Code Layout in Database Systems

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

1 Introduction

Processors now run at two orders of magnitude faster than memory. This speed gap forms a new bottleneck in database systems. While it is reasonable that database systems experience a large number of data cache misses, it's surprising to find that the instruction footprint for database systems is also too large for on-chip cache[2, 3].

Code layout is a traditional compiler techniques to improve the instruction cache behavior of the program. By arranging the code in virtual address space, the compiler can make sure that the frequent-accessed code won't get overlapped in the instruction cache. The access pattern of the code may be derived from profiling information.

It is interesting to see if the code layout techniques can solve or alleviate the problem of bad instruction cache behavior in database systems. We would like to explore this in our project. Big wins are possible since database systems waste 60% to 80% percent of the execution time on cache-related stalls and an average of 50% of the stall time is due to instruction cache misses[2].

2 Project Design

Code layout techniques work on two different granularities: procedures and basic blocks. We plan to work on the basic block level in this project. The basic approach is arranging the code layout of the database system using profiling information.

Database Systems. The database under study is Shore, developed at University of Wisconsin. It is a storage manager for database systems and covers 80% of the functionality of the database servers.

Workloads. The typical database workloads are TPC-C or TPC-D benchmarks[1]. TPC-C is a benchmark for online transaction processing, which involves a lot of small transactions concurrently access the database systems. TPC-D is a benchmark for decision support system, which involves a single user asking complex queries. Borg running on Shore provides the tools to run TPC-D workloads on Shore.

Algorithms. The basic greedy algorithm for basic block arrangement will be evaluated. The algorithm requires the Weighted Call Graph(WCG) for the program as input. The graph gives the

March 21 - March 28	1. Compile Shore on Alpha machines. 2. Compile SUIF2 on Alpha machines.
March 29 - April 4	1. Compile Shore using SUIF2.
April 5 - April 11	1. Build the profiling tools using SUIF2.
April 12 - April 18	1. Profiling Shore. 2. Begin to implement the code layout algorithm.
April 19 - April 25	1. Finish coding. 2. Begin to do evaluation.
April 25 - May 2	1. Finish evaluation. 2. Write the report.

Table 1: Time Schedule

frequencies for all the control transfers between basic blocks. This information is obtained through profiling tools.

Compilers. We plan to SUIF2 with Machine SUIF extensions in this project. Shore is written in C++, so SUIF2 is required.

Profiling Tools. We will use SUIF2 to generate instrumented binaries to get the profiling information.

Evaluation. We would like to see how the code layout techniques change the instruction cache behavior of the database systems. The main metric in our evaluation is the instruction cache miss rate.

Resources. We will work on the color alpha machines of ECE department. Machine SUIF works on Alpha. All the softwares are available online.

3 Schedule

The week-by-week schedule is listed in Table 1. Most of the work will be done together. Making Shore pass through SUIF2 is challenging. It will easily consumes more time than we have allocated here. In the worse case, we should have finished the preparation work by April 18th.

4 Previous Work

There are several code layout algorithms in the literature. The greedy algorithm working on the basic block levels was proposed by Pettis and Hansen[5] and Calder and Grunwald[4]. Further improvements upon the greedy algorithms include [7]. However, all these studies were based on Spec benchmarks, which differ a lot from database systems.

Ramirez et al. tried to employ the domain specific knowledge of the database systems in their code layout algorithm[6]. For example, they used the entry point of each operations (joins, scans) as seeds during the greedy searching. They have shown that code layout techniques both reduce the instruction cache miss rate and increase the average instruction fetch width of Postgres running TPC-D.

References

[1] http://www.tpc.org/.

[2] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. Dbmss on a modern processor: Where does time go? In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pages 266–277.

[3] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *ISCA'98, Proceedings of 25th International Symposium on Computer Architecture*, pages 3–14, 1998.

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

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

[6] Alex Ramirez, Josep Ll. Larriba-Pey, and Josep Torrellas. Optimization of instruction fetch for decision support workloads. In *International Conference on Parallel Processing*, 1999.

[7] Cliff Young, David S. Johnson, David R. Karger, and Michael D. Smith. Near-optimal intraprocedural branch alignment. In *PLDI'97*, 1997.