OdinMP/CCp – A portable implementation of OpenMP for C

Christian Brunschen and Mats Brorsson
Department of Information Technology, Lund University, Sweden

The goal of this project has been to write a portable implementation of the OpenMP specification for C. The result is a C-to-C compiler written in Java that takes a C program with OpenMP directives and produces a C program using POSIX threads, or pthreads for short, to implement the parallelism. The compiler, OdinMP/CCp, implements the entire specification for sequentially consistent shared memory multiprocessors using pthreads. For multiprocessors with a relaxed memory model, a slight modification to the compiler is needed for strictly correct behaviour for volatile variables.

We first approached the problem by attempting to develop a mechanical way to manually translate an OpenMP program into an equivalent program using pthreads. In this process, we had to solve a number of conceptual problems – pthreads offer a granularity of parallelism at the level of one function, which is quite radically different from what OpenMP offers. Likewise, we had to investigate, understand and solve such problems as thread initialization, how to implement threadprivate variables, or how to handle the fact that source code can come in more than one file. Also, while the main focus was to investigate the viability of doing this at all, we could not entirely lose sight of performance issues, both regarding memory use and overhead introduced into the code by the translation. Preliminary performance measurements on an SGI Origin 2000 showed that the performance of a manually translated OpenMP program is comparable to that of MIPSPro compiler version 7.2 using SGI Power C instead of OpenMP.

Though performed manually, the resulting scheme for translating was quite mechanic and suitable for implementation in software. Our next task, thus, was to write a compiler for C with OpenMP, which would generate much the same code as we had generated manually. This compiler should also be reasonably portable, and still generate code that would run with good performance.

To write the compiler itself, we chose to perform the development of the compiler in Java, with the help of two compiler-writing tools, Java Tree Builder and Java Compiler Compiler. The result is a working compiler called OdinMP/CCp, which implements almost all of the OpenMP specification. It generates code that offers good performance, and it has the advantages that is can be used on any platform that offers support for POSIX threads.

Performance measurements on the generated code, performed on a Sun Enterprise 10000 with 64 250 MHz Ultra-Sparc processors, have shown a measured speedup of 7.2 for a molecular dynamics program ‘md’ with a dataset of 2048 particles, when executed on 8 processors. The overhead in the generated code, when entering a parallel region, is less than 10 microseconds plus 10 microseconds per thread that needs to be started. The cleanup overhead when leaving a parallel region is similar.

OdinMP/CCp is publicly available with complete source code at the following URL: http://www.it.lth.se/odinmp. It is also currently being used in another project, to convert the SPLASH program suite from ANL macros to OpenMP, in order to compare the two approaches to parallel programming of shared memory architectures.