A NOVEL APPROACH FOR SELECTION OF BEST SET OF OPTIMIZATION FUNCTIONS FOR A BENCHMARK APPLICATION USING AN EFFECTIVE STRATEGY

J.Andrews
Research scholar, Department of Computer Science Engineering
Sathyabama University, Chennai-600119, India
andrews_593@yahoo.com

Dr.T.Sasikala
Principal
SRR Engineering College, Padur, Chennai, India
sasi_madhu2k2@yahoo.co.in

Abstract

Finding right set of optimization techniques for a given application involves lot of complications. The compiler optimization technique for a given platform depends upon the various factors such as hardware settings and problem domain as well as orderings. Recent version of GCC compiler consists of more number of optimization techniques. Applying all these techniques to a given application is not feasible, because of program performance degradation. Searching best set of optimal techniques as well as orderings for an application is an extremely critical and challenging task. Many previous works tries to reduce the search space, but such approaches take more time and also expensive. Previously machine learning algorithm has been used to predict best set of sequences, but it requires longer training phase and more data sets. In this paper we have proposed an efficient orchestration algorithm such as optimality random search and advanced combined elimination, which selects optimal set from more than 100 techniques. Result shows that advanced combined elimination works well for most of the benchmark applications than optimality random search.

***Keywords*: Optimization; Optimality Random search; Benchmark Applications.**

1. Introduction and Related Work

Compiler goes through different phases before code generation. Every time when running the program the compiler behaves in an unpredictable manner. Some times it may change the functionality of the code, so preserving the meaning of the program is an essential task. Choosing right set of fine tuned optimizations for a given benchmark an application requires depth understanding of system architecture and domain knowledge. Lack of insufficient data and lack of knowledge in problem domain can severely degrade the program performance while optimizing. Recent version of GCC compiler consists of more than 100 techniques using levels -o1 to –o3.o3 is the highest level of optimization techniques. These optimization techniques applied to each and every benchmark applications and the execution time is measured, by applying all these techniques to a given application as many have observed [11-14] the compiler behaves in an unpredictable manner. In order to increase the execution speed up and to search the best set of optimal ordering an effective orchestration algorithm proposed in this paper.

Many researchers have been investigated iterative compilation [6-10].It requires more number of iterations to find an optimal result. Milepost GCC compiler [16] uses static program features. Milepost compiler requires more excessive compilation and executions. The recent research shows applications using dynamic program features outperform than applications using static program features. Even though iterative compilation works well for most of the benchmark applications, it takes more number of iterations to fine tune the program performance. Many researchers have been focused different optimization algorithms [1-5] to find best set of optimization options for a given applications. Z.Pan et all introduce a new algorithm batch elimination [13] which finds best set of optimization techniques and removing the techniques which gives negative relative improvement percentage value. Batch elimination gives the complexity of O (n^2). Z.Pan et all introduce a algorithm called as iterative elimination which iteratively eliminates one techniques which gives negative RIP value. It gives the complexity of O (n^2). Z.Pan et all introduce a algorithm called as combined elimination which is the combination of batch elimination and iterative elimination. Z.Pan et all shows that combined elimination which outperforms other orchestration algorithms. Combined elimination which gives the complexity of O (n^2). Eunjung park et all [17] tested with various prediction models to find best set of optimization sequences for a benchmark applications. Even though tournament predictor model predicts best set of optimization sequences it requires more training and data sets.

In this paper we have proposed new algorithms such as optimality random search and advanced combined elimination to prune the search space with effective manner. Dynamic program features collected using PAPI. The results are compared with advanced combined elimination. Testing various benchmark applications GCC compiler [19] was selected since it supports different architectures. The paper is organized as follows. Section 2 describes overview of system architecture. Section 3 describes proposed methodology. Section 4 describes experimental set up. Section 5 describes results and discussions followed by conclusion and future scope.

