CS-712 Project Proposal
Implementing a Compiler-based I/O Prefetching System for Linux

October 19, 2001

1 Group Members

1. Amit Manjhi, `manjhi+@cs.cmu.edu`, and
2. Mahim Mishra, `mahim+@cs.cmu.edu`.

2 Overview

Compiler based prefetching is an important mechanism for improving the performance of out-of-core applications, i.e., applications which use a data set larger than the memory available to them. Without prefetching, page faults are bound to occur in such applications. Since page-fault handling is a synchronous operation, the application is stalled while the OS services the fault. Prefetching based systems aim to eliminate this page-fault service latency by identifying in advance which data items are likely to be needed soon, and issuing prefetch commands so that no page fault occurs when the application tries to access the data. This identification of which instructions to prefetch can either be statically done by the compiler or a runtime system can implement it dynamically. Since the compiler has the advantage of looking at all the code in its totality on which it can do complex locality analysis and issue **prefetch** instructions in a better way, it is better to use compiler for this purpose. One such compiler based system that has shown tremendous promise, has been designed and implemented by Angela Demke Brown [1] at CMU. Her implementation works on the Hurricane and IRIX OS's, and shows promising performance improvements for applications with large data sets. In this project, we intend to implement her system for the Linux operating systems, and see if we can get the same performance improvements.

The project will involve making significant changes to the Linux kernel. In particular, an entirely new **Prefetch** kernel module will need to be written, and changes will need to be made to the page-fault handling module to make it work correctly with prefetching. Also, a runtime system will have to be implemented that will pass on all the prefetch calls in the application code to the OS prefetch module. The compiler that inserts the prefetch calls itself will not need to be implemented, since we can reuse the compiler that has already been developed.

We believe this project is very well matched with the content of the course, as well as it's general philosophy. It involves studying and understanding a complex, already existing system, and making modifications to it that will significantly (at least, we hope so) improve performance. Also, it will give us plenty of hands-on experience with system-building.

3 Implementation Specifics

The details of the system we are trying to implement can be found in [1]. Although prefetching has been implemented on Linux, the system has not been tested for correctness or performance. Therefore, we are basically looking at re-implementing the whole system from scratch and testing it's performance. The major chunks of work that need to be done are:

1. Testing the correct functioning of the compiler on Linux. This should not be a major problem, but there might be some unexpected surprises.
2. Writing the run-time system, which will pass on prefetch and release requests from the application to the OS module, and also filter out extra requests (based on the memory information that the OS will share with the run time system).
3. Implementing the Linux kernel module, which will perform the prefetch and release operations.
4. Modifying the page-fault routine to help maintain the view of the memory that the run-time system has.
5. Putting in instrumentation calls to study the performance of our benchmarks.
6. Running the benchmarks, analyzing the performance and explaining the differences, if any.

We have not made a complete division of the work that needs to be done between us. However, since the prefetch kernel module is the most complicated component, one of us will probably handle that, while the other will handle other parts of the implementation.

4 Resources needed

We shall need the following things:

1. A machine with high I/O bandwidth for running the I/O intensive applications.

2. Angela's prefetch inserting compiler, which is implemented as a SUIF pass.
3. The SUIF suit to run the compiler, and gcc to compile its output code.
4. Linux source code of the memory management module.
5. *User-mode Linux* to make it easier to implement and debug our system (since we can vary system parameters, and bugs do not crash the machine).
6. The NAS parallel benchmark to test our implementation. This benchmark is used since the data access patterns are easy for the compiler to exploit, and input size can easily be scaled to suit our needs.

References

[1] A. Brown and T. Mowry, "Compiler-Based I/O Prefetching for Out-of-Core Applications", *ACM Transactions on Computer Systems*, Vol. 19, No. 2, May 2001, Pages 111-170.