Parallelizing a Computationally Intensive Financial R Application with Zircon Technology

Zircon Computing LLC[†], Mascon Global Limited[‡], and Garrett Asset Management LLC[§]

[†]30, Galesi Drive, Suite 202B, Wayne, NJ 07470, USA
[‡]1841, Broadway, Suite 700, New York, NY 10023, USA
[§]800, Third Avenue, New York, NY 10022, USA

Abstract

Statisticians, analysts, scientists, and engineers require massive processing power to conduct data analysis, predictive modeling, visualization, and other complex tasks. This paper describes how we substantially improved the performance of a representative complex computational finance application by integrating the Zircon adaptive ultra high-performance computing software platform and tools with the R programming language and environment. This integrated solution uses distribution and parallelization to reduce the total computation time of the R-based application from 3,093 minutes to 40 minutes on a commodity multi-processing platform.

1 Introduction

Companies in competitive domains, such as financial services, digital media, text mining, and enterprise content management, create large data repositories containing large amounts of data collected from their daily operations. Analyzing this archived data can yield knowledge that drives future business and provides significant market advantages over competitors. Although companies could use specialized super computers, such as Crays, the custom development time and hardware costs are prohibitive.

Another approach is to use conventional high-performance computing (HPC) software platforms, such as MPI [9], PVM [6, 8], OpenMP [10], or Globus [5], atop multi-core/multi-node platforms. Although this approach is more flexible than using specialized super computers it has following drawbacks:

- **Doesn't leverage or scale to the entire network.** Applications based on conventional HPC software platforms seldom leverage the capabilities of the latest multi-core processors, the LAN, the enterprise WAN, or any external "cloud" networks since conventional HPC solutions cannot adapt dynamically to changing workloads and resource availability.
- **Custom development and integration.** Conventional HPC software platforms require extensive manual development and integration of custom server and application programming before they can work (and many common and legacy apps cannot be modified unless they are redeveloped).
- **Tied to modified apps.** Once applications are customized, they are locked in to a particular HPC platform and deployment configuration, and cannot leverage updates without redoing the intense customization.
- **Complex setup with no support for automated plug and play.** Conventional HPC software platforms require complex setup and customization to adjust the load manually on all processors in the network since they don't have automatic adaptive load balancing.

This paper describes how we overcame the limitations with conventional HPC platforms to substantially improve the performance of a representative complex computational finance application—the Garrett Asset Management Backtesting System—by integrating R [11] and Zircon software [1]. R is a programming language and software environment for statistical computing and graphics that provides rich platform support for data modeling, analysis, and visualization. Zircon is an adaptive ultra high-performance computing software platform and tools that requires no server development; minimal application integration; is easy and automatic to setup; and scales quickly from a few laptops to tens of thousands of multi-core servers ranging from multi-core, LANs, WANs, and clouds.

The Garrett Asset Management (GAM) Backtesting System financial application uses historical and hypothetical simulations to assess mathematical models used to drive high-frequency electronic trading and help inform complex decision-making. Like many financial applications, the GAM Backtesting System executes a large number of logically independent and computationally intensive calculations to simulate the behavior of various algorithms on historical data. For example, the GAM backtesting system executes backtesting financial models over a sequence of historical time periods - there could be lot of models, and the time periods may also be large. For each combination of a model and a time period, the GAM backtesting system performs the same computationally intensive calculation and collects results until computations are performed for all the models over all the time periods.

R was a natural choice to meet GAM's statistical analysis requirements due to its convenient abstractions that allow statisticians and engineers to run complex data analysis with a few commands. R's interpretive structure, however, was a limiting factor for delivering complex and mission-critical analysis in a timely fashion. By integrating R with Zircon software, therefore, users can focus on writing their application logic related to data modeling, analysis, and visualization in the intuitive R language and leave the complexities of accelerating computation-intensive R applications to Zircon software' adaptive ultra high-performance computing solution.