2. System Architecture

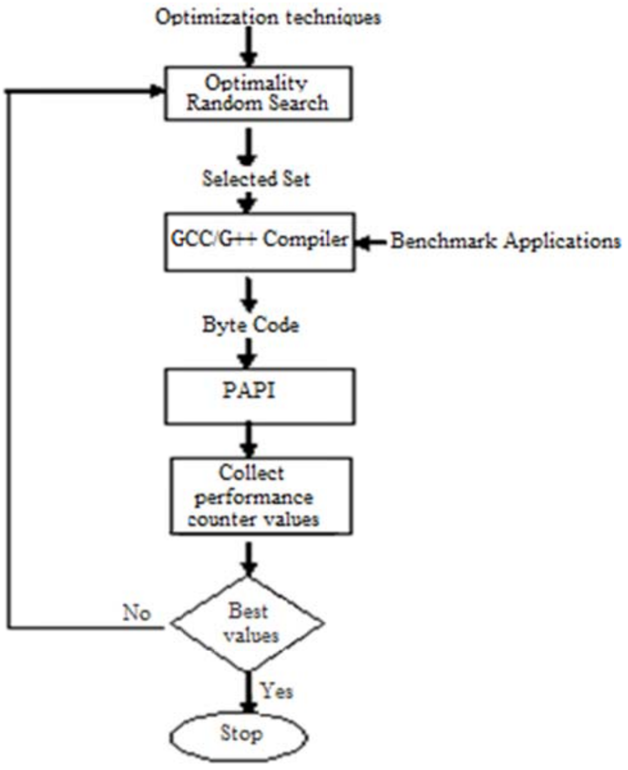

Fig.1. System Architecture

Fig.1 shows over all description of a system. The optimality random search algorithm selects random set of optimizations from an available number of ‘n’ optimization techniques. This random sequence is compiled with gcc or g++ compiler for a benchmark application and produces byte code. Dynamic program features collected using PAPI for a benchmark application. The list of performance counter values listed in Section 3.The given benchmark application, for a selected random sequence whether best values of performance counter are found, if found then the selected sequence gives best optimal set. Otherwise select the next sequence and repeat the steps until best values of performance counter found.

2.1 PAPI Interface

List of dynamic program features collected using PAPI interface [18]. Only 42 set of features used out of many behaviours.These features collected by running a benchmark applications using PAPI.

Table 1. Performance counter values

Performance counter name	Description
HW_INT	Hardware Interrupts
RES_STL	Cycles stalled on any resource
STL_ICY	Cycles with no instruction issue
TOT_CYC	Total cycles
TOT_INS	Instructions completed
VEC_INS	Vector/SIMD instructions
FAD_INS FML_INS FP_INS FP_ OPS FPU_ IDL	Floating point Add Instructions Floating point Multiply Instructions Floating point Total Instructions Floating point Total Instructions Floating point Cycle Idle
BR_INS, BR_MSP BR_ TKN	Branch Instructions Conditional Branches Mispredicted Branches not taken
Level I Cache	
DCA,DCH,DCM	Data cacheAccess,Hits,Misses
ICA,ICH,ICM,ICR	Instruction Cache Access,Hits,Misses,Reads
LDM,STM	LoadMisses, Store Misses
Level II Cache	
DCA,DCH,DCM,DCR,DCW	Data cacheAccess,Hits,Misses
ICA,ICH,ICM	Instruction Cache Access,Hits,Misses,Reads
LDM,STM	Load Misses, Store Misses
TCA,TCH,TCM	Total cache access, Hits and Misses
TLB_DM	Data translation look aside buffer misses
TLB_IM	Instruction translation look aside buffer misses
TLB_TL	Total translation look aside buffer misses

System architecture using these dynamic program features outperforms than using static program features. The performance counter values collected for every benchmark applications while running larger data sets. These values are stored in a repository. For a combination with respect to highest level of optimization (-o3 level) techniques speed up is measured.

3. Proposed Algorithms

Given a set of "n" ON-OFF optimization options {F_1, F_2…Fn), find the best combination of flags that minimizes application execution time using optimality random search. The results were compared with advanced combined elimination strategy.

3.1 Optimality random search

Input: Set of optimization techniques

Output: Optimal set

Procedure

1. Find base time (T_B) for each and every benchmark applications. The base time is measured by compiling and executing each and every benchmark applications with respect to –o3 level.

2. Generate 500 random sequences using random number generator strategy.

3. Copy the content of these sequences in a file.

4. Check each and every sequence to avoid bias. Perform validation logic for testing the output,while preserving the meaning of the program is an essential task.

5. Use a System command to compile and execute all these sequences one by one. Measure the execution speed up. The execution speed up is the ratio between base time and the time required to fine tune the code.

