OCEANS – Optimising Compilers for Embedded Applications*

Michel Barreteau[1], François Bodin[2], Zbigniew Chamski[4], Henri-Pierre Charles[1], Christine Eisenbeis[5], John Gurd[6], Jan Hoogerbrugge[4], Ping Hu[5], William Jalby[1], Toru Kisuki[3], Peter M.W. Knijnenburg[3], Paul van der Mark[2,3], Andy Nisbet[6], Michael F.P. O'Boyle[7], Erven Rohou[2], André Seznec[2], Elena A. Stöhr[6], Menno Treffers[4], and Harry A.G. Wijshoff[3]

[1] Laboratoire PRiSM, Université de Versailles, 78035 Versailles, France.
[2] IRISA, Campus Universitaire de Beaulieu, 35042 Rennes, France.
[3] LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands.
[4] Philips Research, Information and Software Technology, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands.
[5] INRIA, Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France.
[6] Department of Computer Science, The University, Manchester M13 9PL, U.K.
[7] Division of Informatics, The University, Edinburgh EH9 3JZ, U.K.

Abstract. This paper presents an overview of the activities carried out within the second year of the ESPRIT project OCEANS whose objective is to combine high and low-level optimisation approaches within an iterative framework for compilation. In this paper we discuss our approach to iterative compilation.

1 Introduction

Within the OCEANS project, the consortium intends to design and implement an optimising compiler that utilises aggressive analysis techniques and integrates source-level transformations with low-level, machine dependent optimisations [4, 5]. A major objective is to provide a prototype framework for iterative compilation, where feedback from the low-level is used to guide the selection of a suitable sequence of source-level transformations.

In general, compiler optimizations rely on static analysis, simplified processor and cache models and sometimes profiling information. Static analysis and models are necessarily pessimistic approximations. Profile based analysis produces averages of the observed behaviour of the system for a limited number of benchmarks/input sets. Compiler analysis determines the best parameters for each compiler optimisation separately (e.g., tile size). However, optimizations are not independent in their effect. We observe that the traditional approach to optimization only gives suboptimal results [6]. An alternative approach to optimisation, namely, iterative compilation, has been proposed in the OCEANS

* This research is supported by the ESPRIT IV reactive LTR project OCEANS, under contract No. 22729.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 1171–1175, 1999.
© Springer-Verlag Berlin Heidelberg 1999

project. It consists of searching for a good transformation sequence. This approach has also been adopted by some authors. Whaley and Dongarra [8], and Bilmes et al. [1] describe a system for generation highly optimised versions of BLAS routines by probing the underlying hardware to find optimal values for blocking factors, unroll factors etc. These systems are capable of producing code that is more efficient than the vendor supplied, hand optimised library BLAS routines. Wolf, Maydan and Chen [9] have described a compiler that searches for the best optimisation. This compiler uses a static cost model to evaluate the different optimisations in contrast to the present approach. Bodin et al. [2] search for the best optimisation on the assembly level, taking into consideration both execution times and code size.

In this paper, we present an overview of the work that has been carried out during the second year of the project. An overall description of the system is given in Section 2. In section 3 we discuss approaches to iterative compilation. Finally, in section 4 we present some conclusions and directions of future work.

2 An Overview of the OCEANS Compiler System

The OCEANS [4, 5] compiler is centered around two major components: a high-level restructuring system, MT1, and a low-level system for supporting assembly language transformations and optimisations, SALTO. SALTO is coupled with SEA, a set of classes that provides an abstract view of the assembly code, and tools for software pipelining (PILO) and register allocation (LORA). Their interaction is illustrated in figure 1. The OCEANS compilation process is driven by a global driver which select optimisations at the source-level and the low-level iteratively until a certain level of performance is reached.

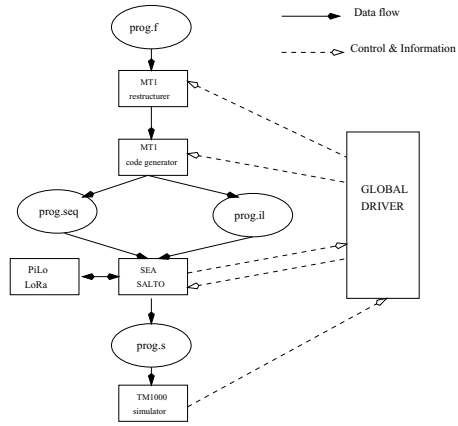

Fig. 1. The Compilation Process.

