Procedures classification for optimizing strategy assignment

O.A. Chetverina <chetverina_o@mcst.ru >,
ZAO MCST, Leninskii prospect, 51, Moscow, 119991, Russian Federation

Abstract. Optimizing compilers make significant contribution to the performance of modern computer systems. Among them VLIW architecture processors are the most compiler-dependent, since their performance is ensured by effective compile time scheduling of multiple commands in a single clock. This leads to an eventual complication of VLIW compilers. Taking as an example optimizing compiler developed for the Elbrus family processors, it runs consequently over 300 stages of code optimization in basic mode. Such an amount of stages is needed to obtain decent performance, but it also makes compilation quite time consuming. It turns out that the main reason for compilation time increase when using high level compilation is applying some aggressive unreversable code transformations, which eventually leads to code size increase that is also unwanted. In addition, there remains the problem of using a number of optimizations that are useful for rare contexts. To reach the objectives, namely increasing performance, decreasing compilation time and code size, it is reasonable to choose an appropriate strategy on an early compilation stage according to some procedure specific characteristics. This paper discusses the procedures classification problems for this task and suggests several possible solutions.

Keywords: optimizing compiler; optimizing phases sequence; performance tuning; reducing compilation time; procedures classification.

1. Introduction

To obtain decent performance modern optimizing compilers apply a huge sequence of code transformations. Usually compilers use a fixed optimization sequence for all procedures according to optimization level (-O0, -O1, -O2, -O3) and each optimization stage tries to improve performance of available code segments using statistically proven heuristics which leads to suboptimal results in most cases [1, 2]. In order to achieve the best possible performance for a given program it is important to find the most suitable optimization sequence for each procedure. This could be done with iterative approaches, which compile procedures in a given program using different optimization sequences with either executing the resulting code [3,4] or estimating the execution time [5] and choosing the best one. Although both techniques achieve good performance results on a number of tasks, their weak spots is a need of a large compilation time which is not always acceptable and a necessity

to execute tasks on appropriate input data so that the training runs would match the further execution in terms of branch probabilities and code coverage. The importance and difficulty of constructing a good training input data can be demonstrated with profiling data that was collected using train execution of the spec2000 benchmark [6] using Elbrus compiler. It was found out for this benchmark that applying a low-optimizing sequence to the procedures with zero train profile data leads to a 6% performance degradation of CFP tasks of spec2000 on average. The biggest decelerations occurred on 179.art (-18%) and on 301.apsi (-47%), where the reason for 301.apsi degradation is that one of its main procedures never executes during train run. As for huge applications it is often too difficult to generate good train data, which will cover all important parts of code, moreover, for some types of code like libraries or operational system it is nearly impossible. Also it should be mentioned that in most cases high compilation time corresponds with the resulting code size growth, this happens because most time-consuming phases including hyper-blocks construction, scheduling and loop software pipelining are located in the end of optimization line and the time they work corresponds with the size of the intermediate code that was made as result of different aggressive loop and acyclic transformations such as splitting, peeling, tail duplication etc.

Earlier researches in the field of iterative compilers [7,8] offer techniques that allow to construct a set of optimization sequences that cover the given procedures space rather well. In those works to minimize the needed execution time authors choose a possibly small set of options or sequences that show performance increase on most tests. To reach good performance results with affordable compilation time and resulting size of code and avoid the need of training executions it is reasonable to try to choose a compilation sequence from such a set on an early compilation stage using some characteristics of the procedure. The main goal of this research is to explore and construct the possible methods of procedures classification that would allow to perform this objective.

First of all it would be shown that to make a good selection of optimization sequences for a set of procedures using characteristics a compilation quality functional is needed (section 2). It would also be explained how to construct a functional to take several factors into consideration, like execution time, compilation time, resulting code size and other possible limits. Then the task of predicting good sequences selection for a given number of procedures would be formulated in terms of minimizing constructed quality functional (section 3). After a list of main existing methods of classification and clusterizations would be described and given a possible one that allows to solve the task. In section 4 some experimental results would be provided.

