

# A Tool Environment for Data Locality Optimization on Chip-Multiprocessors

David Kramer, Oliver Mattes, Rainer Buchty, Wolfgang Karl

Institut für Informatik

Universität Karlsruhe

76131 Karlsruhe

Email: {kramer|mattes|buchty|karl}@ira.uka.de

## I. INTRODUCTION

In the past we experienced an automatic increase of single-threaded performance with every new processor generation. This has come to an end: Due to physical limitations, the clock frequency cannot be increased any longer and only minor improvements to the architecture can be made. Fortunately, Moore's Law [6] is still valid, so the number of available transistors per chip doubles every 18 to 24 months. The additional number of transistors can be used to either enlarge the on-chip caches or to integrate more processor cores onto a single die. The first has the advantage of improving the single-threaded performance, the latter improves only the performance of multi-threaded applications.

But a major problem when executing a multi-threaded application on a chip-multiprocessor is supplying the cores with data to fully utilize the functional units. All cores are sharing the memory bandwidth, therefore reducing the effective bandwidth for each core. Due to the memory wall [12] the proper usage of caches is very important for the overall performance of an application. A poor usage of the cache hierarchy can lead to performance degradation in orders of magnitude. The improvement of the cache usage is the major goal of our tool environment.

## II. OVERVIEW

Our toolset is divided into 3 layers: data acquisition, data analysis, and optimization.

### A. Data Acquisition

The data acquisition layer is organized into three main columns of different kinds of capturing data: the simulation with various simulators, monitoring using hardware counters and using instrumentation for generating memory access traces.

1) *Monitoring*: For monitoring, we offer the data profiler *dprof* [9] for Itanium-II-based CMP systems which employs the four performance counters delivered by the Performance Monitoring Unit (PMU) of the Itanium II processor [4] to collect information of different global events like the number of cache misses or executed instructions.

Using debug symbols and a modified memory allocation mechanism, the *dprof* is able to associate individual cache miss events to the corresponding data structure in the source

code and can generate statistics on them. These statistics may cover the whole problem, a single function, or even a single line of code, enabling location of concrete access hotspots.

2) *Instrumentation*: For generating memory access traces we mainly rely on *doctor* [7]. *Doctor* is a tool for instrumenting and parsing x86 assembly code and was developed for the *augmint* simulator [7]. *Doctor* augments every load/store instruction with additional code, gathering status information like instruction or memory address, and delivering this information to a backend like the *augmint* simulator or stores them into a file for generating memory access traces.

3) *Simulation*: In addition to *doctor*, we use various simulators to obtain information which can't be gathered from hardware counters or instrumentation, e.g. simulating a memory hierarchy when optimizing a program for a CMP system which is not yet build.

For simulating CMP systems we use *Augmint* [7] and the full system simulator *Simics* [5]. For cache simulations we use the *SimpleScalar tool set* [1] or our own cache simulator *CastOr* (see II-B2). While the SimpleScalar tool set is a fully functional simulator, *CastOr* models only the cache hierarchy of an arbitrary CMP and uses memory access traces as input.

### B. Data Analysis

To assist developers in the task of optimizing memory performance of CMP systems, we provide tools analyzing the data gathered from hardware counters, memory access traces, or from simulators, to highlight where a problem might occur and where to optimize first.

1) *Stride/Pattern-Recognition*: The first developer assisting tool is a pattern analyzer [10]. Based on a memory access trace, the analyzer applies appropriate algorithms, e.g. the Teiresias algorithm [8] for pattern recognition, to detect regularities among the memory references like access patterns or strides. Besides the access pattern, such as linked references or access strides, push-back distances are generated in a post-processing step. All the information provided by the pattern analyzer can be used to give a developer hints about optimization possibilities or as input for automatic optimization tools [2].

2) *Hit-Miss-Classification – CastOr*: The second tool is a flexible cache simulator named *CastOr* [3]. This simulator is capable of delivering detailed information about cache

activities at runtime, such as cache misses or cacheline invalidations. It models various cache organizations potentially deployed by a CMP machine including arbitrary number of processors, cache levels, cache organization, private or shared caches, different coherence protocols, write strategies, replacement strategies, and a set of prefetching strategies. The main advantage of this simulator is that it is also able to analyze the reason of cache misses, such as conflict/capacity or false sharing. *CastOr* uses memory access traces generated from *doctor* as input. As output it provides detailed profiles of access histograms, cache updates and cache miss classification at data block granularity.

3) *Visualization – YACO and VRI*: We also provide two visualizers, which can be used for visualizing cache access behaviour of shared memory applications or arbitrary information like a memory access histogram. These visualizers are forming the core of our tool infrastructure as they are providing direct feedback to developers.

