] Lokuciejewski P., Stolpe M., Morik K., Marwedel P., Automatic Selection of Machine Learning Models for WCET-aware Compiler Heuristic Generation, *Proceedings of the 4th Workshop on Statistical and Machine Learning Approaches to Architectures and Compilation (SMART)*, 2010, 3-17

[13] Cavazos J., O'Boyle M.F.P., Method-Specific Dynamic Compilation using Logistic Regression, *ACM SIGPLAN Notices*, 41 (2006), No. 10, 229-240

[14] Park E., Kulkarni S., Cavazos J., An Evaluation of Different Modeling Techniques for Iterative Compilation, *Proceedings of the 14th international conference on compilers, architectures and synthesis for embedded systems*, 2011, 65-74

[15] Pekhimenko G., Brown A.D., Efficient Program Compilation through Machine Learning Techniques, *Software Automatic Tuning*, 2010, 335-351



***Authors**: dr inż. Agnieszka Kamińska, Zachodniopomorski Uniwersytet Technologiczny w Szczecinie, Katedra Inżynierii Oprogramowania, ul. Żołnierska 49, 71-210 Szczecin, E-mail: agnieszka kaminska@wp.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*