2. Compilation quality functional

To make a statistical solution of procedures types selection a large training set is needed. For this purpose all procedures of spec2000 benchmark with a full input data were used. The reason for this pack choice is that it is well balanced in terms of

different types of tasks and is used as a performance benchmark for most high-performance computers. The steps for solution is to choose the best sequences assignment for the training set using full statistic on compilation, execution or other important characteristics and then to make an attempt to predict it using only procedures information available on early compilation stage.

Any type of classification and clusterization methods perform allocation of areas in parameters space, which are then respectively called classes or clusters and could be used to make some assignment of type, in our case an assignment of optimization sequence. Using an example from table 1 it could be easily seen that a need to construct a quality functional comes up even when the only goal of classification is to minimize execution time.

Table 1. Example of sequence choice

	Sequence 1 time	***Sequence 2 time***	***Best sequence***
Procedure 1	100	50	2
Procedure 2	95	100	1
Procedure 3	100	105	1
Sum time	295	255	2

Suppose there are 3 procedures that hit the same area in parameters space, in the shown example the best sequence choice for 2 out of 3 procedures would lead to decrease of performance both in sum and on average. It could be assumed that procedures with different optimal sequences should be in different areas but actually this assumption is wrong because even the same procedure with different input data could lead to different best choices results. This means that there is a need to construct a numerical evaluation method that would qualify the sequences assignments on the whole set of procedures. The most common technique to formalize the understanding of the best choice is to construct a functional, which reaches minimum at decision point. In this case the domain for such functional is an *assignment space for procedures*:

$P = \{p_1, \dots p_n\}$ – all procedures in a set

$L = \{l_1, \dots l_k\}$ – the list of optimization sequences,

$F(l(p_1), \dots l(p_n)) \to R$ – a functional defined on the space L^n, where $l: P \to L$

To minimize the execution time the following functionals could be chosen:

$exe(p_i, l(p_i))$ - execution time of procedure p_i when compiled using $l(p_i)$ sequence, then

$$F\big(l(p_1), \dots l(p_n)\big) = \sum_i exe(p_i, l(p_i)) \quad (1)$$

$$F\big(l(p_1), \dots l(p_n)\big) = \prod_i exe(p_i, l(p_i)) \quad (2)$$

A functional that considers not only the execution time, but also compilation time could be constructed:

$comp(p_i, l(p_i))$ - compilation time of procedure p_i when compiled using $l(p_i)$ sequence

$$F\big(l(p_1), \dots l(p_n)\big) = \left(\sum_i exe(p_i, l(p_i))\right)^r \left(\sum_i comp(p_i, l(p_i))\right) \quad (3)$$

This functional describes the acceptable ratio of performance loss and compilation gain, larger values of "r" mean higher importance of performance over compilation. Though even with infinite value of r compilation could be reduced in case if 2 sequences produce the same code in terms of execution time. Other important limitation as code size could be introduced into quality functional similarly.

3. Functional minimizing classification

Suppose a quality functional was already chosen, then classification task could be formulated in the following terms:

$P = \{p_1, \dots p_n\}$ – all procedures in a set

$L = \{l_1, \dots l_k\}$ – the list of optimization sequences,

H– the space of procedures characteristics

$Ch: P \to H$ – assignment of characteristic vector for procedures

$F(l(p_1), \dots l(p_n)) \to R$ is defined on the space L^n, where $l: P \to L$

Then the classification is an allocation of areas S in the space H with a sequence vector in L that produces a constant assignment for each area S, that is:

$$\forall S \quad l\big(Ch^{-1}(S)\big) = const$$

The goal is to make a classification (with some minimal number of training elements in the area = q), that minimizes the given functional:

$$F\big(l(p_1), \dots l(p_n)\big) \to min \quad (4)$$

To substantiate the statistical approach it is reasonable to require for each procedure p_k having a locality D in characteristic space containing at least q points for which

$$H(p) = \begin{cases} l(p_k), p \in D \\ default, p \notin D \end{cases} \quad (5)$$

$$F\big(H(p_1), \dots H(p_n)\big) \le F(default, \dots default)$$

3.1 Procedures characteristics

As was mentioned earlier the major use of such early compilation stage sequence prediction is expected on codes that for some cases are not suitable for training execution. So the goal is to choose a number of characteristics that work well enough to predict a good optimization sequence and do not depend on precise profile information. To choose the best set different characteristics were considered and using correlation matrix the most valuable were picked and normalized. The best characteristics that were found to predict the optimal compilation sequence with no train profiling information are:

