



Using ParallelR™ for High Performance Monte Carlo Simulation on Multiprocessor Computers

Revolution Computing Inc.
www.revolution-computing.com

1. Introduction

Virtually all enterprises are facing increasing levels of competition in the global economy that has tended to level the competitive playing field. This intensified competition drives us to create new innovative products and processes and to optimize the performance and cost of existing ones. Enterprises that depend on the web as a primary interface with their constituencies have a particularly acute problem. Not only do they have to collect, digest, and analyze ever increasing amount of data, but they also must respond with answers and action in real-time to be effective and remain competitive

While traditional design and optimization by empirical testing are no longer competitive in many arenas, most real world problems are either too complex to be completely analyzable by analytical (mathematical) methods or have an important, nondeterministic component. This is the driving force behind the rapid increase in the use of large scale (statistical) Monte Carlo simulation as a fundamental tool in virtually every domain.

It is widely recognized that a fundamental challenge is to develop and utilize modern high performance computing technology to derive rapid response and adaptive simulation capabilities. Furthermore, in some domains such as scientific and engineering research, new experimental technologies often based on nanotechnologies are providing orders of magnitude more experimental data than we have seen even in the recent past. In other domains, the internet is providing us with ever increasing online data which needs to be analyzed.

High performance computing has been playing an increasingly important role. There are no alternatives. Meeting this challenge is difficult and expensive. We can no longer rely on the computer vendors to regularly produce new computers with ever increasing clock speed that run our legacy (sequential) software faster.

Computer vendors have switched from a basic strategy of providing computers with a single CPU with increasingly faster clock speed to providing computers based on an increasing number of CPUs based on a (more or less) fixed clock speed. This is accomplished by a combination of “multicore” CPU chips and clusters of nodes based on these chips. Users are increasingly using grids of such computers are very large problems.



Scientists and engineers have been successfully utilizing a new strategy of parallel computing on commodity multiprocessors. These systems utilize inexpensive commodity hardware, but the development of suitable “scale out” parallel application software can be very difficult for most programmers. The cost per computer cycle is going down at the same time as the cost of the software necessary to make effective use of the cheaper cycles is increasing.

The cost effective development of parallel Monte Carlo applications that can leverage commodity multiprocessors to provide required performance is a key enabler. Fortunately a la Monte Carlo simulations are “embarrassingly parallel” and can be easily implemented using newly available high level software tools.

In order to keep software development and maintenance costs down, application developers are increasingly using high level scripting languages for software development as an alternative to traditional compiled languages such as Fortran or C. A well-known, increasingly popular choice is the interactive R statistical computing environment, cf. [1]. REvolution Computing Inc has recently released its commercially supported high performance package, ParallelR, for parallelizing applications written in R.

By using the Sleight function in ParallelR, an R user can quickly develop and run “embarrassingly parallel” applications with very good results without becoming formally trained in parallel programming. Using the Sleight function, the developer can work completely within the familiar R environment without having to master and use low level tools such as MPI. For the purpose of producing an easy to understand illustration of the general concept, we present an example of the ParallelR package being used in a Monte Carlo application in financial services.

Our results show that the system provides excellent performance and scales very well with increasing numbers of CPU cores. Perhaps as important, we demonstrate that by utilizing the ParallelR system, this type of parallel application software can be developed and run by a typical R user who is not an expert in parallel programming. Indeed the ParallelR system has been designed and implemented with the goal of minimizing the incremental “cognitive load” that a typical R user need to face in utilizing parallel computing. This is a critical observation because conventional approaches to parallel application development are notoriously difficult and time consuming.



2. The Example

Opportunities for utilizing high performance Financial Services systems are common throughout the financial world. Examples include: credit risk assessment, portfolio optimization, optimization of marketing strategies, and credit card fraud detection.

The specific example we consider is a simple, prototype portfolio optimization problem that is a model of the “efficient frontier” approach suggested by Markowitz. cf., [4]. Our intention here is to illustrate the general ideas behind the use of multiprocessors to accelerate general portfolio optimization rather than to present the design and analysis of a production portfolio optimizer applied to real market data. We present benchmark data showing how well our parallel portfolio optimizer scales as a function of the number of cores.

Our model portfolio consists of fixed number of investments whose prices are characterized by randomly selected normal distributions. We want to emphasize that in the real world of finance, an enormous amount of (proprietary) science and art goes into designing the distribution functions that model the behavior of individual investments. We use random normal distribution functions to represent investments because our entire focus here is on the parallel computing aspects of this problem.

We use a Monte Carlo approach to evaluate the reward/risk of a portfolio chosen from these investments with only “long positions” being allowed, ie., no investments are allowed to be “sold short.” Monte Carlo approaches to studying efficient frontiers are of more than academic interest. Schlumberger has been using similar approaches with great success for analyzing reward/risk models in the petroleum with great success for years, cf., [4].

Each investment has an associated weight, w , which is the nonnegative fraction of the total portfolio value that is allocated to that investment. Clearly, each weight, w , must be a fraction between 0 and 1.