*YACO* is a dedicated tool for visualizing the cache access behaviour of shared memory applications. Not only it can show several performance bottlenecks, but more specifically, also depict the reason and in some cases even the solution. *YACO* has a number of 2D and 3D graphical views to present the runtime cache access behavior and the memory operations on data structures.

The second tool, *Visualizer for Runtime Information (VRI)*, is for visualizing arbitrary information like memory access traces generated by *doctor*, or conventional histograms. To maximize flexibility, a plug-in concept was integrated into *VRI*. Plug-ins exist for importing several trace formats, for transforming, for processing the imported data, and for visualizing them. Plug-ins can be plugged together to form a complete processing chain. Additional plug-ins can be easily integrated, an API is provided for this purpose.

### C. Optimization

Although the primary goal of this tool environment is to support developers in the task of optimizing data locality of their applications, we also provide components to automatically perform certain kinds of code modification and code transformation for automatic program optimization. Usually this task is done manually by developers and requires some experience to be successful, however developers have to know in which case to apply a certain optimization, e.g. loop unrolling or blocking and in which cases this transformation is not promising.

As a concrete example, we implemented a semi-automatic precompiler that modifies a program written in C and inserts several prefetching instructions to optimize data locality of this program. For this purpose the access patterns and strides detected by the pattern analyzer are used by our tool for guided prefetching *DogP* to calculate suitable positions for prefetching instructions within the assembly code of a program.

These prefetching instructions decrease the reuse distance and therefore improve memory performance. This is especially

important for multicore systems as they use the same memory controller and therefore have to share the memory bandwidth.

### D. Program Development Kit

Our program development kit (PDK) features an interactive graphical user interface enabling the developer to conduct the entire optimization process from within a single environment. The PDK acts as a frontend for the introduced tools and enables controlling them via a simple Graphical User Interface without deep knowledge of their parameters.

As of now, we provide the *EzCO PDK* for this purpose. For better integration into existing frameworks a refined PDK is currently developed to be integrated into the Eclipse Framework, enabling the users to use the tool-chain within the same place as for programming.

## III. CONCLUSION

In this paper we presented our tool environment for performing data locality optimization for chip multiprocessors. It consists of several tools covering data acquisition, data analysis, and optimization. To simplify the optimization process we provide a program development kit which enables the control of the introduced tools from within a uniform graphical user interface. This tool suite was extensively tested and used successfully for application optimization [11].

## REFERENCES

- [1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.
- [2] R. Buchty, J. Tao, and W. Karl. Automatic Data Locality Optimization through Self-Optimization. In H. de Meer and J. P. G. Sterbenz, editors, *Self-Organising Systems: First International Workshop (IWSOS2006), LNCS4124*, pages 187–201. Springer Verlag, Berlin–Heidelberg, September 2006. ISBN 3-540-37658-5.
- [3] K. Hoang, J. Tao, and W. Karl. CMP Cache Architecture and the OpenMP Performance. In *Proceedings of IWOMP 2007 – International Workshop on OpenMP*, Beijing, China, June 2007. Lecture Notes in Computer Science.
- [4] Intel Corporation. *Intel Itanium 2 Processor Reference Manual, For Software Development and Optimization*, May 2004.
- [5] P. S. Magnusson, M. Christensson, J. Eskilsson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.
- [6] G. E. Moore. Cramping more components onto integrated circuits. *Electronics*, Volume 38, Number 8, 19. April 1965.
- [7] A.-T. Nguyen, M. Michael, A. Sharma, and J. Torrella. The augmint multiprocessor simulation toolkit for intel x86 architectures. In *ICCD '96: Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, page 486, Washington, DC, USA, 1996. IEEE Computer Society.
- [8] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: The teiresias algorithm. *Bioinformatics*, 14(1):55–67, 1998.
- [9] J. Tao, T. Gaugler, and W. Karl. A Profiling Tool for Detecting Cache-Critical Data Structures. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par*, volume 4641 of *Lecture Notes in Computer Science*, pages 52–61. Springer, 2007.
- [10] J. Tao, S. Schloissnig, and W. Karl. Analysis of the Spatial and Temporal Locality in Data Accesses. In *Computational Science – ICCS 2006*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, May 2006.
- [11] J. Tao, A. Shahbahrami, B. Juurlink, R. Buchty, W. Karl, and S. Vassiliadis. Optimizing cache performance of the discrete wavelet transform using a visualization tool. *Multimedia, 2007. ISM 2007. Ninth IEEE International Symposium on*, pages 153–160, 10–12 Dec. 2007.
- [12] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, 1995.