- number of operations in the procedure;
- average node size, which in some sense stand for the branch frequency;
- number of call operations;
- maximum loop level in a procedure;
- average operation counter, which could also be considered as *procedure density*;
- percentage of operation of field reads;
- percentage of operations with floating point;
- percentage of operations that calculate an address for a read.

Most of those are profiling data independent, though the average operation counter is not. In case of no train profile information Elbrus compiler uses a predicted profiling based on statistical information. It was found to be good enough to use this static profiling for classification.

3.2 Ideal theoretical solution

First of all for the given training space that includes all characteristics, which are used in quality functional, an optimal solution that stands for the minimum functional point could be calculated. For the chosen functional (3) and the considered lines finding the minimum required making about 2*n steps of gradient descent, that is 2*n steps, where on each we make a change of a coordinate in assignment vector that gives the maximum functional value decrease. To check the stability of the resulting vector in L^n several starting points with the constant assignment of each line for all set of procedures were used. The solution is a vector with *n* coordinates where *n* is the number of procedures in the training set:

$$\left(l_{b_1}, l_{b_2} \dots l_{b_n}\right) \quad (6)$$

Sequence vector (6) would be called the *optimal theoretical vector of sequences* for procedures *P*, where l_{b_i} is the *optimal theoretical sequence* for procedure p_i. It should be noted that l_{b_i} would not always afford the best performance or performance with compile time result on procedure p_i. It is optimal only in sense of the whole set of considered procedures, which is due to functional minimum.

As it would be shown in experimental section solution (6) doesn't always lead to best results on a real run when assigning the corresponding compilation sequences

for all procedures in program, and therefore it is declared theoretical. This occurs because statistical information for each procedure is collected with simultaneous sequences assignment for other procedures in the program, modification of those procedures sometimes leads to other memory usage interaction and as a result to different execution time. The only way to completely avoid this effect is to collect statistical information for all possible configurations, which is not feasible and even to be partially used requires availability of information for all additionally executable procedures to make the right choice for the given one. Therefore, it was decided to drop out this fact in the currently constructed solution, though keep it in consideration for future researches in case of –fwhole-program compilation mode.

3.3 Existing classification and clusterization methods

Unlike to methods of clusterization [9] in this situation it is impossible to construct a metric that would determine the valuable in terms of our needs distance between procedures. The reason is that the distance between couples of procedures would depend on the other procedures in same cluster. For this case the clusterization methods allow to selects areas according to only characteristic metrics, but it is possible only with appropriate characteristics normalization. The uniform normalization by itself works out bad for this task, thought probably some techniques that use functional value movement with characteristic change could be developed.

Classification methods (support vector machine - SVM, Bayesian network) don't require to construct a metric that would divide classes. But as was mentioned before it is not enough to increase the possibility of picking the best sequence when using procedures characteristics for prediction. Though in the first attempt to make a classification solution a Bayesian network [10] has been tried. Although it showed a high percent of an optimal sequence prediction (above 95%) the resulting execution time of training tasks set increased by 21% on average. It was found out that the most frequently optimal sequence reduced the performance of some weighty procedures, which required a number of aggressive transformations to achieve acceptable performance. Due to this reason even a small percent of mistakes leaded to unacceptable result. Other considered methods have the same problem - the maximum that they allow is to add a weight to the mistake when choosing the wrong solution, which in our case means not optimal, but they don't differ the value of a mistake.

