Rdsm: Distributed (Quasi-)Threads Programming in R

Norman Matloff\textsuperscript{1,*}

1. Dept. of Computer Science, University of California, Davis
*Contact author: matloff@cs.ucdavis.edu

Keywords: parallel programming, threads, shared-memory, collaborative applications, Web data collection

\textbf{Rdsm} provides a threads-like programming environment for R, usable both within a multicore machine and across a network of multiple machines. The package gives the illusion of shared memory, again even across multiple machines on a network.

Consider for instance the assignment \( y \leftarrow x \). In a message-passing setting such as \texttt{Rmpi}, \( x \) and \( y \) may reside in processes 2 and 5, say. The programmer would write code

\begin{verbatim}
send x to process 5
to run on process 2, and write code
receive data item from process 2
set y to received item
to run on process 5.
\end{verbatim}

By contrast, in a shared-memory environment, the programmer would merely write

\begin{verbatim}
set y to x
\end{verbatim}

vastly simpler.

Accordingly, many in the general parallel processing community find that the shared-memory approach makes code easier and faster to develop, and easier to maintain and extend. See for example Chandra (2001), Hess \textit{et al} (2003) etc.

\textbf{Rdsm} should have a wide variety of applications, such as

- Performance programming, in “embarrassingly parallel” settings.
- Parallel I/O applications, e.g. parallel Web data collection.
- Collaborative tools.

Again, these applications can be done via message-passing too, but we argue that the shared-memory paradigm makes these applications easier to develop, maintain and extend.

We will also compare \textbf{Rdsm} to other parallel R packages, in terms of paradigm, flexibility and convenience.

Finally, two general points about parallel R and parallel programming in general will be presented. First, it will be argued that for non-embarrassingly parallel situations, the nature of R presents a fundamental obstacle to performance, so that one is essentially forced to have R call parallel C code, say with direct threads or via OpenMP. Second, a simple mathematical argument will be presented regarding parallelization of for loops, showing that in most cases it is not profitable to micromanage allocations of iterations to processes.

References

Chandra, Rohit (2001), \textit{Parallel Programming in OpenMP}, Kaufmann, pp.10ff (especially Table 1.1).