The OpenMP Source Code Repository

Antonio J. Dorta, Casiano Rodríguez and Francisco de Sande
Dpto. de Estadística, I. O. y Computación
Universidad de La Laguna
c/ Astrofísico F. Sánchez s/n
38271 La Laguna, Tenerife, Spain
fsande@ull.es

Arturo González-Escribano
Dpto. de Informática
Universidad de Valladolid
Spain
arturo@infor.uva.es

Abstract

The OpenMP source code repository (OmpSCR) is an infrastructure that we make available to the OpenMP community. It is based on a set of representative applications and it is supported by a web site.

The aim of this repository is to contribute to the knowledge and spreading of OpenMP that in the recent times has become the standard tool for program developing in shared memory multiprocessor platforms.

The discussions originated by the use of this framework will define the real needs of the users and developers community, and they will lead the future of this project.

1. Introduction

In this work we present OmpSCR, an OpenMP Source Code Repository. This repository is composed by a representative set of OpenMP applications written in C, C++, and Fortran and includes a wide spectrum of parallelizable situations. Our aim is that OmpSCR be free, public, non proprietary and available to use as reference to the OpenMP community.

The repository is not composed only by the applications, but it is supported by an infrastructure that eases tasks such as the compilation and execution of the applications in different platforms. It is provided with a set of support functions used by the codes so that they are unified and facilitate common operations as arguments processing, time measurements, results presentation, etc.

A web infrastructure has been created to give support to the repository, increasing its features. The High Performance Computing (HPC) community will not only be able to obtain the source code of the applications: the web site establishes a bidirectional way by what, the registered members, will be able to publish opinion articles, to propose downloads, to access to the discussion forums and a large etc. What is claimed is that OmpSCR is a meeting point for the OpenMP community that encourage the communication among the users and favor the development and learning of OpenMP. OmpSCR is also registered as project at sourceforge.net [16] and relies on all the possibilities that it offers.

We hope that the HPC community finds interesting the repository and that OmpSCR turns into a useful tool for the community. The proper tour and future of OmpSCR depends on the appraisals and contributions of the community and we do not discard that it could turn into a benchmark.

The remaining of the paper is organized as follows: In section 2 we deepen in the motivations of this work and we outline our goals. Section 3 is dedicated to explain the source codes which are currently included in the repository. In section 4 we show the most relevant characteristics of the web site that supports the repository. Some computational results obtained for different platforms using the programs in the repository are presented in section 5. Finally, section 6 offers conclusions and comments on future work.

2. OmpSCR: the OpenMP source code repository

Usually, when a research team publishes a work containing computational results, the corresponding source codes are not available to the scientific community. Upon request, authors usually provide a copy of the source codes, but it is not always easy to know the details about how a computational experiment has been accomplished. Details like the way in which execution times have been measured, the points in the code where they have been measured, the arrangement of the input data for the application or the size of these data are some of the factors to be considered.

Let consider [19] as an example. Among the computational results presented in this work the authors show the speedup obtained when sorting a $10^6$ elements vector us-
ing the Quicksort algorithm. Reading the information provided in the paper about the experimental framework the reader can know the compiler, the platform, the number of processors involved or the speedups obtained. Nevertheless there are fundamental aspects that do not result clear. Some of these are:

- Is the data type integer or real?
- How do they generate the data?
- How many instances have been sorted?
- Have they taken the mean time or the best time?
- When computing the time, have they considered the time for vector generation or only the sorting time?
- What is the sequential algorithm used to compute the speedup?

All these details are not specified when writing a scientific paper due to space limitations. However, it is crucial to know them to understand a work and to establish its relevance. A similar situation occurs in many other works where computational experiments are involved.

Sometimes the authors overlook this lack of information referring to their technical reports. In these documents the authors can provide extensive details of the work with no space limitation. Nevertheless, the use of technical reports is not always widespread, and in many cases the specific details of the experiments are lost among huge amounts of data or accessory information.