3.4 Procedures classification

To solve this problem a cluster error minimization algorithm was developed. First we construct the full error table. For each sequence l_{i_k} and for each procedure p_k the minimization error is the following

$$err\left[p_k, l_{i_k}\right] = \log\left(F\left(l_{b_1} \dots l_{i_k} \dots l_{b_n}\right)/F\left(l_{b_1}, l_{b_2} \dots l_{b_n}\right)\right) \quad (7)$$

For optimal sequence of procedure p_k functional

$$F(l_{b_1} \dots l_{i_k} \dots l_{b_n}) = F(l_{b_1}, l_{b_2} \dots l_{b_n}),$$

so error (7) is zero for the optimal sequence and could be zero or positive for the other sequences.

The main idea is to allocate on each step an area with new sequence assignment that would give a good functional value decrease comparing to the current. Which in terms of calculated errors would mean minimizing the summary error.

The clusters construction:

- Start.
- Assign the default sequence for each procedure. Calculate sum error W for all procedures.
- Repeat:
  - Choose not marked procedure p with maximum current error and the optimal sequence l_{p_k}.
  - Calculate the distances to all characteristics borders. Calculate sum error for all space with l_{p_k}.
  - Define it as a current cluster.
  - Repeat for each characteristic:
    - Repeat until cluster size $\geq q$ and the calculated error decrease: with coefficient $t_1 < 1.0$ decrease the distance to one of the borders of the cluster
    - Repeat until the calculated error decrease: with coefficient $t_2 < 1.0$ increase the distance to one of the borders of the cluster
  - Accept the cluster if it decreases error by $dW \geq t_3 * W$. Mark the starting procedure with the flag.
- End.

The constructed areas are $q - dimesional$ rectangles and could intersect. To choose the sequence for a procedure with the set of constructed cluster borders we take the sequence that corresponds with the last cluster that procedure belongs to. Parameters t_1, t_2, t_3 are heuristically chosen so borders movement would capture enough procedures to get more precise direction of error change.

Classes' construction can be started with any sequence; in proposed algorithm the default sequence was chosen because it is optimal on average. Also was made an attempt to start cluster construction with all procedures and choose the one that gives the highest minimization of functional value.

The received clusters with both attempts are very similar, though the last one is much more time-consuming. The other variant that was tested is the binary search of boundaries. This gave also a close result, and this mechanism could be assumed preferable because of no border parameters need.

The possible weakness of proposed classification is the absence of functional monotony by parameter coordinates; this could lead to inaccurate border calculation. Parameters t1, t2 or binary search of boundaries should reduce this effect because in both cases first steps in parameter space are big in terms of considered procedures number thus are statistically proven. One more limitation of constructed classes is that they are $q - dimesional$ rectangles, though with the allowed intersection could actually take other forms. This could perform less accurate area selection but further significantly reduces required time for compiler to compute the proper class for a procedure.

4. Experimental results

The proposed clusterization was implemented in Elbrus compiler. As the training set 9183 procedures of spec2000 benchmark were used. The whole amount of procedures in the given pack is much greater but it was possible to use only the procedures with a measurable execution time. In all cases the clusterization was constructed using full information on execution and compilation time corresponding with each sequence assignment to each procedure, then the solver, that computes procedures characteristics on early compilation stage and chooses the cluster according to calculated borders, was developed in the compiler. The assignment takes place in the end of interprocedural compilation stage, thus the time required for the sequences selection is included in whole task compilation time and is counted in the recieved compilation speedup.

