

OpenMP: what's inside the black box?

Eduard Ayguadé and Lluç Àlvarez

Computer Architecture Department, Universitat Politècnica de Catalunya (UPC-BarcelonaTECH)

Computer Sciences Department, Barcelona Supercomputing Center (BSC-CNS)

Barcelona, Spain

{eduard, lluca}@ac.upc.edu

Abstract—This paper presents the “Implementing a minimal OpenMP runtime using Pthreads” assignment that is offered to students of *Parallel Programming and Architectures (PAP)*, a third-year subject in the Bachelor Degree in Informatics Engineering at the Barcelona School of Informatics (FIB) of the Universitat Politècnica de Catalunya (UPC) - BarcelonaTech. We believe this is a high-quality assignment, previously-tested (and evolved) in the past 3 academic years, that is readily adoptable by other educators with the objectives of 1) understanding how a high-level parallel programming model such as OpenMP is implemented and 2) learning Pthreads in a very interesting and applied way. This assignment contributes to open the black box behind the compilation and execution command lines for OpenMP programs.

Index Terms—Parallel programming, OpenMP, Pthreads, compiler and runtime system

I. INTRODUCTION

OpenMP is the de-facto mature standard API for writing shared-memory parallel applications in C/C++ and Fortran. For the programmer it consists of a set of compiler directives, runtime routines and environment variables. As programming model it is ideally suited for multicore and shared-memory multi-socket architectures, allowing programmers to explicit opportunities for parallel execution in regular (e.g. loop based) and irregular (e.g. recursively traversing dynamically created data structures such as lists, trees, ...) applications.

The two hidden components that support the execution of programs written in OpenMP are the compiler and the runtime system. The OpenMP compiler transforms code with OpenMP directives to explicitly multithreaded code (Fortran/C/C++ with calls to the OpenMP runtime library). The OpenMP runtime system supports the code generated by the compiler, providing support for thread and task management, work dispatch, synchronisation, intrinsic OpenMP functions and internal control variables. Most OpenMP runtimes are implemented using Pthreads (also named POSIX threads), the standardised C language threads programming interface that was specified by the IEEE POSIX 1003.1c standard.

II. WHY DO OPEN THE OPENMP BLACK BOX?

In this scenario, our students usually ask the same questions: “Why do we want to study how is an OpenMP program

actually executed on our parallel machine? Is it not enough to be able to write correct OpenMP programs?, ...”. We believe that as a programmer, this knowledge may help to understand program performance and possible sources of overhead, helping to write more efficient code. But as potential developer to be able to develop a runtime system on top of low-level threading offered by the OS.

The laboratory assignment that we present as a “peachy” assignment consists on building a simplified OpenMP runtime system that gives support to the code generated by the GNU gcc compiler, alternative to the libgomp runtime system that is distributed as part of the gcc distribution. The gcc compiler is used unmodified and students analyze how the compiler translates the different directives to invocations of services in libgomp. The implementation of the runtime is based on the POSIX Pthreads standard library which provides the basic support for thread creation and synchronisation on top of which the work-sharing and tasking models can be built. In this way students learn the main features in the POSIX library. The libgomp library is simply replaced using the LD_PRELOAD mechanism, making use of the library constructor and destructor mechanisms to setup the OpenMP environment necessary to execute the OpenMP programs.

Two alternative itineraries are proposed in this assignment, each one implementing different functionalities of the OpenMP runtime:

- **Itinerary 1 – Basic implementation for the work-sharing model:** parallel regions, work distributors (single and for) and synchronizations (barrier and critical). In the simplified implementation threads are created and finished in each parallel region (i.e. no thread pool), there is no support for nested parallel regions, makes use of synchronization constructs provided by Pthreads (locks, barriers,...) and there is no support for thread binding policies. As part of their work, and based on their own skills and interests, students decide to relax some of these constraints and/or to consider some additional OpenMP features initially not considered.
- **Itinerary 2 – Basic implementation of the tasking model:** unified parallel-single region, task, taskloop, and synchronizations (critical and taskwait). As in the first itinerary there is no support for nested parallel regions nor nested tasks, which can be later relaxed based on student's interest and time.

This work is partially supported by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project and by the Generalitat de Catalunya (contract 2017-SGR-1414).