We believe that a first step to know the way in which experiments are performed is to have some mechanism that allows access to the source code. Furthermore, in order to compare results it is very advantageous to have source codes with a similar programming style, well documented, with similar execution interfaces and with an agreed methodology for performance measurement.

This is the main idea behind the creation of OmpSCR: to provide the OpenMP users with an infrastructure that allows both to evaluate the performance of OpenMP codes and to compare the performance of the codes on different platforms and compilers. It should be a framework that simplifies, as far as possible, the process of massive experimentation and results processing. It will rapidly allow to choose representative applications. The results obtained will be easily comparable with those coming from other groups.

With the recent arrival of OpenMP as de facto standard for shared memory parallel systems programming, the need of applications that allow to measure the performance has become clear. If recent works in this field are studied, we find that two different approaches have been used: initially the researchers used a limited set of applications to test their implementations [19]. In the recent time, this lack has been partially covered by the benchmarks developers providing OpenMP versions for existing benchmarks or developing specific new OpenMP benchmarks [8], [2], [14].

The most popular benchmarks in this field (SPEC and NAS) are highly oriented to a pure performance analysis and mostly they pay attention to numerical applications and to simple parallelization of loops. It is difficult to find in these benchmarks, for example, applications that use multilevel parallelism [6], a characteristic of the OpenMP standard that has been widely discussed in OpenMP conferences. This is the motivation for many researchers that still continue using specific applications that make evident situations usually not considered in the classic benchmarks.

One of the goals of the OmpSCR is to standardize a set of applications to be an alternative to these classic benchmarks. The characteristics that the OmpSCR applications should fulfill are:

- They should be coded using ANSI C/C++ or standard Fortran90/95.
- Both kernel applications and complete applications can be considered to be included in the repository.
- In any case they should be programs reasonably simple (up to 5000 code lines). This will ease their use by non-expert OpenMP users.
- Applications will be representative of the real world, or applications that make evident situations that require special treatment when programmed using OpenMP.
- They should be codes extensively tested and validated in a wide variety of systems.
- The code must be written according to quality standards.
- The applications must be well documented.
- It is a desirable property that the application be self-checking (containing code to test the correctness of its results).
- The programs must be freely distributed, and not affected by copyrights.

The approach that we plan to follow in the development of the OmpSCR is collaborative and incremental. We hope that different research groups contribute their own codes to be added to the repository, as far as we know that many groups use codes compliant with the above characteristics. We think that the repository must be open, not limiting the number of applications available on it. Any contribution can be incorporated if it fits the former requisites.

Initially we do not plan the creation of an OpenMP benchmark, but we do not discard this future evolution.
3. Applications in OmpSCR

The number of available applications to be downloaded in OmpSCR is of 12. Among them they you can find applications in several versions for different languages some of them provided by the OmpSCR team and other contributed by other authors. The number of programs available will clearly increase depending on the interest raised on the the OpenMP users and developers community.

Although most of the applications in the repository are coded using C, the OmpSCR is not oriented to a specific programming language. One of the aims of OmpSCR is to make available the same code written in different languages to check the behavior of the corresponding compilers.

The kind of applications included in OmpSCR goes from very simple examples of OpenMP parallelizations to complex codes with several parallel loops. Some of the programs constitute examples for different parallelizations that are not usual or that can not be easily implemented using OpenMP.

It is also our goal to include applications with specific interest for OpenMP compiler designers. For instance, some applications requiring multilevel parallelism have been included. Also there is an example of parallel graph search using a master-slave paradigm. OmpSCR is also thought as a repository of examples useful for learning how to properly use OpenMP. Thus, some of the programs are provided in several versions using different parallelization strategies or primitives.

For simple codes showing different ways of using OpenMP primitives it is even possible to include in the repository bad-parallelized examples. For instance, we have included some examples of loops with carried dependences. The bad solutions degenerate in sequential codes or produce inconsistent results. These implementations are clearly marked as wrong in the distribution.

OmpSCR is distributed as a whole in a tarball archive. It contains a directory structure which may be installed in a local account. The distribution contains simple installation instructions, a guide for new applications developers and a FAQ file.