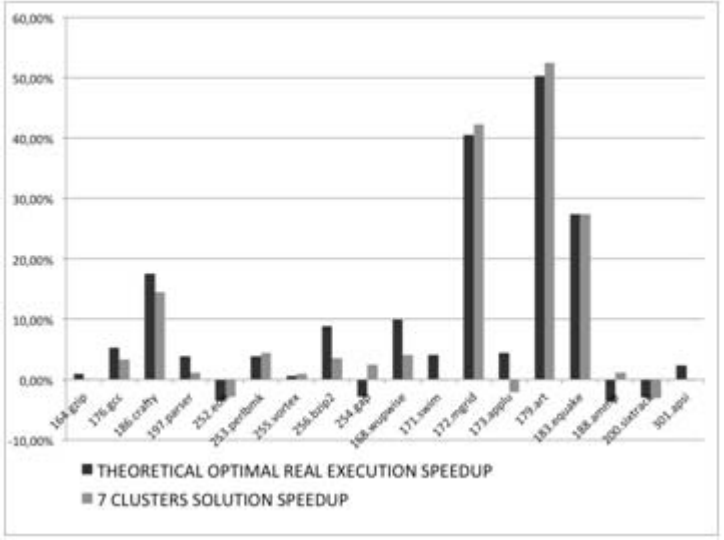

Fig. 1. Optimal and cluster solution, spec2000, 7 clusters.

As was already explained, the effectiveness of sequences assignment depends not on the highest probability of choosing the best line for procedure alone but on

integral characteristic for the whole set. So to show the quality of constructed clusterization it is reasonable to consider all the tasks and not procedures separately. For this purpose results of implementing sequences assigned by optimal and clusterization selections were compared on whole spec2000 benchmark tasks. In this case we used functional that minimizes only performance time(1) and constructed 7 clusters. The result is shown on fig. 1. As it was already discussed in section III "Ideal theoretical solution" the optimal solution for the tasks was combined of optimal theoretical sequence for each procedure. It was noted that because of the memory interaction some tasks, for example, 200.sixtrack, slowed down even with applying this optimal solution. As the result the real measure of optimal solution gained almost 5% less performance increase than it was supposed to be according to theoretical calculations. The same comparison with functional (3) – considering both execution and compilation time yielded worse clusterization results, it occurred mainly because a large amount of procedures are not executed and optimal solution gave much better compilation time results on them.

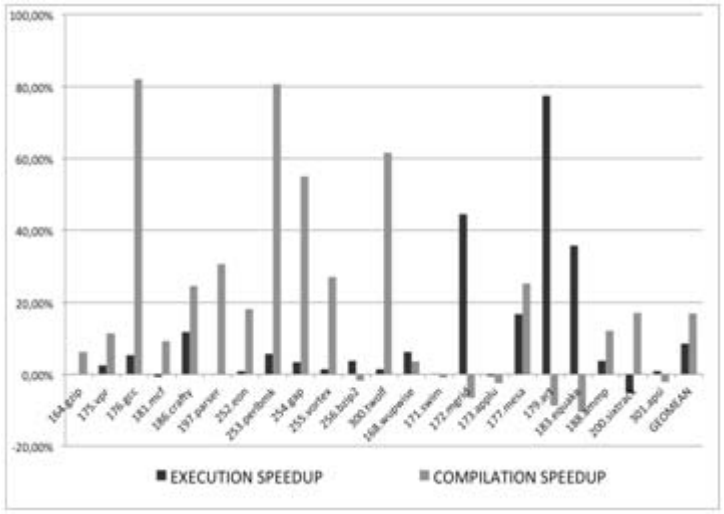

Fig. 2. Spec2000 no train execution, 5 clusters.

When using functional (3) most effect was achieved after constructing first 5 clusters. The corresponding sequence assignment for those clusters reduced compilation time by 17% on average and increased performance by 8.5% on the training set. fig. 2 shows the improvement obtained on certain tasks of spec2000 benchmark. As a test pack for the clusterization spec95 [6] benchmark was used. The execution and compilation result for this pack is shown on fig. 3. The average increase of performance reached 3% and the average compilation time decrease was over 16%.

Measured results prove effectiveness of classification algorithm, though due to the absence of functional coordinate monotony it is not proved that the best possible

solution is received. Another question is the quality of available procedures characteristics choice, which showed to be good enough for the considered set of compilation sequences but could appear not to be representative to make quality selection from different set of sequences.

