ompP: A Profiling Tool for OpenMP

Karl Fürlinger and Michael Gerndt

Institut für Informatik,
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München
{Karl.Fuerlinger, Michael.Gerndt}@in.tum.de

Abstract. In this paper we present a simple but useful profiling tool for OpenMP applications similar in spirit to the MPI profiler mpiP [16]. We describe the implementation of our tool and demonstrate its functionality on a number of test applications.

1 Introduction

For developers of scientific and commercial applications it is essential to understand the performance characteristics of their codes in order to take most advantage of the available computing resources. This is especially true for parallel programs, where a programmer additionally has to take issues such as load balancing, synchronization and communication into consideration. Accordingly, a number of tools with varying complexity and power have been developed for the major parallel programming languages and systems.

Generally, tools collect performance data either in the form of traces or profiles. Tracing allows a more detailed analysis as temporal characteristics of the execution is preserved, but it is usually more intrusive and the analysis of the recorded traces can be involved and time-consuming. Profiling, on the other hand, has the advantage of giving a concise overview where time is spent while causing less intrusion.

The best-known tracing solution for MPI is Vampir [12] (now Intel Trace Analyzer [6]) while mpiP [16] is a compact and easy to use MPI profiler. Both Vampir and mpiP rely on the MPI profiling interface that allows the interception and replacement of MPI routines by simply re-linking the user-application with the tracing or profiling library. Unfortunately no similar standardized profiling or performance analysis interface exists for OpenMP yet, making OpenMP performance analysis dependant on platform- and compiler specific mechanisms.

Fortunately, a proposal for a profiling interface for OpenMP is available in the form of the POMP specification and an instrumenter called Opari [10] has been developed that inserts POMP calls around instrumented OpenMP constructs. The authors of POMP and Opari also provide a tracing library, while we have implemented a straightforward POMP-based profiler that is similar in spirit to mpiP and which accordingly we call ompP [13].

* This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under contract GE1635/1-1.
The rest of the paper is organized as follows: In Sect. 2 we describe the design and implementation of our tool and in Sect. 3 we demonstrate its functionality on some example programs. Finally in Sect. 4 we review related work, we conclude and present ideas for future work in Sect. 5.

2 Tool Design and Implementation

In this section we present the design and implementation of our profiling tool ompP.

2.1 Instrumentation

Opari [10] is an OpenMP source-to-source instrumenter for C, C++ and Fortran developed by Mohr et al. that inserts calls to a POMP compliant monitoring library around OpenMP constructs. For each instrumented OpenMP construct Opari creates a region descriptor structure that contains information such as the name of the construct, the source file and the begin and end line numbers. Each POMP_* call passes a pointer to the descriptor of the region being affected. In the example shown in Fig. 1, Opari creates one region descriptor for the parallel region and this descriptor is used for the POMP_Parallel_[fork,join,begin,end] and also for the POMP_Barrier_[Enter,Exit] calls. The barrier is added by Opari in order to measure the load imbalance in the parallel region, similar implicit barriers are added to OpenMP worksharing constructs.

![Fig. 1. Instrumentation added by Opari for the OpenMP parallel construct. The original code is shown in boldface, the square brackets denote the threads that execute a particular POMP_* call. The right part shows the pseudo region nesting used by ompP.](image)

2.2 Performance Data Collection

Our profiler keeps track of counts and inclusive times for the instrumented OpenMP constructs. In order to simplify performance data bookkeeping (the same region descriptor can be used in a multitude of POMP_* calls), each Opari region is broken down into smaller conceptual “pseudo” regions and performance