Adding MPI Parallelization to SUIF

CSC791A Final Project
Todd Gamblin & Prachi Gauriar
http://www4.ncsu.edu/~pgauria/csc791a/

Introduction

For our final project, we intend to modify Stanford's SUIF Parallelizing Compiler framework to generate MPI code. MPI is the de-facto standard for distributed-memory multiprocessing. While SUIF's optimizations are currently designed for shared-memory multiprocessors, we would like to show that with some changes they can be retargeted for distributed memory while maintaining acceptable performance.

The SUIF Compiler Framework

The Stanford University Intermediate Form (SUIF) Compiler is a parallelizing, optimizing compiler. It can take programs in Fortran, C, or Java and compile them into C, Alpha assembler, or x86 assembler code. This code is linked with a runtime library that facilitates communication between processes through shared memory.

In contrast to parallelizing compilers for vector processors, which search for fine-grained parallelism over individual data elements, SUIF's optimizations attempt to find coarse, loop-level parallelism. This is necessary because finer granularity would result in too much communication over the memory bus, slowing down the system.

We believe that SUIF is a good candidate for modification because it is modular. SUIF is extensively documented, and modifications are encapsulated into *passes*. The researcher need only design a pass specifying the transformations to be carried on the SUIF code, and he can focus on his algorithm rather than complicated integration details.

MPI

The majority of today's parallel scientific codes are written using the Message Passing Interface, or MPI. MPI is designed for distributed-memory machines, and rather than referencing common memory locations, processes using MPI specify communication explicitly. MPI provides calls specifying *what* communication is to take place, and the implementation takes care of *how* the communications occur. Typically, messages are transferred between processors over the network.

Because communication must be specified explicitly, distributed-memory interfaces like MPI are generally considered more difficult to program with than shared-memory systems. Compiler automation of distributed-memory parallelization would help to solve this problem.

Parallelizing Loop Optimizations

Hiranandani et al. have addressed the subject of distributed-memory parallelization extensively in their work on the Fortran D compiler [3]. Fortran D is a predecessor to the modern High Performance Fortran (HPF). Hiranandani shows that nested DO loops for a

number of scientific computations can be parallelized through four phases of program analysis: dependence, data decomposition, partitioning, and communication. The parallel loops generated contain send and receive communication calls for distributed-memory environments.

Goumas , et al. [1] have revisited the loop tiling problem, and they have shown that selecting properly sized parallelopiped tiles instead of simple squares can yield optimal performance with respect to communication cost or idle processor time.

Project Plan

We have divided our project plan into the following phases:

1. Select a simple tiling example from either the Haranandani or Goumas paper, and examine it in its optimized and unoptimized form.

2. Become familiar with SUIF's runtime parallel library, and determine how well it maps onto MPI. For now we will go with the most straightforward mapping, without concern for granularity of parallelism.

3. Get our simple tiling example working as a SUIF Pass with the modified runtime library. This may be inefficient due to overly fine-grained parallelism.

4. Become familiar with SUIF's analysis passes, and modify them so that we detect coarser-grained parallelism. Re-run the pass from step 3, and compare performance.

5. Time permitting, extend our passes to include more complicated tiling schemes and Goumas, et al.'s parallelopiped tiles.

Works Cited

1. G. Goumas, N. Drosinos, M. Athanasaki, N. Koziris. **Automatic Parallel Code Generation for Tiled Nested Loops**. In *Proceedings of the 2004 ACM Symposium on Applied Computing.*

2. M.W. Hall, J. Anderson, S. Amarasinghe, B. Murphy, S-W. Liao, E. Bugnion, and M. Lam. **Maximizing Multiprocessor Performance with the SUIF Compiler**. In *Computer* volume 29 number 12.

3. S. Hiranandani, K. Kennedy, C.-W. Tseng. **Compiling Fortran D for MIMD Distributed-Memory Machines**. In *Communications of the ACM: vol 35, number 8*. 1992.