Inside the distribution, the directory applications contains specific subdirectories for each program or set of versions. In the download section of the OmpSCR web site, the applications are also available individually to be installed under the applications directory.

The only platform dependent information needed to compile the OmpSCR applications is the name of your OpenMP compilers front-ends and the appropriate command-line options for them. The final users must configure these details editing a configuration file or using an interactive script. Templates with usual options for common compilers and development platforms are also provided in the distribution.

To avoid vendor-specific make utility details the applications building is controlled with GNU make. In each application directory, its developer provides a very simple GNU makefile. A template for this file is also provided in the distribution to be adapted by the new application developers. Usually, they only need to include the target binary name in a variable. At compile time, this standard GNU makefile includes a common file with all the implicit rules needed to build C, C++ or Fortran applications, using the compiler details provided during the user’s configuration. Thus, an application may be rebuild alone from its own directory, or with the whole distribution.

The programs written using each programming language are totally independent. If there is only one compiler definition, the compilation rules simply ignore the source code files for which there is no compiler definition. The binaries for the applications are stored in the bin directory of the distribution and command lines used to compile each application are also stored in a log directory for future reference.

All the applications in OmpSCR present a common style. All use the functionalities provided by a common module included in the distribution (stored in the common directory). This common toolkit-module unifies aspects related to the applications programmer interface. The module eases the automation of experimentation and results collection. It implements APIs for: command line arguments processing, timers definition, time measurements and execution reports generation. This module has bindings for C, C++ and Fortran90/95 languages, presenting a very similar API.

All the applications currently included in the OmpSCR distribution contains the GNU general public license (GPL) or BSD license. Although the OmpSCR team encourages the use of an open software license, the original author may protect her code as she wants, as far as she grants: free distribution of the source code in OmpSCR, free use of the generated program and free publication of performance results on any platform. Our purpose is to open the possibility for industry programmers to show up their developments and include real or even commercial applications in the repository.

In the remaining of this section we will briefly describe each of the applications currently included in the repository.

3.1. Computing π

Computing π is a classical program in any parallel API. It is a very simple code that should provide almost linear speedup on any platform, because the iteration space of the main loop is parallelized. The only input parameter of the program is the desired precision in the estimation of π.
3.2. Mandelbrot set area

The Mandelbrot set area is an open question. Statistical methods have been used to estimate it, because obtaining an accurate analytical estimate is not easy. The Mandelbrot program in the OmpSCR computes an estimation to the Mandelbrot set area using MonteCarlo sampling. This code is frequently used to model situations with load imbalance, because the number of iterations to be performed to check the convergence of a point in the complex plane changes depending on the point. The input parameter for this program is the number of points in the complex plane to be explored.

3.3. Molecular dynamic

The molecular dynamic (MD) application in the OmpSCR is an implementation of the velocity Verlet algorithm for MD simulation. Given positions, masses and velocities of \( np \) particles, the program computes the energy of the system and the forces on each particle. A numerical iterative procedure is used to obtain an approximation whose precision depends on the number of simulation steps. The number of particles, \( np \) and the number of simulation steps are the input parameters for the program.

3.4. Quick sort

The Quick sort algorithm is one of the most popular sorting algorithms. The Quick sort application in the OmpSCR mixes recursion and parallelism and makes use of multilevel parallelism. The execution time depends on the specific instance of the problem and therefore the program computes the mean time for 10 different instances randomly generated. The size of the integer vector to be sorted is the only input parameter.

3.5. Divide and conquer fast Fourier transform

This program is a parallel and recursive version of the well known Fast Fourier Transform (FFT) algorithm. It computes the discrete Fourier transform of a signal. The signal is represented by a vector of \( N \) complex elements. The code uses multilevel parallelism and the only input parameter is the size \( N \) of the input signal.

3.6. Bailey’s “6-step” FFT

It is another implementation of the FFT algorithm. This program is part of the CMU task parallel suite and mixes data and task parallelism.

