CLaX – A Visualized Compiler

G. Sander*, M. Alt*, C. Ferdinand*, R. Wilhelm
(sander|alt|ferdi|wilhelm@cs.uni-sb.de)

Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken

Abstract. The CLaX compiler was developed in the project COMPARE as reconfigurable demonstration compiler. Its various optimization phases are visualized and animated by the graph layout tool VCG. Visualization allows to understand and to improve the behavior of the algorithms of the compiler phases. We present a tour through the CLaX compiler and demonstrate the newest extensions of the VCG tool, that help to explore large compiler data structures.

1 Introduction

The Esprit project #5399 COMPARE (COMpiler Generation for PARallel MachinEs) aims at reconfigurable, optimizing compilers for different source-target combinations. A new compiler organization CoSy [1] has been designed and implemented. It is based on a generic data base approach with modularization of compilers into phases, where all phases work concurrently on a common intermediate representation and each phase solves a certain subtask of the compilation.

Before developing production quality compilers the feasibility of CoSy was evaluated by the prototype compiler CLaX [3]. The CLaX compiler was used to demonstrate the benefits of distributed compiler phase supervision. The compiler data structures are visualized and animated. This allows to understand and to improve the behavior of the algorithms of the phases.

The VCG tool [4], also developed in the project COMPARE, was used for the layout of the compiler graphs. This tool is specialized for the requirements of compiler data structures which are usually very large. It provides different folding mechanism to focus on certain aspects of the graphs, and allows to reduce the amount of visible information to the points of interest. It provides browsing facilities like tracing of chains of edges or searching of nodes, unlimited scaling and different views onto the graph. The layout methods are a fast variant of the hierarchical layout algorithm of Sugiyama et al. [6] for directed graphs, and a fast force directed placement for undirected graphs (the spring embedder algorithm of Frick et al. [2]). Both layout algorithms can be controlled in a wide range.

The VCG tool and the CLaX compiler run concurrently while the CLaX compiler controls the VCG tool via an animation interface. The VCG tool produces good drawings and runs reasonably fast even on very large graphs.

* This work was partially supported by the ESPRIT Project #5399 COMPARE

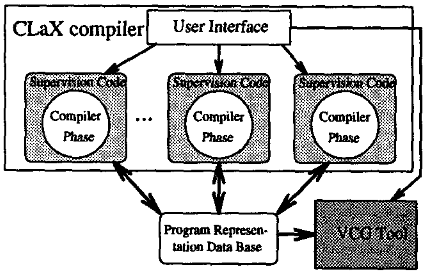

Fig. 1. CLAX Compiler Organization

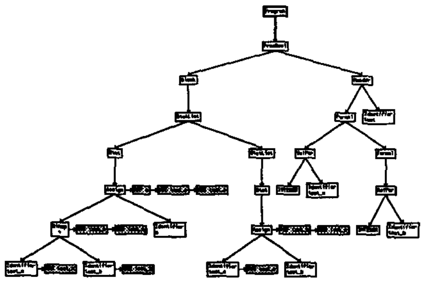

Fig. 2. Annotated Syntax Tree

2 Compiler Phases

In traditional compiler construction, the phases of a compiler are executed in a fixed sequential order. Each phase translates from one intermediate representation of the program into another one. In this sequence, the representations change from source dependent information to target dependent information. Since each phase requires a different format of the representation as input, a reorganization of the compiler or a reuse of a phase is difficult or impossible.

The problem with the traditional model is, that a fixed, optimal order of compiler phases in the middle end does not exist. One optimization may influence another optimization, while the latter might calculate a good starting position for the former. A fixed order of the compiler phase would not produce an optimal result, even if each individual phase would calculate the optimum. Thus, the CLAX compiler works demand driven by the phase dependence graph on a data base of program representations (Fig. 1). This graph identifies which phases are required as precondition of an optimization, and which phases optionally support the optimization. In CoSY, there is no fixed order, but all sequences of phases that do not violate the dependences are allowed. The dependence graph further shows the potential phase parallelism of the compiler. The data base is kept consistent, i.e. if the precondition of a phase is destroyed, the phase is automatically restarted. The supervision code is distributed among the phases. Cyclic execution of optimization is possible, and the progress is visualized by the phase dependence graph.

3 Data Structure Visualization and Animation

The CLAX compiler is demand driven and can be controlled interactively: Sending a request to a data structure results in the generation of this data structure. Sending a optimization request starts all optimization phases that can contribute, until a fixed point is reached. In each step a new instance of the program representation is calculated. The representation is transformed towards their optimum. The compiler uses syntax trees with different annotations (Fig. 2), control flow graphs (Fig. 3), data dependence graphs (Fig. 4), dominator trees, etc. [7]. These graphs are visualized by the VCG tool and can be examined interactively after each step to inspect details. Changes after each phase can be animated automatically. This shows an overview over the compiling process. It is also possible to examine one phase by animating the internal steps.