6. Repeat steps (5) until best execution speed up found for every benchmark applications.

7. Use these speed up sequences to predict best set of optimal orderings.

3.2 Advanced combined elimination

Let S be the set of available optimization options

Let B represents selected compiler options set.

1. Find T_B, by applying all flags are ON.

2. Compile the program with T_B configuration and measure the program performance.

3. Calculate Relative improvement percentage (RIP) for each and every optimization options. Relative improvement percentage is calculated based on finding the time required by applying particular flag ON and OFF with respect to T_B.

4. Store all the values in an array based on ascending order .i.e. the most negative RIP is stored in first position of the array.

5. Remove the first two most negative RIP’s from an array instead of one. Now the value of T_B is changed in this step.

6. Remaining values in an array i.e. i vary from 3 to n, Calculate RIP, and store the negative RIP’s in array.

7. If all values in an array represent positive values then set of flags in B represents best set.

Else

8. Repeat steps 2 until B contains only positive values.

9. Stop

3.3 Algorithm Analysis

In the case of random optimality search it requires more number of random sequences (nearly 500).These sequences are generated using pseudo random number generator algorithm. For collecting these set of sequences an algorithm take a complexity of O (n). These sequences are stored in a repository. Compile and execute each and every sequence one by one and measure the execution time and then measure the execution speed up. The sequence which gives speed up <1, those sequences removed from the set. The same step is repeated for all the sequences for each and every benchmark applications. Top predicted sequences for each and every benchmark applications noted. In this algorithm there is no need for calculating relative improvement percentage value. In the case of advanced combined elimination algorithm the time complexity is O (n^2) because of loop for calculating RIP.

4. Experimental setup

We have conducted experiments Intel Core2 Duo T6600 CPU 2.2Ghz.With 3GB DDR2RAM, L1cache 64KB, L2cache 2MB, using ubuntu11.10 operating system, GCC compiler 4.5.2. The list of performance counter values collected for every benchmark applications collected using PAPI hardware counter library. Table 1 represents description of performance counters used in our experiments. For conducting experiments open source GCC compiler used.GCC compiler provides different levels optimization techniques.GCC is currently the only production compiler that supports different architectures and has multiple aggressive optimizations making it a natural vehicle for our research. GCC provides three levels of optimization techniques [19]. To obtain the best performance a user usually applies the highest optimization level –O3. In this level the compiler perform the most extensive code analysis and expects the compiler generated code to deliver the highest performance. We have selected 65 optimization techniques from GCC compiler. Table 2 shows list of important optimization techniques found in GCC compiler [19] version4.5.2.

Table 2.List of optimization techniques

Level-o1 techniques	Level-o2 techniques	Level-o3 techniques
fcprop-registers fdefer-pop fdelayed-branh fguess-branch-probability fip-conversion fip-conversion2 floop-optimize fmerge-constants fomit-frame-pointer ftree-ccp ftree-ch ftree-copy-rename ftree-dce ftree-dominator-opts ftree-dse ftree-fre ftree-lrs ftree-sra ftree-ter	falign-functions falign-jumps falign-loops falign-labels fcaller-saves fcross-jumping fdelete-null-pointer-checks fexpesive-optimizations fforce-mem fgcse fgcse-lm fgcse-sm foptimize-sibling-calls fpeephole2 fregmove freorder-blocks freorder-functions frerun-cse-after-loop frerun-loop-opt fsched-interblock fsched-spec fschedule-insns fschedule-insns2 fstrength-reduce fstrict-aliasing fthread-jumps ftree-pre fweb	fgcse-after-reload finline-functions funswitch-loops

The various mibench [15] benchmark applications used for conducting the experiments. Basic math used for performing various mathematical operations. Bit count used for performing various bit manipulation operations.files.Consumer_tiff2bwconverts an RGB to a grayscale image by combining percentages of the red, green and blue channels. Dijkstra used for computing shortest paths. Patricia data structure used in place of full trees with very sparse leaf nodes. String search used for searching a text in a pattern.