The remainder of this paper is organized as follows: Section 2 describes the limitations of applying traditional parallel computing solutions for speeding up compute-intensive R applications such as the GAM backtesting system; Section 3 describes the structure and functionality of the Zircon software; Section 4 describes how the Zircon software was used to parallelize and speed up the complex compute-intensive calculations of the GAM backtesting system; Section 5 analyzes the empirical results that demonstrate the performance improvements after parallelizing the sequential GAM backtesting system using the Zircon software; and Section 6 presents concluding remarks and lessons learned.

2 Strategies and Challenges for Accelerating Compute-Intensive R Applications

R is a statistical software package that provides analysts with a comprehensive environment for statistical computation, data manipulation, data analysis, and graphical data visualization. Apart from providing a full-fledged statistical language with first-class support for performing operations on complex data structures such as arrays and matrices, R also provides many libraries and packages for performing complex statistical operations and assisting analysts in varied domains including finance, econometrics, clinical trials, machine and statistical learning. natural language processing, and genetics [7].

With the proliferation of R in various domains requiring complex data analysis tools, several strategies have emerged for accelerating R applications. For example, custom analytic routines and libraries can be written in compiled languages, such as C and C++, and run within the R environment to speed up analysis and enhance reuse, thereby saving significant development time and effort. To facilitate this integration, the *Rcpp* [4] and *RInside* [3] packages can be used to seamlessly integrate C/C++ code in R packages.

Another strategy for accelerating compute-intensive R applications is to leverage current generation hardware advances and perform complex calculations in parallel across multiple processors and deliver results quickly and scalably. For example, R has been integrated with *Rmpi* [16] and *snow* [15] to distribute calculations over many computers. Existing strategies for accelerating complex R applications incur the several challenges for mission-critical applications, including:

- Existing parallel computing solutions [12] for R are based on conventional grid computing middleware. Traditional grid middleware is cumbersome to program, however, for the reasons described in Section 1. As a result, significant amounts of ramp-up time and effort must be spent on tutorials, webinars, and other study materials to educate developers on these hard-to-program technologies.

- Integration technologies, such as *Rcpp*, are effective for speeding up computations within a single processor by replacing interpreted R code with efficient compiled code. These integration technologies, however, cannot easily perform complex calculations and analysis in parallel across multiple computers in a cloud environment.

What is needed, therefore, is a solution that can leverage both hardware and software innovations in distributed and parallel computing, while simultaneously reducing the learning curve and effort needed to incorporate these innovations into mission-critical applications. In particular, an ideal solution should allow data analysts to (1) easily distribute complex calculations/analysis across multiple processors and execute them in parallel, (2) invoke analysis routines in compiled and/or interpreted languages seamlessly on any OS/hardware platform, and (3) improve runtime performance via advanced hardware technologies (*e.g.*, using massive processing capabilities on a cloud) and software technologies (*e.g.*, by switching between R and compiled code as needed).

3 Solution Approach: Integrating R with the Zircon Software

The Zircon Software Product Suite [1] from Zircon Computing provides an adaptive ultra high-performance computing middleware platform that substantially accelerates the performance of R applications and addresses the challenges with existing strategies described in Section 2. Zircon software automatically deploys a distributed computing infrastructure across (potentially) heterogeneous hardware platforms and operating systems, maps compute-intensive applications to a pool of processors, manages their execution, and dynamically equalizes the workload in real time to fit available resources. Application developers can thus exploit the processing power available to them, including newer technologies, such as multi-core processors and cloud computing systems, as well as traditional desktops and servers.

3.1 Features and Benefits of Zircon Software

Below we describe the key features and benefits of Zircon software.