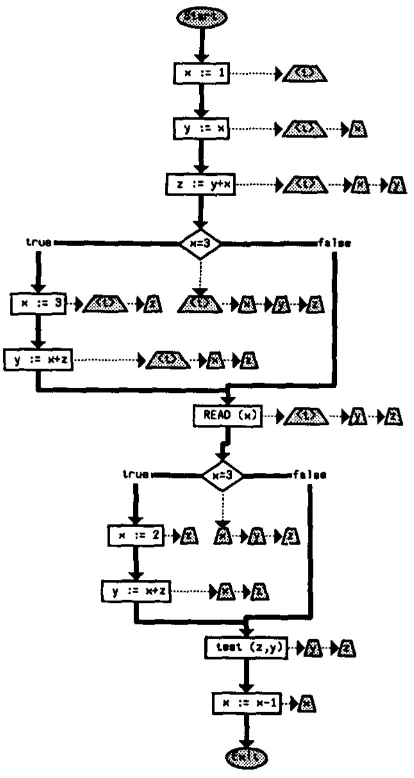

Fig. 3. Annotated Control Flow Graph (Manhattan Layout)

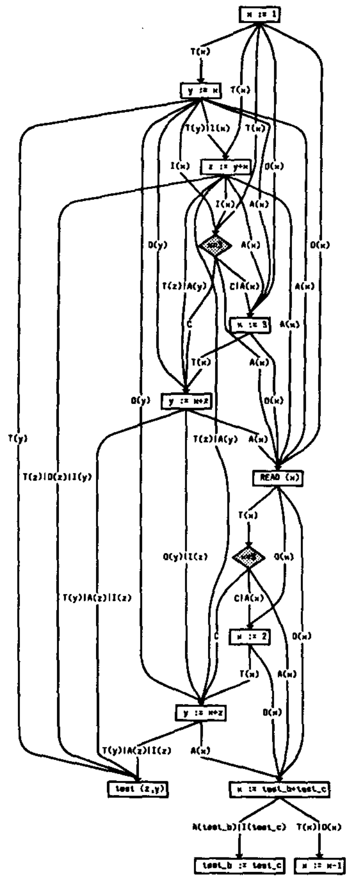

Fig. 4. Data Dependence Graph (Spline Layout)

Each kind of compiler graph needs different visualization methods: Attributed syntax trees are visualized by a specialized tree layout (Fig. 2). It is possible to interactively hide or expose the attributes. Control flow graphs have a uniform appearance if a manhattan layout is used, where all edges consist of horizontal or vertical line segments (Fig. 3). Very large graphs cause the problem that the user may loose the orientation when only a small part of the graph is displayed in the VCG tool window. Thus, fisheye views [5] can be used which distort the graph picture such that the whole graph (or a large part of it) is visible while only the focus point is displayed in all details at the same time. By moving the focus point through the graph, details can be inspected without loosing the orientation (Fig. 5).

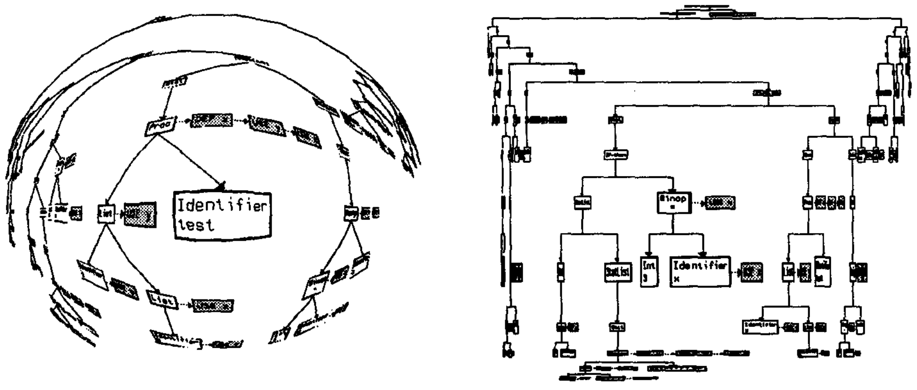

Fig. 5. Syntax Trees, Polar and Cartesian Fisheye View

References

1. Alt, M.; Aßmann, U.; Someren, H.: Compiler Phase Embedding with the CoSy Compiler Model, *in* Fritzson, P.A., editor: Compiler Construction, Proc. 5th International Conference CC'94, Lecture Notes in Computer Science 786, pp. 278-293, Springer-Verlag, 1994
2. Frick, A.; Ludwig, A.; Mehldau, H.: A Fast Adaptive Layout Algorithm for Undirected Graphs, *in* Tamassia, R.; Tollis, I.G., editors: Graph Drawing, Proc. DIMACS International Workshop GD'94, Lecture Notes in Computer Science 894, pp. 388-403, Springer-Verlag, 1995.
3. Müller, T.; Vollmer, J.: Description of the CoSy prototype, COMPARE Technical Report, Rel 1.3, GMD Karlsruhe, 1991
4. Sander, G.: Graph Layout Through the VCG Tool, *in* Tamassia, R.; Tollis, I.G., editors: Graph Drawing, Proc. DIMACS International Workshop GD'94, Lecture Notes in Computer Science 894, pp. 194-205, Springer-Verlag, 1995. The VCG tool is publicly available via http://www.cs.uni-sb.de:80/RW/users/sander/html/gsvcg1.html.
5. Sarkar, M.; Brown, M.H.: Graphical Fisheye Views, Commun. of the ACM 37(12), pp. 74-84, 1994.
6. Sugiyama, K., Tagawa, S., Toda, M.: Methods for Visual Understanding of Hierarchical Systems, IEEE Trans. Sys., Man, and Cybernetics, SMC 11(2), pp. 109-125, 1981.
7. Wilhelm, R.; Maurer, D.: Compiler Design, Addison-Wesley, 1995.