Specifically, we generate a large number of random portfolios of “long positions” and for each such portfolio we compute the expected reward and risk by another Monte Carlo process. To be precise, we randomly sample each investment in the portfolio a large number of times and compute the mean (average reward) and risk (variation) for the portfolio. The resulting (reward, risk) points can be plotted in a plane to yield a collection of points. The upper bound of the convex hull of these points is generally called the “efficient frontier.”



3. Discussion

Portfolio optimization problems can be very compute intensive and in practice variants may need to calculate in real time because of market considerations.

This sequential computation is simple to implement in R and the parallel version is naturally expressed in the Sleigh function from Revolution Computing's ParallelR [2]. The program that studies this problem is remarkably short which illustrates the power of R as a statistical modeling languages and ParallelR as a parallel programming language.

Specifically we parallelized the outer loop of our simple code in which the random portfolios are constructed. In theory this can create a large number of tasks, each of which involves the evaluation of the reward and risk of a random portfolio (or a subset of them). In our model this evaluation is itself calculated from the given distribution functions by means of Monte Carlo procedure.

Our benchmark was run on a six node cluster. Each node was equipped with dual, dual-core AMD Opteron™ CPU chips. There were 24 cores available for the parallel application.

4. Benchmark Results

The benchmark we studied consists of 3600 portfolios and each of which contained 25 stocks. To calculate the reward and risk for each stock, we used a Monte Carlo approach. We randomly sampled 10000 points from each stock's distribution function. We then took the inner product of the stocks and their weights. This gave us the reward, which was used to calculate average reward and risk (variance of the reward). Figure 1 shows the amount of time it took to analyze 3600 portfolios with a varying number of Sleigh workers. Figure 2 shows the speedup of different number of Sleigh workers compares to sequential version program. There is a factor of 3.6 speedup with four Sleigh workers, a factor of 7 speedup with eight Sleigh workers, and a factor of 19 speedup with 24 Sleigh workers.



The following chart reflects the workload timings per the # of workers applied to the portfolio problem. As depicted in the matrix, the parallel activity represents an almost linear speedup when adding workers to the problem. This is further illustrated in the attached Figures 1 and 2, which graphs the workload against the number of workers applied to the parallel execution and load balancing applied when using Parallel R technology. For these problems, we are very satisfied with the linear speedup attributed to the cluster interconnect using simple Gigabit Ethernet. For problems with larger data sets, a high performance interconnect may be an important system component.

# of workers	Initialize time (seconds)	computation time (seconds)	Total time (init + comp)	Speedup (sequential divide by # of workers)
sequential	0.00	971.49	971.49	1.00
2	0.01	536.23	536.24	1.81
4	0.01	272.08	272.09	3.57
6	0.01	191.22	191.23	5.08
8	0.01	135.60	135.62	7.16
10	0.01	114.94	114.96	8.45
12	0.02	94.77	94.79	10.25
14	0.02	82.38	82.40	11.79
16	0.03	74.14	74.17	13.10
18	0.03	66.31	66.34	14.64
20	0.04	59.42	59.45	16.34
22	0.04	55.47	55.51	17.50
24	0.04	50.61	50.65	19.18