3.1.1 Extreme Performance

Zircon provides an adaptive ultra high-performance computing solution via the following features:

- **Real-time load equalization.** Zircon utilizes and distributes the workload in real-time across all available computing and networking resources, including multi-core desktop, LAN, WAN or any accessible cloud network. This load equalization ensures every processor in the grid is optimized to maximize computing performance.
- **Transparent scalability.** Processors and cores can be added or removed (and allocated for other tasks) without disrupting ongoing operations since Zircon software automatically recognizes the state of the processors and allocates workload without changing application software.
- **Distributed data caching.** Zircon software can cache large data structures on servers by sending the data just once, then sending a reference to the cached data on each server during each request. This distributed data caching accelerates distributed applications where communications overhead is significant compared to the actual computation time.
- **Ultra fast data transfer.** Unlike conventional middleware, that bottlenecks application data between clients and servers by using text-based protocols (*e.g.*, HTML, XML, and SOAP), Zircon software automatically generates optimized binary protocols that transfers results much faster.
- **No virtualization overhead.** Zircon software runs at native operating system speeds on heterogeneous operating systems, hardware platforms, programming languages, and network environments with no virtualization overhead. These features enable the fastest computing performance possible, increasing computations up to 98faster for select applications, and from 2 to 9 times faster than conventional HPC software platforms.

3.1.2 Minimal Development Effort

Zircon minimizes the time to develop HPC applications via the following features:

- **No server-side development.** Unlike competing HPC platforms, Zircon can execute existing application functions and algorithms in parallel without requiring any server-side development. This capability allows quicker development and can utilize existing servers including those within the cloud. Conventional HPC software platforms cannot be used in cloud computing–since external data centers would not want you tampering with their servers.
- **Minimal application development.** Zircon software is designed as a component-based framework that contains many "knobs" can be extended and tuned transparently to easily and quickly support new user requirements and application feature enhancements.
- **Maintains application security.** Zircon software does not need to know the application data and sensitive business logic to operate, which means it is always secure and confidential.

3.1.3 Rapid Configuration and Deployment

Zircon minimizes the time to configure and deploy HPC applications via the following features:

- **Automatic parallel configuration.** Zircon includes asynchronous adapters that quickly configure existing non-parallel application code to run in parallel execution that run much faster by leveraging the processing power of the entire grid. These adapters can also configure applications to run in collocated and/or distributed parallel deployments that maximize the use of available computing resources.
- **Platform independence.** Any distributed and/or collocated computation can be deployed on any popular

operating system or platform with complete and automatic interoperability. This platform independence means a Windows application can leverage the processors on Linux, Solaris, AIX, Mac, and other operating systems without having to re-write or port the applications themselves since Zircon handles the conversions automatically.

3.1.4 Intuitive Use and Administration

The following features make Zircon intuitive to use and administer:

- **Automatic load equalization.** Unlike conventional HPC software platforms, Zircon software automatically equalizes the load between all of the available (heterogeneous and/or homogeneous) processors in the grid adaptively, which eliminates the tedious trial and error needed to maximize performance.
- **Automatic service discovery.** Zircon software dynamically discovers and optimizes all processors available at runtime. When new machines are added or removed from a deployment, Zircon software will automatically reconfigure accordingly, which ensures maximum performance at all times with little or no administrative input.
- **Automatic real-time monitoring and auditing.** Zircon software provides powerful tools for automatically monitoring and transparently auditing huge volumes of application and system events. These tools minimize the total cost of ownership by enabling real-time decision making that is more accurate and relevant than is possible with manual monitoring and current auditing approaches.
- **Persistent and recoverable.** Zircon software uses a self-adaptive, fault-tolerant architecture that ensures that applications will automatically recover and transparently re-execute requests on different servers if existing servers disconnect or fail.

3.2 The Zircon Software Architecture

Applications built using Zircon software are known as *zEnabled* applications. Zircon software supports three computing and communication models required by many mission-critical zEnabled applications that need ultra high performance, as shown in Figure 1 and described below:

- **Application function parallelism**, such as the capabilities provided by computation grids to run application operations in a cluster of servers as if they are programmed for a single computer. The *zFunction* function parallelism API and supporting tools hide many low-level network programming concerns and unexpected complexities, simplifying fine-grained application parallelization.