3.7. Loops with dependences

This application bind includes several examples of parallelization of loops with forward and backward carried dependences. Examples of bad parallelized codes are included. One of the loops is parallelized building a threads virtual pipeline. This application is intended as a primer on OpenMP use for non-trivial loops parallelization.

3.8. Cellular automaton

This program includes 4 different Fortran90 implementations of a cellular automaton program implementing the Jacobi iterative algorithm to solve the heat equation in a 2D space represented by a cell matrix. Parallel structure is clearly exposed. Each solution uses a different type of primitives to keep the synchronization: parallel-for primitives; parallel-sections; lock-variables; or even flushed-variables. It is another teaching example for alternatives to direct loop parallelization, focusing on comparing directives and performance.

3.9. LU decomposition

In linear algebra, a LU decomposition is a process for decomposing a matrix \( M \) into a product of a lower triangular matrix \( L \) and an upper triangular matrix \( U \): \( M = LU \). The LU program in the repository is an implementation of the LU algorithm for a dense matrix. It is an example of the use of a parallel-for directive with a stride schedule for load-balance and non-uniform load across iterations.

3.10. Graph search

This application is an example of how to implement a master-slave paradigm to parallelize a deep-first search algorithm on a direct acyclic graph. The master-slave paradigm solves the irregular load-balance produced during the search expansion. It uses a task pool which access is controlled by critical sections to avoid duplicated search paths across processors. Thus, scalability may be compromised by this highly synchronized scheme.

3.11. Jacobi

This program has been provided by Dieter an Mey [1] and includes three different C versions and ten Fortran90 implementations to solve a finite difference discretization of Helmholtz equation using the well known Jacobi iterative method. Different implementations show how the OpenMP parallelization overhead and performance can be modified using several strategies, like extracting the parallel region, barriers, using pipelining techniques, etc.
3.12. SortOpenMP

Some algorithmic patterns are difficult to express in OpenMP. This set of applications provided by Michael Süss [17] are written in C++ and consist of several different implementations of a simple sorting algorithm (Quick-Sort) to illustrate problems with recursion and the avoidance of busy waiting. Implementations compare several solution approaches with respect to programming expense and performance: stacks, nesting, as well as condition variables and the sched yield-function (for busy waiting), etc. Also, a non OpenMP standard version that uses the workqueuing model [15] is available. In this version the recursion problem is solved using task and taskq pragmas.

4. The OmpSCR infrastructure

The infrastructure of the OmpSCR web site [10] is based on PostNuke [12]. PostNuke is a portable multilingual open software content management system (CMS) written in PHP. It is specifically oriented to user communities in which users actively post contents using web browsers. We have chosen this type of tool to support OmpSCR because it simplifies the administration of such a web site. The CMS allows administrators to dynamically work with a structured environment, manipulating any kind of contents: articles, news, FAQs, file downloads, etc.

To join the community, a new user must register herself on the web site. This allows her to actively participate, posting contents, asking questions, or joining the discussion forums. Moreover, a registered user may configure her account, adapting her web site interface. Contributions from registered users must be reviewed and approved by the administrators before they are released.

We now quickly describe the most relevant modules of the web site, focusing on their practical use. However, the best information source about the web site contents and use can be found inside itself. The user will find a complete FAQ file with answers to most of the usual questions.

The downloads section is the core of OmpSCR. It allows to download the repository and other related information. Registered users may also apply to add new files to be download section filling up a simple form. This section has the following subsections: Documentation, OmpSCR distribution, Publications, and Results. The OmpSCR subsection contains the repository directory structure and applications, which may be downloaded separately. Results subsection includes performance results for the different applications measured for specific platforms. These results are shown as plots as well as raw numerical data. More results may be also supplied by users.

A registered user may also post other information items to the OmpSCR web site: news items, reviews, comments, or hyperlinks to interesting web sites. News are the easiest way to supply new information of any nature to the OmpSCR community. Usually the news section will be used to discuss new applications added to the repository, computational results, call for papers for related conferences, etc. Reviews and discussion forums are other alternatives to actively participate in OmpSCR.

