Abstract

R\(^1\) is a domain specific language widely used for data analysis by the statistics community as well as by researchers in finance, biology, social sciences, and many other disciplines. As R programs are linked to input data, the exponential growth of available data makes high-performance computing with R imperative. To ease the process of writing parallel programs in R, code transformation from a sequential program to a parallel version would bring much convenience to R users. In this paper, we present our work in semi-automatic parallelization of R codes with user-added OpenMP-style pragmas. While such pragmas are used at the frontend, we take advantage of multiple parallel backends with different R packages. We provide flexibility for importing parallelism with plug-in components, impose built-in MapReduce for data processing, and also maintain code reusability. We illustrate the advantage of the on-the-fly mechanisms which can lead to significant applications in data-centered parallel computing.

Categories and Subject Descriptors D.1.3 [Programming Techniques]: Concurrent Programming-Parallel Programming

General Terms Languages, Performance, Design

Keywords parallelization, domain specific language, automatic code generation, data-centered applications, MapReduce

1. Introduction

As R is arguably the most popular programming environment used in data analytics, mining, and statistical computing, R’s performance in handling large-scale data and its parallelism become imperative. R is a functional language and contains interfaces to other general-purpose programming languages. As of August 2011, there are more than 3,000 R packages and among these more than 20 are dedicated to parallel computing. There is some overlap in functionality between many of these and between them they cover some aspects of shared memory, cluster and GPU computing.

OpenMP is a popular specification for parallel programming that is primarily intended for shared memory multiprocessing. Several compilers implement the OpenMP API for C/C++/Fortran programming languages. While many R functions are written in C/C++/Fortran and potentially can access OpenMP parallelism of the compilers, this is not possible in general R code because it is an interpreted language; the interpreter must come along to every R thread. In this paper, we consider the OpenMP API separately from its compiler implementations. We use the API to specify parallelism in R code at the frontend but implement it with a mix of available parallel R packages. This brings a major parallel programming standard to R. Based on user-added pragmas, we make the necessary R code transformation with the following properties:

- **Flexible Plug-In to Parallel Backends**: Any supported R parallel backend can be loaded to transform the code on demand.
- **Data Processing with Built-In MapReduce Mechanism**: The R data frame or variable declared in pragma, associated with a combination function, leads to a multithreaded MapReduce-style execution of the R function after code transformation.
- **Code Reusability**: All the pragmas appear as comments in the code, so the sequential execution is never affected.

2. Related Work

Data-intensive computing continues to grow in modern sciences. The broad range of applications in statistics, machine learning and data mining result in a set of special-purpose languages and tools such as OptiML [1]. Such a domain-specific approach implements data-parallel operations at a higher level than a general-purpose programming language and users call such APIs with built-in parallelism. In R, pnmath [6] provides an OpenMP wrapper framework for some vector math functions that are written in C. Recent work [3, 5] in imposing parallelism through user-added pragmas illustrates the blending between advanced parallelism and lower-level code, but many are dedicated to one specific type of parallelism. We take advantage of multiple R parallel backends, Rcpp [4]...
Parallelizing R programs differs from that of a general-purpose language. R is a mix of interpreted and compiled components, it is single-threaded, and common parallel standards are not directly available in R. However, there are different levels of parallelism for R: in the spirit of the domain-specific approach taken by [1], a set of common functions, such as probability density functions, is implemented with OpenMP in `pnmath` and can scale very well up to 1,024 cores; multiprocessing, represented by R packages `multicore` and `foreach`, is widely used for its generality in handling R functions and types of system calls at process level; and R’s interoperability allows linking external libraries for high-performance computing (e.g. BLAS and ATLAS). Figure 1 shows our proposed parallelism at a higher level: a piece of R code can be partitioned to sections with dependency (by data or as a workflow), and then other R packages provide support as parallel backends on heterogeneous hardware architectures.

We implement some OpenMP parallel specifications with existing parallel R components. Our R code translator function takes an R function with embedded OpenMP-style pragma comments and produces an R function that runs in parallel. Its parallel implementation is executed with a specified existing backend such as `foreach` and `multicore`. Because the implementation is with an existing backend, its performance is comparable to the usual performance of the given backend.

We illustrate this by inserting a Map-Reduce pragma comment in the example given for the function `find.matches` of package `Hmisc`, given in its documentation:

```r
x <- matrix(runif(50000), ncol=2)
y <- matrix(runif(100000), ncol=2)
#pragma mpr parallel partition data(x:matrow)
w <- find.matches(x, y, mxmatch=5, tol=c(.02, .03))
```

Fig. 2 shows its performance using a strong scaling test with up to 128 cores (on an SGI Altix UV 1000 system, 2.0GHz per core, 4 TB global shared memory and 8 GPUs in a single system image). The data-centered task is to find close row matches of two matrices. Detailed pragma specification and available parallel constructs will be provided with the alpha release of our package `ROPenMP`.

**References**