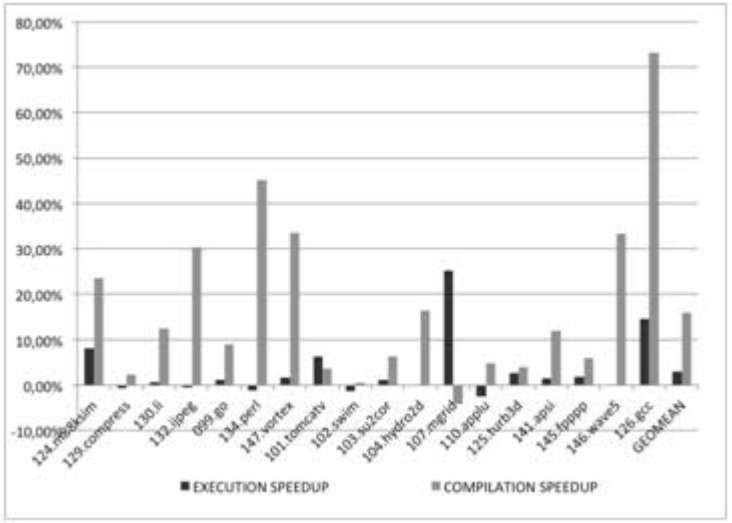

Fig. 3. Spec95 no train execution, out of train set, 5 clusters.

5. Future works

Results presented in experimental section show the possibility of good sequence prediction using classification methods. But some questions should be cleared and researches to be done. First, it could be possible to make hierarchical clustering if inserting some metric that would allow to avoid problems with sporadic points that give inaccurate values for some reasons, this could allow better cluster borders calculation. Another question is how to construct the best training set in sense of avoiding procedures execution interaction. As it can be seen on Figure 1 the execution profiling of the whole task with one sequence can lead to errors in future procedure sequence selection. Also it could be more effective to combine sequences construction with some estimation of future prediction possibility using available procedures characteristics. Finally, there could be done some researches on ascertainment if the found procedure characteristics are good enough to provide maximum possible potential in best classes allocation.

6. Conclusion

This paper introduces problems that come up on the way to develop automatic optimizing sequence selector that provides performance increase and reduces the needed compilation time for each procedure. Necessity of a quality functional on the space of all possible assignment is explained. Also it should be mentioned that such

functional could include any possible limitations besides compilation and execution, in some cases it could be valuable to limit code size increasing or reduce the number of registers that are allowed for code planning. The last limit could be useful to lower register spill fill blocking between the calls and returns from large procedures.

An effective algorithm that can be used to select clusters in the procedures characteristics space is suggested.

The classification methods were implemented in Elbrus compiler. It was shown that a good optimization sequence could be chosen even when it is impossible to execute the code and no train profiling information is available. The results were achieved and introduced using spec2000 and spec95 benchmarks.

References

[1]. Prasad A. Kulkarni, W.Zhao, H.Moon, et al. Finding Effective Optimization Phase Sequence. [A]. Proc. of ACM SIGPLAN 2003 Conference on Languages, Compilers and Tools for Embedded Systems, US: 2003.

[2]. Spyridon Triantafyllis, Manish Vachharajani, Neil Vachharajani, David I. August. Compiler optimization-space exploration. Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, March 23-26, 2003, San Francisco, California.

[3]. Keith D. Cooper, Alexander Grosul, Timothy J. Harvey, Steven Reeves, Devika Subramanian, Linda Torczon, Todd Waterman. ACME: adaptive compilation made efficient. LCTES '05 Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, Pages 69 – 77

[4]. Prasad A. Kulkarni, David B. Whalley, Gary S. Tyson. Evaluating Heuristic Optimization Phase Order Search Algorithms. Proceedings of the International Symposium on Code Generation and Optimization, p.157-169, March 11-14, 2007

[5]. Prasad A. Kulkarni, Michael R. Jantz, David B. Whalley. Improving both the performance benefits and speed of optimization phase sequence searches. LCTES'10 Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems, April 2010

[6]. Standard Performance Evaluation Corporation, http://www.spec.org/

[7]. Suresh Purini, Lakshya Jain. Finding good optimization sequences covering program space. Transactions on Architecture and Code Optimization (TACO), January 2013.

