# Auto-Tuning Support for Manycore Applications - Perspectives for Operating Systems and Compilers

Thomas Karcher
University of Karlsruhe, IPD
76131 Karlsruhe, Germany
karcher@ipd.uka.de

Christoph Schaefer
University of Karlsruhe, IPD
76131 Karlsruhe, Germany
cschaefer@ipd.uka.de

Victor Pankratius
University of Karlsruhe, IPD
76131 Karlsruhe, Germany
pankratius@ipd.uka.de

## 1. INTRODUCTION

Due to stagnating processor clock rates, parallelism will be the source for future performance improvements. Despite the growing complexity, users now demand for better performance of general-purpose parallel software without sacrificing portability and maintainability. This is difficult to achieve, and a coordinated approach is needed for the entire software hierarchy, including operating systems, compilers, and applications.

In particular, performance optimization becomes more difficult because of the growing number of targets to optimize for. With mass markets for multicore systems, the diversity of multicore architectures has increased as well. Their characteristics often differ slightly, e.g., in the number of executable threads, the cache sizes and architectures, or the memory access times and bandwidth. Many parallel applications are optimized at design-time to achieve peak performance on a particular machine, but perform poorly on others. This is unacceptable for applications used in everyday life.

Auto-tuners [1] have great potential to tackle this problem effectively. Instead of being hard-wired in the code, the performance-relevant parameters of a multicore application are made configurable. An auto-tuner is used on the target platform where the program is executed to systematically find an optimal configuration; this is typically not known beforehand and may be counter-intuitive. When the program is migrated to another machine, auto-tuning is repeated, thus preserving portability.

In this paper, we present our novel contributions to make auto-tuning work for general-purpose parallel programs, not just scientific numerical programs. Our experimental results show that auto-tuning is promising and worth integrating into operating systems and compilers, so that manycore applications can be tuned more effectively at run-time.

## 2. AUTO-TUNERS FOR MANYCORE APPLICATIONS

Earlier auto-tuning approaches [1, 2] mainly focus on scientific, numerical programs. For example, tuning parameters are often limited to expressing ranges for loop unrolling or variabilities in matrix multiplication algorithms. Our approach [3, 5] extends the auto-tuning ideas and makes them applicable for general-purpose parallel programs (cf. Fig. 1). The starting point in the tuning cycle is a configurable parallel program that has several tuning parameters (e.g., #threads, #data partitions, partition sizes, #instances of a pipeline stage) and parameter ranges defined by a developer. As an extension of earlier approaches, we also provide code annotation constructs to define architecture variants. Depending on the value of a certain parameter, the auto-tuner has a choice to try out different variants of a program that employ different algorithms to solve the same problem. The auto-tuner calls a search-based optimizer to generate values that would minimize an objective function (based on different heuristics [1, 2, 5]), configures and executes the application, and monitors its performance. After termination, new parameter values are generated based on monitoring results, and the entire process is repeated until some termination condition is satisfied. The best parameter values are then employed throughout the productive use of the application. In contrast to most compiler optimization techniques, auto-tuners dynamically execute an instance of the parallel program and use run-time information to calculate new tuning parameter values. Alternatively, the auto-tuning cycle can be realized as an online process that measures and adapts parameters during the execution of a long-running parallel application.

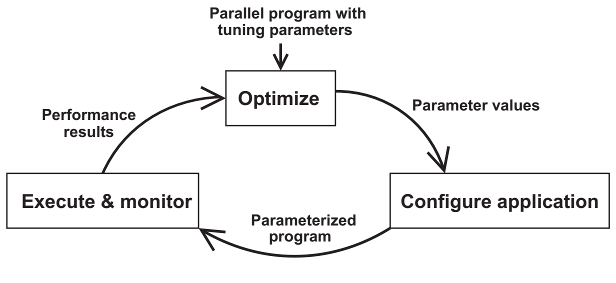

**Figure 1: Auto-Tuning Cycle**

Extending operating systems and compilers with auto-tuning capabilities is far from trivial and requires significant effort. To reduce risk, we first evaluated for multicore programs how effective auto-tuners are as a library and as a stand-alone application. Our feasibility studies [3, 5] show promising results that are presented next. Thereafter, we present our proposals for the integration of auto-tuners into operating systems and compilers.

### 2.1 Auto-Tuners as Libraries

As a first proof of concept, we implemented an auto-tuner as a library in C# and parallelized a commercial biological data analysis application with over one hundred thousand lines of code [3]. The value assignment for tunable variables as well as run-time measurements were handled by library calls. Without modifying the algorithmic internals, we achieved just by coarse-grained parallelization and auto-tuning a speedup of 2.9 on an eight-core machine (2x Intel Quadcore at 1.83 GHz, 8GB RAM, Windows 2003). The difference of execution times obtained with the best and worst parameter values was 40%, and traditional compiler optimization techniques did not help much. Although the library approach works, it is highly language-specific and the program does not execute without the library.