3 Iterative Compilation

We selected three important kernels and examined their behaviour across seven separate commodity processors and different data sizes: Matrix-Matrix Multiplication, Matrix-Vector Multiplication, and Successive Over Relaxation. We restricted attention to loop unrolling (with unroll factors from 1 to 20) and loop tiling (with tile sizes from 1 to 100). From the results reported in [3, 6] we can conclude that if static techniques are to find the local minima, they need to model program/processor interaction extremely closely. Given the difficulty of statically finding the minima, the next section considers the use of iterative compilation to search through the transformation space.

3.1 High Level Searching

Our compiler searches for the best transformation, by sampling the transformation space and measuring execution times. The algorithm used is grid based: we define an initial grid over the transformation space and refine around good candidate points, that is, points with a low execution time. See [6] for details. In figure 2, we have given the average percentage of how close to the absolute minimum the search algorithm comes across all platforms, benchmarks and data sizes. The x-axis shows the number of evaluations and the y-axis shows the distance to the minimum. The figure shows a monotonic decreasing graph that reaches high levels of optimization rapidly. Hence, we conclude that by visiting only a small fraction of the entire transformation space we can obtain good optimisations.

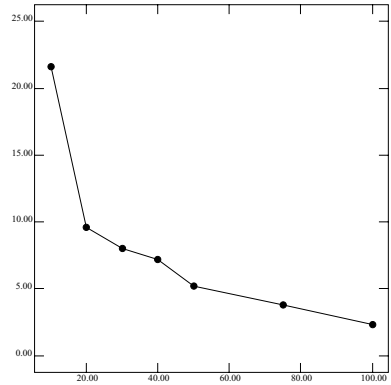

Fig. 2. Average percentage difference minimum

3.2 Alternative Search Techniques

Code Size–Performance Trade-Off An additional constraint of compilers for embedded applications compared to traditional compilers is that code size is important. We therefore not only search for the best optimisation concerning code performance but try to find a trade-off between these two aspects, using a cost model which takes dynamic and static feedback into account.

Genetic Algorithms We are also investigating the application of genetic algorithms (GA) as a means of determining the best transformation sequence. This has the potential benefit of investigating transformation spaces which cannot easily be described as a cartesian domain and is extremely robust in the presence of local minima. The OCEANS GA search is implemented as part of the GAPS compiler framework described in [7].

4 Conclusions and Open Problems

In this paper we have described the activities within the second year of the Esprit project OCEANS. We have addressed the problem of finding the best optimisation for a given processor, program and data size. We have shown that an iterative compilation approach based on a simple search algorithm may be able to find a good optimisation by visiting a relatively small fraction of the entire optimisation space. However, for real applications, the search spaces that need to be considered are extremely large. Hence aggressive pruning strategies need to be developed. The optimisation approach described in this paper is intended to be used for (kernels of) embedded applications. In such cases, long compilation times can be afforded since highly efficient code is required for these systems and compilation times can be amortised over a large number of shipped products.

References

[1] J. Bilmes, K. Asanović, C.W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: A portable, high-performance, ANSI C coding methodology. In *Proc. ICS'97*, pages 340–347, 1997.

[2] F. Bodin, Z. Chamski, C. Eisenbeis, E. Rohou, and A. Seznec. GCDS: A compiler strategy for trading code size against performance in embedded applications. Technical Report 1153, IRISA, Rennes, 1997.

[3] F. Bodin, T. Kisuki, P.M.W. Knijnenburg, M.F.P. O'Boyle, and E. Rohou. Iterative compilation in a non-linear optimisation space. In *Proc. Workshop on Profile and Feedback Directed Compilation*, 1998. Organised in conjuction with PACT'98.

[4] B. Aarts et al. OCEANS: Optimizing compilers for embedded applications. In *Proc. Euro-Par 97, LNCS 1300*, pages 1351–1356, 1997.

[5] M. Barreteau et al. OCEANS: Optimizing compilers for embedded applications. In *Proc. Euro-Par 98, LNCS 1470*, pages 1123–1130, 1998.

[6] T. Kisuki, P.M.W. Knijnenburg, M.F.P. O'Boyle, F. Bodin, and H.A.G. Wijshoff. A feasibility study in iterative compilation. In *Proc. ISHPC'99*, 1999.

[7] A. Nisbet. GAPS: Genetic algorithm optimised parallelization. In *Proc. Workshop on Profile and Feedback Directed Compilation*, 1998. Workshop organised in conjunction with PACT'98.

[8] R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of Alliance 98*, Illinois, US, April 1998. Available through `http://www.netlib.org/atlas/`.

[9] M.E. Wolf, D.E. Maydan, and D.-K. Chen. Combining loop transformations considering caches and scheduling. *Int'l. J. of Parallel Programming*, 26(4):479–503, 1998.