Table 1. Benchmark result

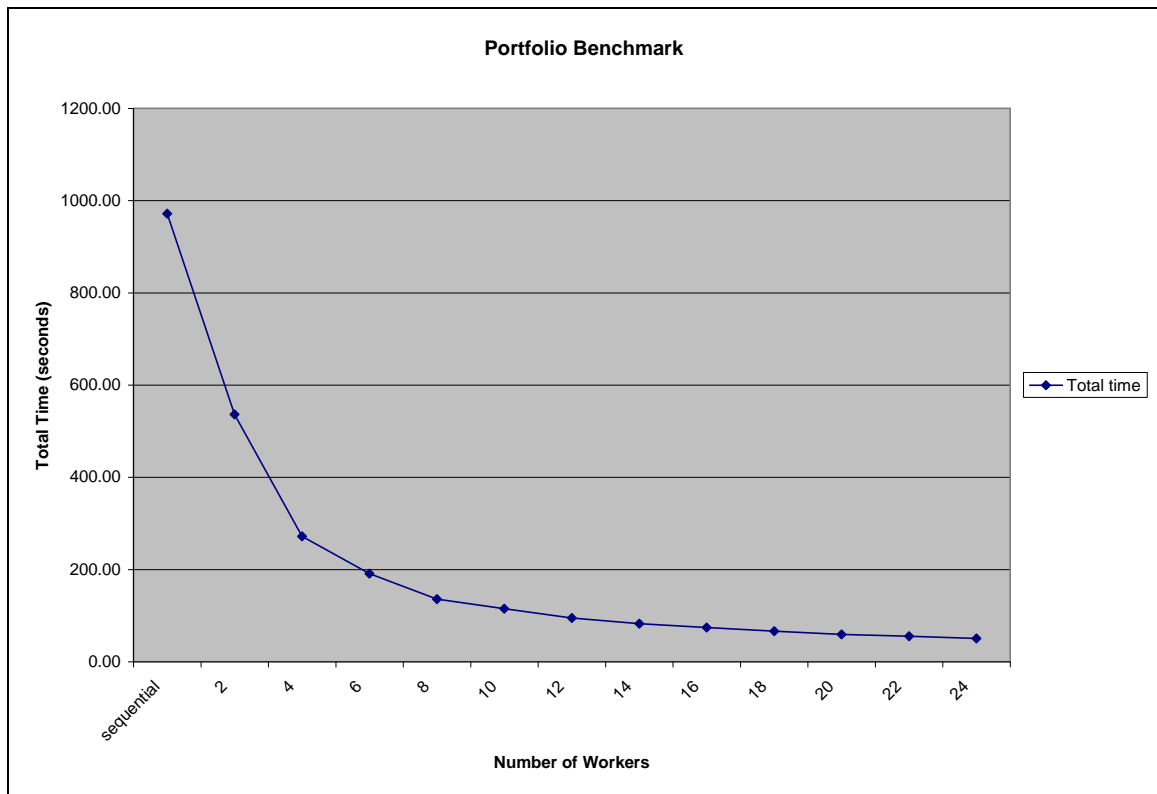


Figure 1. Amount of time versus the number of workers to finish a portfolio analysis.

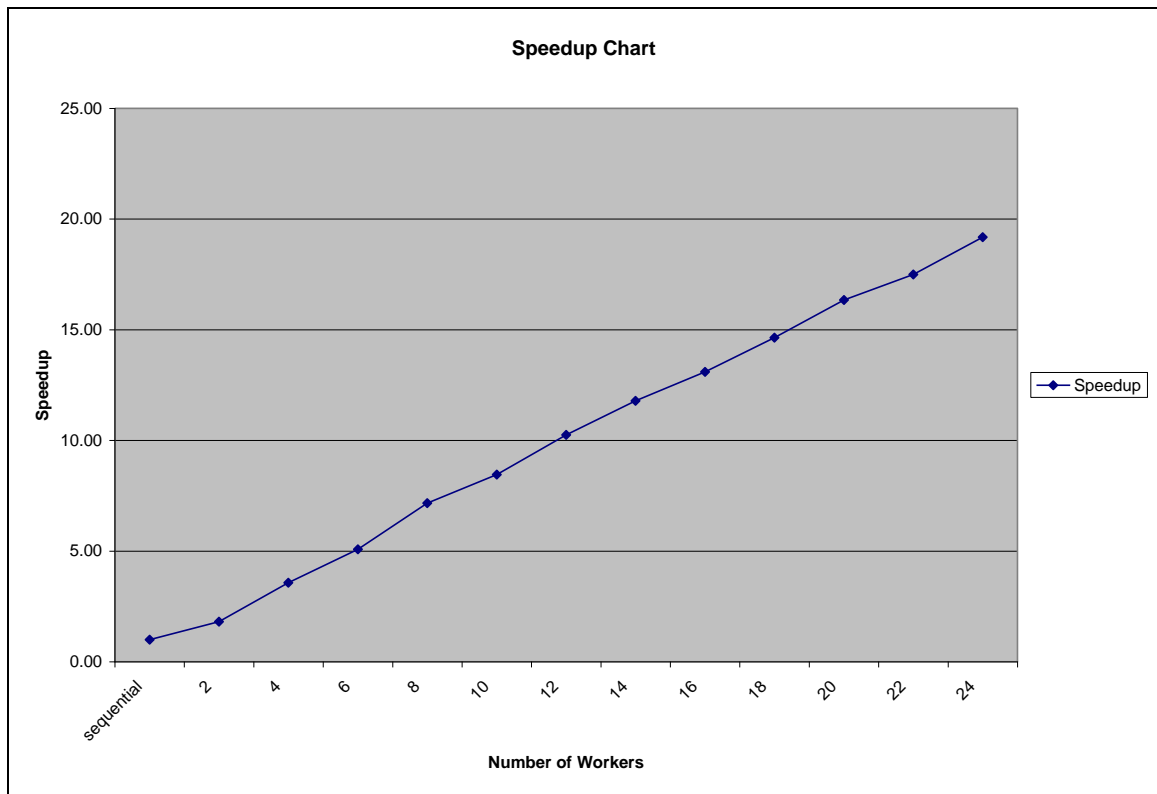


Figure 2. Amount of speedup versus the number of workers.

5. Conclusions

We draw three main conclusions from our study:

1. A multiprocessor can provide remarkable scalability up to the natural limits of the problem size.
2. By using the ParallelR package, an R user can easily create and run parallel R programs that make efficient use of multiprocessor hardware.
3. Multiprocessors with Revolution's ParallelR enable the quick and cost effective Monte Carlo simulations. Jobs are completed in a timely fashion.



7. References

[1] www.r-project.org

[2] www.revolution-computing.com

[3] http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/

[4] Monte Carlo: An Alternate Approach to Efficient Frontier
Balancing portfolio risk and return with efficient frontier

By: Jason McVean.

http://www.slb.com/content/services/software/valuerisk/expert_paper_monte_carlo.asp?seg=www.pipesim.com&printView=true&