[8]. M. Haneda, P. M. W. Knijnenburg, H. A. G. Wijshoff. Generating new general compiler optimization settings. Proceedings of the 19th annual international conference on Supercomputing, June 20-22, 2005, Cambridge, Massachusetts

[9]. Jain, Murty and Flynn. Data Clustering: A Review. ACM Comp. Surv., 1999.

[10]. Judea Pearl, Stuart Russell. Bayesian Networks. UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.

Классификация процедур для выбора стратегии оптимизации

Ольга Четверина <chetverina_o@mcst.ru>
ЗАО МЦСТ, Ленинский проспект, 51,
Москва, 119991, Россия

Аннотация. Оптимизирующие компиляторы вносят существенный вклад в повышение производительности современных вычислительных систем. Наиболее чувствительными к качеству компиляции являются процессоры с VLIW архитектурой, поскольку в этом случае производительность обеспечивается за счет одновременного исполнения в одном такте нескольких статически спланированных команд, это приводит к усложнению VLIW компиляторов. Так, компилятор для семейства процессоров Эльбрус в режиме –O3 выполняет последовательно более 300 оптимизирующих фаз. Такое количество этапов необходимо для достижения требуемой производительности итогового кода, но является затратным по времени компиляции. Значительное увеличение времени компиляции при высокоуровневой оптимизации в первую очередь вызвано применением ряда агрессивных необратимых преобразований, приводящих к также нежелательному росту итогового кода. Кроме того, остается проблема использования некоторых полезных только для отдельных контекстов оптимизаций. Для одновременного учета требований повышения производительности, уменьшения времени компиляции и размера итогового кода имеет смысл выбрать подходящую оптимизирующую последовательность на раннем этапе компиляции в зависимости от специфических характеристик процедуры. В представленной статье обсуждается проблема классификации процедур для осуществления такого выбора и предлагается ряд способов ее решения.

Ключевые слова: optimizing compiler; optimizing phases sequence; performance tuning; reducing compilation time; procedures classification.

Список литературы

[1]. Prasad A. Kulkarni, W.Zhao, H.Moon, et al. Finding Effective Optimization Phase Sequence. [A]. Proc. of ACM SIGPLAN 2003 Conference on Languages, Compilers and Tools for Embedded Systems, US: 2003.

[2]. Spyridon Triantafyllis, Manish Vachharajani, Neil Vachharajani, David I. August. Compiler optimization-space exploration. Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, March 23-26, 2003, San Francisco, California.

[3]. Keith D. Cooper, Alexander Grosul, Timothy J. Harvey, Steven Reeves, Devika Subramanian, Linda Torczon, Todd Waterman. ACME: adaptive compilation made efficient. LCTES '05 Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, Pages 69 – 77

[4]. Prasad A. Kulkarni, David B. Whalley, Gary S. Tyson. Evaluating Heuristic Optimization Phase Order Search Algorithms. Proceedings of the International Symposium on Code Generation and Optimization, p.157-169, March 11-14, 2007

[5]. Prasad A. Kulkarni, Michael R. Jantz, David B. Whalley. Improving both the performance benefits and speed of optimization phase sequence searches. LCTES'10 Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems, April 2010

[6]. Standard Performance Evaluation Corporation, http://www.spec.org/

[7]. Suresh Purini, Lakshya Jain. Finding good optimization sequences covering program space. Transactions on Architecture and Code Optimization (TACO), January 2013.

[8]. M. Haneda, P. M. W. Knijnenburg, H. A. G. Wijshoff. Generating new general compiler optimization settings. Proceedings of the 19th annual international conference on Supercomputing, June 20-22, 2005, Cambridge, Massachusetts

[9]. Jain, Murty and Flynn. Data Clustering: A Review. ACM Comp. Surv., 1999.

[10]. Judea Pearl, Stuart Russell. Bayesian Networks. UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.