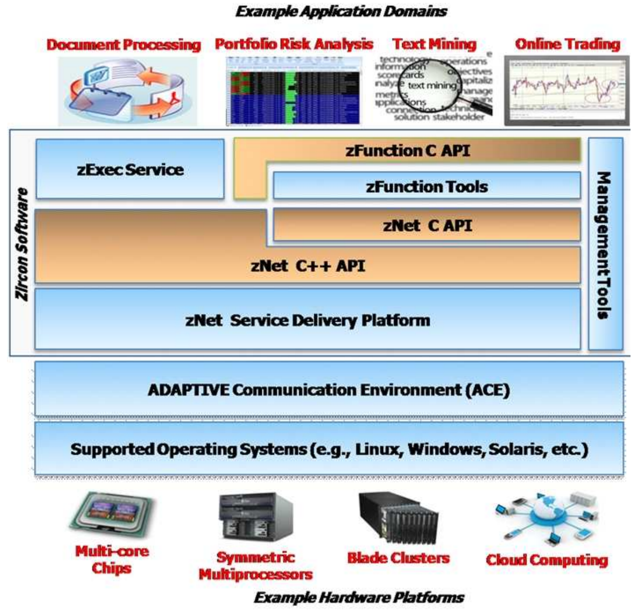

Figure 1: **Zircon Software Architecture**

- **Application executable parallelism**, such as the capabilities provided by data centers and clouds to launch applications on demand. The *zExec* application execution parallelism service runs any executables in a cluster of servers as a set of parallel jobs, thereby simplifying coarse-grained application parallelization.
- **Service delivery platforms**, such as the capabilities provided by distributed computing environments that support cooperating business tasks via distributed infrastructure patterns [2], such as *Messaging*, *Broker*, and *Publisher/Subscriber*. The *zNet* API provides a C++ interface to the zNet service delivery platform that handles service discovery, reliable multicast communication, request load balancing, and request dispatching.

Requests from applications that use these three models can run on processors and cores in a collocated and/or distributed manner, with the choice of collocation or distribution largely transparent to application clients and servers. Zircon software runs on all popular general-purpose and real-time operating systems since it is implemented atop the open-source ADAPTIVE Communication Environment (ACE) [13, 14], which is portable C++ host infrastructure middleware that shields Zircon software from operating system dependencies.

The remainder of paper describes how we developed a distributed and parallelized zEnabled version of the GAM Backtesting System using the Zircon zNet service delivery platform described above.

4 A zEnabled-R Solution for the GAM Backtesting System

This section describes how the Zircon software was used to parallelize and speed up the complex compute-intensive calculations of the GAM backtesting system. First, we describe the structure of a typical zEnabled R application. Next, we describe how we used this structure to implement the parallelized version of the GAM backtesting system.

4.1 Structure of a Typical zEnabled R Application

zEnabled R applications follow the general Zircon software paradigm of having a client application invoke multiple asynchronous requests on remote compute servers (Figure 2, Steps 1 and 2) through Zircon's dynamic real-time load equalizer (which resides within the zNet Client Middleware as shown in Figure 2). Parallelization is achieved

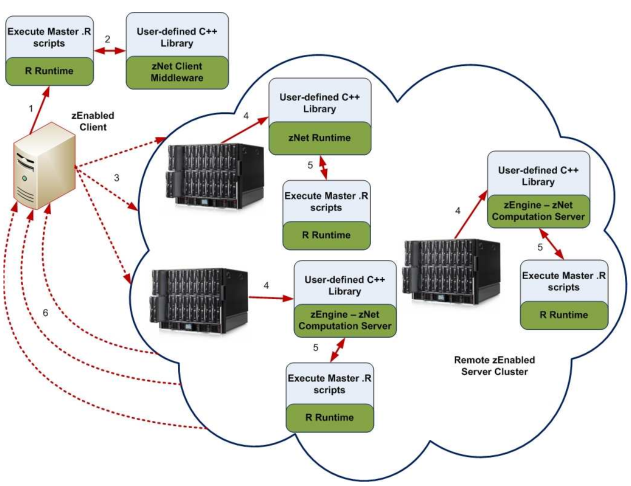

Figure 2: **The Structure of a Typical zEnabled R Application**

by executing these requests in parallel on multiple generic Zircon compute servers, known as *zEngines* (Figure 2, Step 3). As soon as each computation completes its results are returned to the client application (Figure 2, Step 6). The request and results are performed in one function call that is synchronous from the perspective of R application developers, i.e., remote computation results are available for use by a client after all results are returned.

