TuningGenie — an autotuning framework for optimization of parallel applications

P. Ivanenko

Paper proposes software tool for automatic generation of autotuners — special kind of applications to optimize running time of parallel applications in target computing environment. Traditional approach to autotuners creation is amplified by utilizing facilities of rule-based rewriting framework for code transformation purposes. Experimental results show effectiveness of this approach and exhibit ease of use of presented framework.

Introduction

In software development for any field optimization phase is both significant and complicated. This phase is extremely sophisticated and resource-intensive when major aim is to create application which will be efficient in various multiprocessor environments. An autotuning [1], [2] is a modern approach for resolving this issue. Autotuner is separate software which optimizes target software. Optimization typically consists of generating different, but predefined in common sense, variations of optimized software and selecting the most efficient one based on empirical evaluation in target environment. A usually chief criterion of effectiveness is running time. Main advantage of autotuning is in one-time optimization for execution environment — derived variation is most effective until environment configuration remains changeless. Autotuning methodology on contrary to parallel compilers does not require complex source code analysis. It makes possible to create generalized framework which is independent from host programing language, application domain and can be applied for optimizing software for various computing environments - from mobile devices to hybrid clusters. Object of this article is to introduce such framework.

Autotuning — state of the art

There has been solid investigation of search-based autotuning for high-capacity computing. Among them it's worth to mention such well-known systems as ATLAS [3] or FFTW [4]. They utilize autotuning approach to generate high-performance platform-specific libraries in target operational environment. Weakness of this approach is that such solutions are not generic — they propose highly-efficient implementation of most common operations for specific application domain.

Contrary to previous systems Atune-IL [2] proposes an instrumentation language for autotuning which is usable with any programming language, application domain, supports nested parallelism and uses #pragma-based approach to define domain where optimal configuration is searched. In general it allows defining different values for inner program variables that will be probed during instrumentation. For code transformations StringTemplate [5] is used.

Problem statement

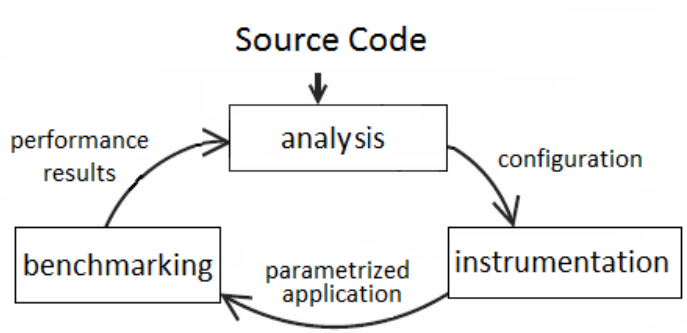
Motivation for this research is idea to create autotuning framework that is independent from programming language and application domain, is able to perform structural code transformations, provides performance monitoring support and allows easily introducing numerical information about target platform (RAM/CPU access speed, basic arithmetic operations execution time, etc.) to parallel programs.

TuningGenie

TuningGenie is autotuning framework that works with source code of application. It uses similar to described in [2] #pragma-based approach to pass expert knowledge from developer to

tuner. It's important to mention that it doesn't automatically parallelizes input application but generates separate version of program for each defined by developer parameters configuration. To achieve desired flexibility in code conversion TuningGenie uses TermWare [6] — rule-based rewriting framework. TermWare transforms source code into term, executes rewriting rules and transforms resulted term back to text representation. Such technology allows not only perform simple substitutions in source code but to enter structural changes in program's algorithms which excels capabilities of existing autotuning frameworks. For instance, it can experiment with data traversal directions (described below pragma *bidirectionalCycle*). It also contains knowledge base to store facts and operate with them in term transformation phase (this feature is used in described below pragma *calculatedValue*). TermWare uses own syntax for rules definition, comes with "out of the box" parsers for Java and FORTRAN language and can be easily extended to add support for other programming languages. In scope of this work probing part of TuningGenie was implemented only for Java applications. It uses custom class loader to reload and benchmark optimized software variations.