OmpSCR is an active SourceForge project [11]. The advantages that the OmpSCR web site offers are increased by the possibilities of SourceForge: bugs reports, features requests and much more support is now available.

5. Computational results

In this section we present some performance results obtained for some of the OmpSCR applications. We focus on the kind of discussions that such results may motivate, instead that on the values themselves.

Figure 1 shows the results obtained when executing the Mandelbrot and Molecular dynamics applications on four different platforms. The Intel label corresponds to a shared-memory Intel-Xeon multiprocessor PC, with four 1.4 GHz processors, and a 400 MHz system-bus. The IBM tag represents an RS-6000 IBM machine with 375 Mhz Power3 processors with 64Gb of memory. HP corresponds to a Compaq HPC 320 with 1 GHz Alpha EV68 processors and 80Gb of memory. The SGI plot corresponds to a SGI Origin 3000 with 600 MHz MIPS R14000 processors, and 160Gb of memory. In all the platforms we have used the native OpenMP compiler. In the case of the Intel and IBM platforms we had exclusive use of the resources.
Both applications, Mandelbrot and Molecular dynamics, show a similar behavior for all the platforms: almost linear scalability, as expected. However, in the case of the Compaq platform, the non-exclusive use of the CPUs produces a negative impact on measures.

On the other hand, Figure 2 compares the speedups obtained on the same platform (Intel) for different applications. They are compiled using the Intel C++ 7.0 compiler for Linux. Applications considered are:

- **Mandelbrot (MANTEL)** with an input-data of 16k.
- **Molecular dynamics (MD)** with an input-data of 8k particles and 10 simulation steps.
- **Pi (PI)**, computed with a precision of $10^{-8}$.
- **FFT (FFT)**, with an input signal of 2M complex data-items.
- **FFT-Bailey (FFT6)**, with an input signal of 1M complex data-items and 10 iterations.
- **Quicksort (QSORT)**, sorting 4M integer data-items.

The first three applications show an almost linear scalability behavior. However, FFT-Bailey show a logarithmic speedup, and the last two applications sublinear speedups. This effect appears because the compiler is not exploiting the multilevel parallelism present in these applications.

### 6. Conclusions and future work

In this paper we have presented OmpSCR. We expect that this infrastructure becomes a meeting point for the OpenMP users and developers community. The repository allows:

- To access and share source codes using them for development, experimental research, etc. Also, it allows to contribute new codes to be included in the repository.
- To obtain, compare, publish and discuss computational results obtained from the applications.
- To show and to evaluate different parallelization schemes for a given application on different platforms, using different languages and/or compilers.
- To turn into a meeting point to discuss about OpenMP. Forums and sections of opinion articles have been created with this intention.
- To benefit from a web infrastructure and the advantages of a SourceForge project.
- Finally, it will be used as a resource to help in the OpenMP learning, presenting real implementation examples of the language features.

For these reasons, we want to appeal the OpenMP community to use OmpSCR, and we offer our support to adapt those relevant applications whose developers are interested on including them in OmpSCR.

OmpSCR is nowadays in a stable version and constant development. At the present time, the tasks we are dealing with are:

- Popularization of OmpSCR among the OpenMP users and developers community.
- Finding and including new applications in the repository.
- To appeal to the active working groups on this field to contribute with their own codes.
- Conducting computational experiments on different platforms, and publishing the results in the web site.

Although OmpSCR is not proposed as an OpenMP benchmark on its own, we do not discard a future effort in this direction. Anyhow, the own success of OmpSCR depends on the acceptance of the HPC community, and it will determine its long term scope.

### Acknowledgments

Computational results have been obtained using the installations of CEPBA, CESGA and CIEMAT. This work has been partially supported by the Canary Islands government, contract PI2003/113, and also by the EC (FEDER) and the Spanish MCyT (Plan Nacional de I+D+I, contracts TIC2002-04498-C05-05 and TIC2002-04400-C03-03).
References