The zEnabled client application consists of R code that loads Zircon software as a plug-in library. Likewise, the server application dynamically loads the R language library and as user-defined R functions into a zEngine process that provides a lightweight container for zEnabled software (Figure 2, Steps 4 and 5). To execute R code, the zEngine loads both the R language interpreter library and application-specific R definitions for functions that will process requests received from clients in parallel with other requests.

Zircon software on the client and server(s) uses a compact, application-specific binary protocol to transmit requests and results. This protocol is implemented via serialization/deserialization operators for user-defined C/C++ types. The interface between the user's R code and Zircon software therefore requires defining mappings between native R data types and serializable C++ objects for each R function with a distinct signature. These are provided by a user library that can be written using Zircon APIs and tools.

To simplify the development of user libraries for R integration, Zircon software provides the *zNet-R* component, which wraps an instance of the R interpreter implemented as a *RInside* object. RInside and its associated *Rcpp* package provide a facade for accessing and manipulating native R types from within a C++ application. Developers of user libraries can use these utilities to quickly map types used by R to/from serializable C++ objects, such as STL vectors and strings. The Rcpp package, in particular, allows users to serialize diverse R types, such as values, vectors, matrices, and data frames.

4.2 Implementing a Parallelized Garrett Asset Management (GAM) Backtesting System

Figure 3 compares and contrasts the original sequential GAM Backtesting System with the corresponding zEnabled parallelized version. Figure 3a shows how the original application contains two nested loops that iterate over a number of financial models (the `MODELS` list) and a large set of points in a multi-dimensional parameter space (`StratPars`). The bulk of the calculation is performed by the `GenericgetNAVs ()` function that simulates a particular model with a set of parameters drawn from this space. The parameter space is large and may have any number of dimensions, with typical runs totaling nearly 100,000 distinct invocations. Most of these invocations are homogeneous, taking several seconds to complete on commodity hardware.

The parallel zEnabled implementation shown in Figure 3b is structurally similar to the sequential code. Just as the sequential implementation can be defined by the post-condition that the `allres` matrix has been completely populated with results at the end of the loops, the parallel implementation exits the block of code with the same state. Similarly, both pieces of code iterate over a number of ob-

```
for (x in MODELS)
{
    DoBacktest (….)
    {
        BacktestAStrategy (…..)
        {
            CompleteSpace (…..)
            {
                for (l in 1:length (StratPars$x)
                {
                    allres <- rbind (allres,
                                GenericgetNAVs (…))
                }
            }
        }
    }
}
```

(a) Pseudo Code of the Sequential Implementation

```
for (x in MODELS)
{
    DoBacktest (….)
    {
        BacktestAStrategy (…..)
        {
            CompleteSpace (…..)
            {
                // every iteration of the for loop
                // executed in remote servers
                // with independent data

                    allres <- call_get_navs  (…))
            }
        }
    }
}
```

(b) Pseudo Code after Parallelization with Zircon Software

Figure 3: **Different Implementations of the GAM Backtesting System**

jects in the `MODELS` list and execute simulations with them. In contrast, the nested loop in Figure 3a that iterates over the points in a parameter space is replaced in its entirety in Figure 3b with a single call to the `call_get_navs()` function, which is an entry point into this application's user library code that interfaces directly with Zircon software.

The `call_get_navs()` function reproduces the behavior of the inner loop in Figure 3a by (1) dispatching asynchronous requests to remote zEngine compute servers, (2) awaiting the delivery of all results, and (3) populating the `allres` matrix. While each request is asynchronous, and thus may execute in parallel in a cloud, the `call_get_navs()` function is synchronous from an R application perspective. This routine is implemented as a lightweight R wrapper around a C++ function and uses Rcpp to map between native R types (such as the `allres` matrix and its constituent rows) and serializable C++ types (such as std::vector<double>). The performance boost delivered by the zEnabled implementation outlined in Figure 3b was dramatic, as shown in Section 5.