In general following diagram describes tuning cycle:



Currently framework contains three main pragmas for tuning configuration definition:

- *tuneAbleParam* — defines range of values for numeric variable. Can be used, for instance, to find size of optimal data decomposition in algorithms that fit "divide-and-conquer" or "geometric decomposition" patterns [7]. An example of resource-critical program which performance considerably depends on granularity of data decomposition is considered in previous work of author "Automatic optimization of meteorological forecasting problem" [8];
- *calculatedValue* — specifies function that will be calculated during instrumentation and variable that will be initialized by this function's result. Allows execute benchmark in target environment and embed empirically-derived data into optimized program;
- *bidirectionalCycle* — this pragma points that direction of cycle iteration does not affect result of calculation and can be changed to inverse. Allows TuningGenie to experiment with data traversal (efficiency of using system caches in particular) and see impact on application's performance;

These three pragmas allow performing optimization for quite extensive class of applications but as a simple demo-example let's consider how TuningGenie can be applied for simple sorting algorithm optimization.

Experiment

As a demo example let's see how parallel sorting algorithm can be tuned. Chosen hybrid algorithm is based on classic parallel implementation of merge-sort. Modification consists in switching to insertion sequential sorting method when size of array becomes less than threshold. Such modification is reasonable since insertion sort is known to be faster on small arrays. Autotuner's task

consists of empirically finding a value of this threshold. The only modification needed to perform it is a single *tuneAbleParam* pragma:

```
//tuneAbleParam name=threshold start=10 stop=500 step=10
int threshold = 10;
.....................................

if (high - low < threshold) {
insertionsort(a, low, high);
      return;
}
int m = (low + high) / 2;
invokeAll(new SortTask(a, tmp, low, m),
new SortTask(a, tmp, m + 1, high));
merge(a, tmp, low, m, high);
.....................................
```

TuningGenie will consequently generate variations of source code for threshold value from [10 - 500] range with step equal to 10, compile and benchmark it. The fastest variation for target environment will be stored to be used in future calculations.

For benchmarking array with 2 million integers was sorted. Configuration of environment:

- Intel® Core™ i5-2410M Processor (3M Cache, up to 2.90 GHz)
- 4 GB DDR2 RAM

Optimal value of threshold was proven to be 120 and this configuration was 23% faster than variation with initial threshold value equal to 10.

Conclusion

Autotuning is a powerful methodology for software optimization which also significantly shortens development time. Taking advantage of rewriting rules technique for code transformation considerably strengthens methodology of automatic software optimization. Presented framework allows abstracting software development away from target environment details while guaranteeing optimality of software's runtime execution. TuningGenie easily provides to applications empirically derived data about running environment and simplifies benchmarking of software. Experiment demonstrated ease of use and effectiveness of introduced framework. In further work functionality for performance data analysis can be added to framework. Usually a lot of configurations are probed during application tuning so structured and visualized information about their impact on performance can be very useful for developers.

References

[1] K. Asanovic et al, "The Landscape of Parallel Computing Research: A View From Berkeley" // Technical Report, University of California, Berkeley, 2006.

[2] Schaefer C.A., Pankratius V., and Tichy W.F., "Atune-IL: An instrumenta- tion language for auto-tuning parallel applications" // Euro-Par '09 Proc. 15th Int.Euro-Par Conf. on Parallel Processing Springer-Verlag Berlin, Heidelberg 2009.

[3] R. Whaley, A. Petitet, and J.J. Dongarra, "Automated empirical optimizations of software and the ATLAS project" // Parallel Computing, 27(1-2), pp. 3-35, Jan. 2001

[4] M. Frigo and S. Johnson, "FFTW: An adaptive software architecture for the FFT" // Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, vol. 3, pp. 1381-1384 vol.3, 1998.

[5] T. Parr. The StringTemplate Homepage. *http* : *//www.stringtemplate.org/*.

[6] TermWare *http* : *//www.gradsoft.ua/products/termware_rus.html*
[7] T. Mattson, B. Sanders, B. Massingill, "Patterns for Parallel Programming" // Addison-Wesley Professional, Reading, MA, 2004.
[8] P.A. Ivanenko, A.Y. Doroshenko, "Automatic optimization of meteorological forecasting problem" // Programming problems. - 2012. - N 2-3. pp. 426-434.

Authors

Pavlo Andriiovych Ivanenko — junior researcher, Institute of Software Systems of National Academy of Sciences of Ukraine, Kiev, Ukraine; E-mail: *paiv@ukr.net*