4.1 Performance Metrics used for evaluation

The Relative Improvement is calculated using the following equation

$$RIP(Fi) = (T(Fi = 0) - T(Fi = 1)TB \;\; X100 \qquad (1)$$

Where T (Fi=0) means the time required to find execution time without applying optimization function Fi;

Where T (Fi=1) means the time required to find execution time with applying optimization function Fi;

Where T_B represents base time.

$$TB = T(Fi = 1, Fi + 1 \ldots\ldots Fn) \qquad (2)$$

T_B represents the time required to find execution time by applying all optimization functions for a probe.

$$Execution\ time\ speed\ up = \frac{TB}{fine\ tuned\ code} \qquad (3)$$

Execution time speed up is the ratio between base time and the time required to fine tune the code.

Data set optimal speed up

$$fd = \frac{sd}{optimal\ speed\ up} \qquad (4)$$

Where fd is fraction of optimal data set optimal speed up. It is the ratio between speedup and optimal speed up.

The optimal speed up is measured by picking maximum speed up among 500 combinations of sequences for a benchmark application.

5. Results and discussions

Experiments are conducted for a set of benchmark applications using advanced combined elimination and optimality random search. The results are tabulated in Table 3.

Table 3.Optimal speed up findings

Benchmark applications	Advanced combined elimination	Optimality random search(maximum speed up)
Basicmath	1.24	1.05
Bitcount	1.52	1.01
Consumer_tiff2bw	1.2	1.06
Dijkstra	1.09	1.75
Patricia	1.24	1.05
Stringsearch	1.24	1.4

Table 3 shows comparison between advanced combined elimination and optimality random search. Result shows that for most of the benchmark applications advanced combined elimination gives better speed up than optimality random search. Advanced combined elimination works well for most of the benchmark applications, because of updated value of T_B.T_B is updated for every iteration after elimination of top most two negative RIP value. For finding an optimal set by average more than 50 iterations are required. In the case of optimality random search algorithm a random 500 sequences compiled and executed one by one for each benchmark applications. Speed up and data set optimal speed up is measured using Eq. (3) and Eq. (4) respectively. In this paper we have considered 6 benchmark applications. For these benchmark applications 500 random sequences are applied. So total data point is 6X500= 3000.For a new bench mark applications 3000 data points are given as a training data set along with performance counter values for a benchmark application. The best predicted sequence (i.e. the sequence which gives optimal speed up) for each benchmark applications are measured. The sequence which gives speed up <1 are removed from the set. In the case of basic math nearly 41 sequences are given speed up more than 1.Out of these 41 sequences the sequence which gives maximum speed up is noted. Similarly for other bench mark applications also maximum speed up is noted. In the case of bit count only one sequence gives best speed up, other sequences are removed from the set. Top 10 predicted sequences for each benchmark applications are taken with respect to maximum speed up. These sequences solve the problem of phase ordering. Based on these orderings one can apply optimization techniques for a given benchmark applications. Increasing number of random optimal sequences may increase the performances of optimal speed up. Fig. 2 shows comparison between advanced combined elimination and optimality random search.