5 Analysis of Empirical Results

This section presents the results of the experiments that quantify the benefits of parallelizing and distributing the GAM Backtesting System application using the Zircon zNet-R software. All experiments were run on a testbed containing 20 Intel-Xeon 1520 dual-series dual-processor/dual-core (for a total of 80 cores) 1.86 GHz machines running on 64-bit Red-Hat Enterprise Linux 2.6 and connected using Gigabit Ethernet. We dedicated one core to run the client application that triggered requests to all the zEngine computation servers. Three other cores on the client processor were left idle since we did not want to run computation servers on the same machine where the client application ran. We therefore used a total of up to 76 cores for zEngine computation servers.

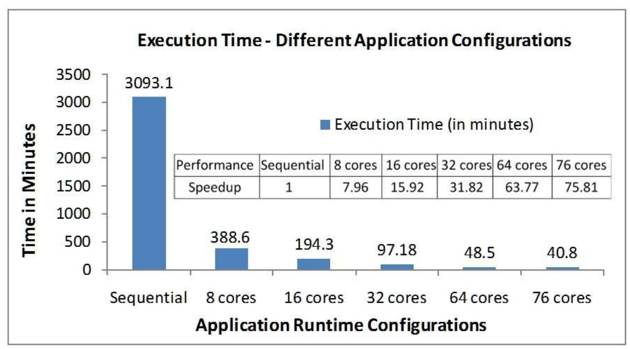

Figure 4: **Performance of zEnabled GAM Backtesting System**

We ran the distributed parallelized version of the GAM Backtesting System application on four, eight, sixteen, and nineteen multi-core machines and compared the performance of the application in each of these configurations with the performance of the baseline sequential application implementation. Since each machine in the experiment contained four cores, we started four instances of the GAM Backtesting application server on each machine to leverage all the four cores in each machine. The results of the experiments shown in Figure 4 show how is the performance gain was nearly linear with respect to the number of cores used in each experiment configuration.

In particular, the time taken for the sequential application to calibrate and analyze two models is 3,093 minutes.

Conversely, it took 40 minutes for the distributed parallel version of the GAM Backtesting System to process and to analyze the same two models, thereby demonstrating substantial acceleration and efficiency. The performance gains achieved by the distributed and parallel version of the GAM Backtesting System are limited by the number of cores/machines available to run these experiments in our testbed. In large-scale deployments (*e.g.*, clouds or datacenters with hundreds of machines and thousands of cores), there is no limit on the number of cores/machines that can be used to deploy the parallelized GAM Backtesting application. In such deployments, the GAM Backtesting System could be scaled transparently to utilize all the available cores and provide accelerations much higher than those shown in Figure 4.

6 Concluding Remarks

This paper demonstrated how the Zircon adaptive ultra high-performance computing platform has been integrated with the R programming environment to accelerate complex and compute-intensive R applications in competitive domains, such as financial services, digital media, text mining, and enterprise content management. Zircon software dramatically improves performance with little learning curve and configuration effort, and run seamlessly over local-area networks; wide-area networks; public, private, or hybrid cloud deployments; and/or in dedicated data centers. The results of our experiments show how Zircon software accelerated the performance of the GAM Backtesting system significantly on a cluster of commodity multi-core machines.

Based on our experience parallelizing the GAM Backtesting system, the following are the advantages that R programmers would experience while parallelizing applications like the GAM Backtesting system:

- Programming with Zircon Software and the zNet-R middleware is straightforward and application developers are shielded completely from tedious and error-prone low-level network programming, distributed programming, and parallel programming. This advantage is significant for R users, who chose R to avoid having to wrestle with these low-level programming concerns. R users can thus focus on their application logic related to data modeling, analysis, and visualization, and leave the complex details related to distributed and network programming to Zircon software.

