Code Layout in Database Systems - Project Milestone Report

Pedro Artigas and Mengzhi Wang
{artigas,mzwang}@cs.cmu.edu

April 18, 2001

1 Organization

The objective of this report is to communicate the current status of the project, therefore, we will concentrate in changes made to the project specification,in describing the work already done and presenting the planned schedule for the remaining of the project. For more information please refer to the original project proposal. The following sections are Project Changes, describing changes made to the project specification and the evaluation of the obtained results; Compiler Structure, describing the overall structure of the compiler used in this project; Steps Implemented, reporting the status of the required compiler passes; and, finally, the Remaining schedule, presenting the plan to complete this project.

2 Project Changes

During the project implementation it became clear that some of the original goals were unachievable given the limited time period allocated to the project implementation, compiling a full fledged research database engine using a research compiler infrastructure seems beyond reasonable expectations for this project. On the other hand, we argue that we could base the evaluation of the proposed code layout algorithm using benchmarks that reasonably approximate a real, practical, database. The benchmark suite SPEC95 contains one such database, Vortex. The benchmark builds a in-memory database containing a large set of people and performs several operations on this set, simulating the behavior of a full fledged database system.

3 Compiler Structure

We now present the overall structure of our compiler, an augmented version of SUIF coupled using the MachineSUIF back-end(Figure 1). All the analysis passes are implemented in the SUIF1 compiler infrastructure and the code layout algorithm is implemented in the final code layout step. We now present a quick summary of the passes used: The SUIF front-end is used to convert the source code into the SUIF IR. The Control Flow instrumenting pass is used to instrument the code with calls to a profiling library. The Control Flow combining pass is used to combine profiled information back into the SUIF IR representation. The conversion pass modifies the control flow related annotations in

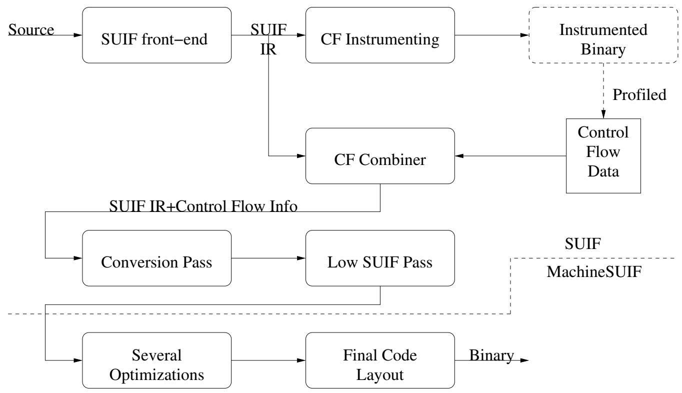

Figure 1: SUIF compiler passes

the SUIF IR in order to ensure that the control flow information is available to MachineSUIF passes. The Low SUIF pass converts the code into the MachineSUIF representation, that representation is fed into several MachineSUIF passes that perform low-level optimizations, the final pass converts this representation into an assembly file, not shown, that can be translated into an object file by any convenient linker.

We now present a more detailed description of the compiler passes related to the code layout algorithm implementation.

3.1 Control Flow Passes

Two passes are involved in obtaining profiled control flow information, the first pass, called Control Flow Instrumenting, is used to instrument the code and produce a binary containing calls to a profiling library responsible to collect control flow information during a profiling run. Several calls are inserted in interesting points in the program control flow in order to capture a complete picture of the control flow behavior of the program under consideration. The information obtained during a profiling run is then fed back into the SUIF IR by the Control Flow combiner pass. All the collected statistics are transferred to the SUIF IR as annotations to the control flow constructs in the source code, such as loops and if statements, that generated such statistics. The combiner pass uses labels added to each such node in the high level abstract syntax tree representation of the program to locate the object that is responsible for each piece of collected information, we observe that objects that generated no profiling data, such as code that was never executed, are not annotated.

The last pass required in order to guarantee that the collected control flow statistics is available to the final code layout algorithm is a conversion pass. Nodes that represent high level constructs in the SUIF IR are not translated into the MachineSUIF representation since MachineSUIF representation is low level, therefore, it does not contain such constructs. The conversion pass is responsible to moving the control flow information into no-op instructions that are added as leaders of each basic block, such instructions have the same representation in both IRs and, therefore, can be used to forward information to the final MachineSUIF code layout pass.

Time	Scheduled task
April 19 - April 25	1. Finish coding the final code layout pass 2. Start the evaluation
April 25 - May 2	1. Finish the evaluation 2. Write the final Project Report

Table 1: Time Schedule

3.2 MachineSUIF Passes

The MachineSUIF back-end contains several low-level optimization passes, such as dead and unreachable code elimination, and register allocation. Those are standard passes; therefore, they are not described in further details. The only MachineSUIF pass of interest for this project is the final Code Layout step. In this step the MachineSUIF representation is converted into a linear list of instruction statements, therefore the final code lay-out is produced. The base algorithm used by MachineSUIF is extremely simple and should be modified in order to produce an instruction cache aware version of the final binary. The algorithm that is used to replace the standard MachineSUIF algorithm is described in more detail in [2, 1] and was presented in the project proposal, therefore we refrain from describing it any further here.

4 Steps Implemented

Several passes used in our research compiler are standard SUIF and MachineSUIF passes, those include all MachineSUIF optimization passes the Low SUIF conversion pass and the SUIF front end, as such, they are already implemented. The control flow instrumenting and combining passes are based on the STAMPede compiler, a research compiler targeting speculative multi-threading architectures, and are already functioning properly. The conversion pass responsible for the adequate forwarding of the control flow information to the MachineSUIF back-end is new, is already completed and functioning correctly. The final code layout pass is derived from the MachineSUIF equivalent pass. The work required to adapt such pass is currently in progress. This steps complete our self-imposed milestone.

5 Remaining Schedule

Table 1 presents the planned schedule for the remaining of the project.

References

[1] B. Calder and D. Grunwald. Reducing branch costs via branch alignment. In *PLDI'94*, 1994.

[2] K. Pettis and R. Hansen. Profile guided code positioning. In *PLDI'90*, 1990.