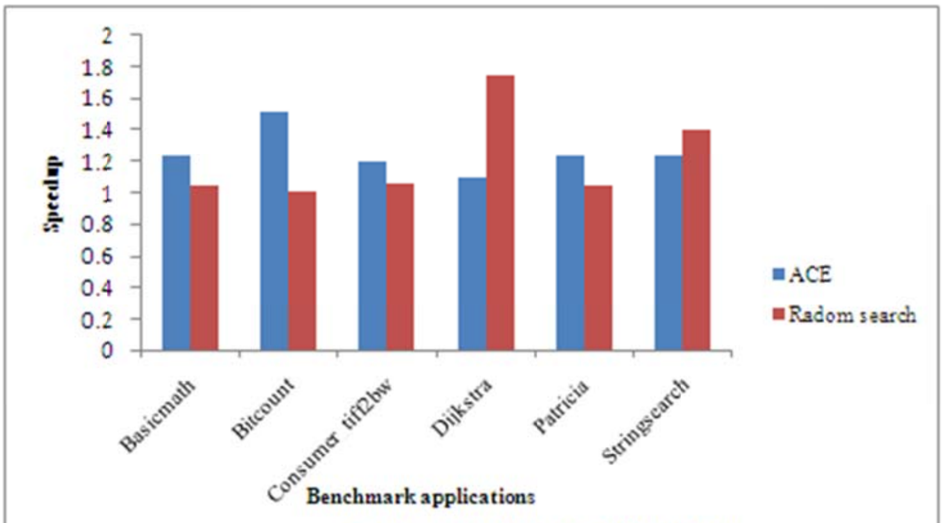

Fig.2.Comparison between ACE and Random search

6. Conclusion and future scope

In this paper we have proposed optimization algorithm such as advanced combined elimination and optimality random search to evaluate its efficiencies to improve execution speed up. Result shows that advanced combined elimination outperforms random search for most of the benchmark applications. For finding optimal set using ACE requires lesser number of iterations when compared to random search. Random search uses fixed value of T_B. While combined elimination uses dynamic value of T_B. In future we consider more benchmark applications and other optimization algorithm. In future we can design effective framework for selecing optimal set by consider other compilers such as ROSE,LLVM and open path compiler.

References

[1] L. Almagor, K. D. Cooper, A. Grosul, T. J. Harvey,S. W. Reeves, D. Subramanian, L. Torczon, andT. Waterman.Findinge_ective compilation sequences.In Proceedings of the Conference on Languages,Compilers, and Tools for Embedded Systems, 2004.

[2] J. Cavazos, G. Fursin, F. V. Agakov, E. V. Bonilla,M. F. P. O'Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In Proceedings of the International Symposium on Code Generation and Optimization,2007.

[3] B. Franke, M. O'Boyle, J. Thomson, and G. Fursin.Probabilistic source-level optimisation of embedded programs. In Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages,Compilers, and Tools for Embedded Systems, 2005

[4] M. Haneda, P. M. W. Knijnenburg, and H. A. G.Wijsho_. Automatic selection of compiler options using non-parametric inferential statistics. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques,2005.

[5] T. Kisuki, P. M. W. Knijnenburg, and M. F. P.O' Boyle. Combined selection of tile sizes and unroll factors using iterative compilation. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2000.

[6] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P.O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In Proceedings of the International Symposium on Code Generation and Optimization [7](CGO), pages 295–305, March 2006

[7] K. D. Cooper, A. Grosul, T. J. Harvey, S. Reeves, D. Subramanian,L. Torczon, and T. Waterman. ACME: adaptive compilation made efficient. In Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES),pages 69–77, July 2005

[8] B. Franke, M. O'Boyle, J. Thomson, and G. Fursin.Probabilistic source-level optimisation of embedded programs. In Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), pages 78–86, July 2005.

[9] P. Kulkarni, S. Hines, J. Hiser, D. Whalley, J. Davidson, and D. Jones.Fast searches for effective optimization phase sequences. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 171–182, June 2004

[10] M. Stephenson, M. Martin, and U. O'Reilly. Meta optimization:Improving compiler heuristics with machine learning. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 77–90, June 2003.

[11] K. Chow and Y. Wu.Feedback-directed selection and characterization of compiler optimizations. In Second Workshopon Feedback Directed Optimizations, Israel, November 1999.

[12] T. Kisuki, P. M.W. Knijnenburg, M. F. P. O'Boyle, F. Bodin, and H. A. G. Wijshoff. A feasibility study in iterative compilation.InInternational Symposium on High PerformanceComputing (ISHPC'99), pages 121–132, 1999.

[13] Z. Pan and R. Eigenmann. Compiler optimization orchestration for peak performance. Technical Report TR-ECE-04-01, School of Electrical and Computer Engineering, Purdue University, 2004

[14] R. P. J. Pinkers, P. M. W. Knijnenburg, M. Haneda, and H. A. G. Wijshoff.Statistical selection of compiler options. In The IEEE Computer Societys 12th Annual InternationalSymposium on Modeling, Analysis, and Simulationof Computer and Telecommunications Systems (MASCOTS'04), pages 494–501, Volendam, The Netherlands,October 2004.

[15] Mathew R.Guthaus,JeffryS.Ringberg et al.(2001),'Mibench:A free commercially representative embedded benchmark suite',Workload characterization,2001.WWC-4.IEEE int.workshop on,pp.3-14.

[16] GrigoriFursin,OliverTemam et al 2011.Milepost GCC:machine learning enabled self tuningcompiler,,International journal on parallel programming,Springer,vol.39,pp.296-327.

[17] Eunjung Park et al.(2011),'An Evaluation of different modeling techniques for iterative compilation',

[18] PAPI: A Portable Interface to Hardware Performance Counters. http://icl.cs.utk.edu/papi.

[19] GCC online documentation http://gcc.gnu.org/onlinedocs/