- The parallelization benefit that R users derive from Zircon software is not just restricted to R applications. They could also parallelize analysis applications written in compiled languages, such as C and C++, thereby providing an integrated environment for accelerating complex and compute-intensive applications developed in compiled and interpreted languages.

- Zircon software is extremely efficient and scalable in utilizing all the available processing cores of a high-performance computing platform, as shown in the linear acceleration obtained by the GAM Backtesting System.

7 Participants

Zircon Computing. Zircon Computing, LLC, is an international software and services company based in Wayne, New Jersey. Founded in 2005 by senior technologists from the financial services industry, Zircon Computing is a leading provider of ultra high-performance middleware software and services worldwide, and markets both directly to enterprise clients and through an international network of partners. Zircon Computing is privately held. For more information, please visit `http://www.zircomp.com`.

Mascon Global Limited. Mascon Global Limited is an IP-led, domain-centric technology solutions company with development centers and business operations in the US, Europe, Asia and Mexico. Mascon Global Limited delivers technology solutions across multiple industries and hold leadership positions in the worlds of travel and hospitality, finance, healthcare and life sciences, education, media and telecommunications. Mascon Global Limited uses a comprehensive blend of products, services and a world-class delivery model to build, deploy and maintain technology solutions that help our clients meet their most aggressive business objectives. For more than 25 years, Mascon Global Limited has been the innovation partner of choice for blue-chip firms around the globe. For more information, please visit `http://www.mgl.com`.

Garrett Asset Management. Garrett Asset Management, LLC, is a systematic trader in the financial and commodity futures. The firm was founded in 2009 with emphasis in research in portfolio construction, position sizing, and the development of technical trading models for different market conditions. For more information, please visit `http://www.garrettassetmanagement.net`.

References

[1] J. Balasubramanian, A. Mintz, A. Kaplan, G. Vilkov, A. Gleyzer, A. Kaplan, R. Guida, P. Varshneya, and D. C. Schmidt. Adaptive Parallel Computing for Large-scale Distributed and Parallel Applications. In *Proceedings of the 1st International Workshop on Data Dissemination for Large scale Complex Critical Infrastructures (DD4LCCI 2010)*, Valencia, Spain, Apr. 2010.

[2] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture: A Pattern Language for*

Distributed Computing, Volume 4. Wiley and Sons, New York, 2007.

[3] D. Eddelbuettel and R. Francois. *RInside: C++ classes to embed R in C++ applications*, 2010. R package version 0.2.2.

[4] D. Eddelbuettel, R. Francois, with contributions by Simon Urbanek, D. Reiss, D. B. based on code written during 2005, and . by Dominick Samperi. *Rcpp: Rcpp R/C++ interface package*, 2010. R package version 0.8.0.

[5] I. T. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol.*, 21(4):513–520, 2006.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[7] K. Hornik. The R FAQ, 2010. ISBN 3-900051-08-9.

[8] D. Kranzmuller, P. Kaczuk, and J. Dongarra. Recent advances in parallel virtual machine and message passing interface. *IJHPCA*, 19(2):99–101, 2005.

[9] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2*. High Performance Computing Center Stuttgart (HLRS), September 2009.

[10] OpenMP Architecture Review Board. Openmp application program interface. Specification, 2008.

[11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[12] M. Schmidberger, M. Morgan, D. Eddelbuettel, H. Yu, L. Tierney, and U. Mansmann. State of the art in parallel computing with r. *Journal of Statistical Software*, 31(1):1–27, 8 2009.

[13] D. C. Schmidt and S. D. Huston. *C++ Network Programming, Volume 1: Mastering Complexity with ACE and Patterns*. Addison-Wesley, Boston, 2002.

[14] D. C. Schmidt and S. D. Huston. *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks*. Addison-Wesley, Reading, Massachusetts, 2002.

[15] L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova. *snow: Simple Network of Workstations*, 2010. R package version 0.3-3.

[16] H. Yu. *Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)*, 2010. R package version 0.5-8.