## 2.2 Auto-Tuners as Stand-Alone Applications

With an auto-tuner as an application in its own right, all optimization strategies are implemented just once. However, techniques are required to interact with external applications in a language-independent way; from the spectrum of realization strategies, we explored two extremes. We implemented the Atune-IL [5] instrumentation language that includes constructs to provide an auto-tuner with meta-information to reduce the search space. Developers insert language-independent annotations into source code, to mark for example tunable variables or independent tuning blocks. A pre-processor extracts this information before compilation and communicates it to the auto-tuner. For our biological data analysis application, Atune-IL helped reduce the search space by over 90%, saving many unnecessary cross product operations on tuning parameters [5]. In a different approach, applications can be designed to accept tuning values from the command line, without requiring re-compilation; however, getting feedback with run-time information back to the auto-tuner is more complex, but offers interesting research opportunities for binary instrumentation.

## 2.3 Auto-Tuners as a Part of Operating Systems

Integrating an auto-tuner into an operating system is not an easy task, but offers several advantages compared to the previous approaches. One advantage is that the auto-tuner needs to be implemented just once for the entire system. Using a standardized interface, it is our vision that all applications will be auto-tuned by default.

To asses a short-term integration solution, we looked at the Linux operating system. In particular, it is possible to modify the program loader (*/lib/ld-linux.so*) to transparently auto-tune an application before its productive execution. Tuning parameters can be directly modified within the program's binary file. This approach looked promising, however, we only concentrated on rudimentary value changes during the initialization of the program. Further research is needed to explore the details for run-time variable modifications and performance monitoring.

In the long run, more invasive changes to the OS architecture are needed to exploit the full potential of auto-tuning. Auto-tuners as a part of schedulers seem promising for:

**Global optimization.** Auto-tuners as libraries or stand-alone applications focus on a single application, resulting in a partial optimization. By contrast, an OS-integrated auto-tuner should take the whole system into account and optimize globally. Several applications intended to run simultaneously can be tuned at once, considering cross-process workloads, episodes of heavy/light CPU utilization, or different types of applications at once.

**Dynamic performance adaptation.** The traditional approach to optimize performance mainly during development is far too rigid for the manycore world. We propose that parallel applications are designed with explicit tuning parameters by default, which are passed on to the OS. Auto-tuning should be realized as an ongoing process controlled by the OS. With this extension, OS scheduling will become more invasive, but also more effective as it reaches deeper into parallel applications. Thus, future OS schedulers will have to be substantially rewritten.

Furthermore, drastic changes of future OS architectures may additionally simplify optimization and improve performance for manycore systems. Assuming there will be "enough" cores, the OS may assign disjoint subsets exclusively to one or more applications, and auto-tune each set independently.

## 2.4 Compiler Extensions for Auto-Tuning

The instrumentation process described earlier can be simplified if compilers transparently insert tuning meta-information into the binary executables of manycore programs. Programming language extensions should be considered to make developer knowledge available to auto-tuners. New opportunities are created for coarse-grained optimizations on higher abstraction levels, complementing traditional instruction level optimizations. Earlier studies [3, 4] have shown that parallelism on higher abstraction layers is needed to achieve good performance while instruction-level optimizations might contribute just small speedup improvements. We explored the principles of compiler extensions by analyzing the *GNU compiler collection (gcc)*. It would be feasible to use Atune-IL [5] annotations to generate supplemental tuning data that can be compiled into the binary file of a parallel application. Although we cannot go into more details, this approach is promising as well and provides excellent opportunities for further research.

## 3. CONCLUSIONS

Affordable manycore systems push parallelism into applications used in everyday life, but introduce more complexity for users and developers. Auto-tuning extensions to operating systems and compilers become indispensable for hiding complexity, making performance optimization manageable and keeping parallel applications portable. Our feasibility studies show promising results and interesting directions for further research.

**Acknowledgements.** We thank the University of Karlsruhe and the Excellence Initiative for the support of the Young Investigator Group.

## 4. REFERENCES

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

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

[3] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy. Software engineering for multicore systems: an experience report. In *Proc. IWMSE '08*. ACM, 2008.

[4] V. Pankratius, A. Jannesari, and W. F. Tichy. Parallelizing Bzip2. A case study in multicore software engineering. *accepted Sep. 2008, to appear in IEEE Software.*

[5] C. A. Schaefer, V. Pankratius, and W. F. Tichy. Atune-IL: An instrumentation language for auto-tuning parallel applications. *Technical Report, University of Karlsruhe, 2009.*