Performance Evaluation of R with Intel Xeon Phi Coprocessor

[1*]Yaakoub El-Khamra, [2*]Niall Gaffney, [3*]David Walling, [4+]Eric Wernert, [5*•]Weijia Xu, [6+]Hui Zhang

[*]Texas Advanced Computing Center,
University of Texas at Austin
Austin, Texas USA
[1] yye00@tacc.utexas.edu, [2]ngaffney@tacc.utexas.edu,
[3]walling@tacc.utexas.edu, [5]xwj@tacc.utexas.edu,

[+] Pervasive Technology Institute,
Indiana University
Bloomington, Indiana USA
[4]ewernert@iu.edu, [6]huizhang@iu.edu

Abstract-- **Over the years, R has been adopted as a major data analysis and mining tool in many domain fields. As Big Data overwhelms those fields, the computational needs and workload of existing R solutions increases significantly. With recent hardware and software developments, it is possible to enable massive parallelism with existing R solutions with little to no modification. In this paper, we evaluated approaches to speed up R computations with the utilization of the Intel Math Kernel Library and automatic offloading to Intel Xeon Phi SE10P Co-processor. The testing workload includes a popular R benchmark and a practical application in health informatics. There are up to five times speedup gains from using MKL with a 16 cores without modification to the existing code for certain computing tasks. Offloading to Phi co-processor further improves the performance. The performance gains through parallelization increases as the data size increases, a promising result for adopting R for big data problem in the future.**

Keywords: Performance Evaluation; Statistic software; Parallel Computing; Intel Xeon Phi

I. INTRODUCTION

In many academic domains, the increasing sheer volume of the data availability presents exciting opportunities of new discoveries as well as significant challenge in analytic needs. The problem is often not characterized by the absolute number of bits of data, but the mismatch between the availability of the data and the existing analytic methods. On one hand, the Big Data problem will eventually bring a fundamental shift in computing needs for scientific data analysis and require new data computing models. On the other hand, advances in hardware and software tool development together could meet immediate needs in scaling up computation using existing solutions without costly efforts in developing new software tools. In this paper, we focus on how new hardware capable of providing massive parallelism can be utilized with a popular data analysis tool R and provide a performance comparison.

Due to its high extensibility and open source development, R has become a popular software tool and has been adapted to many scientific fields by researchers over the years. First created as an open source statistical computing language in 1993, the R ecosystem has evolved with features for statistical analysis, data mining and visualizations [1]. Furthermore, R enables users to develop domain focused libraries as packages which can be easily distributed and shared by the user community. There are 4,744+ packages available through the Comprehensive R Archive Network (CRAN). Those packages are developed to address problems from various research domains, such as social science[2], bioinformatics [3, 4], geosciences [5], business analysis[6] and in clinical science[7], to just list a few. Developed packages often utilize existing features implemented in the R core packages. Therefore, improvement and optimization made with basic computing tasks, such as matrix manipulations and linear algebra computations can benefit a large number of packages built upon them.

Although R is clearly a "high productivity" language, high performance has not been a development goal of R. Designed as a computing language with high level expressiveness, R lacks much of the fine grained control and basic constructs to support highly efficient code development. One approach to improve performance of R has been using dynamic libraries written in other programming language for expensive computations. While most features in R are implemented as single thread processes, efforts have been made in enabling parallelism with R over the past decade. Parallel package development coincides with the technology advances in parallel system development. For computing clusters, Li and Rossini first introduced a package, *rpvm*, based on Private Virtual Machine (PVM) [8]. Yu et al created a package *Rmpi* as an interface between R and Message Passing Interface in 2002 [9]. A popular parallel package, *Snow*, can utilize several low level communication protocols including MPI, PVM and socket connection [10]. There are also a number of

♦ Authors are listed in alphabetical order.

• Author of correspondence.

packages developed to utilize a multicore system including *fork*, *rparallel* and *multicore* [11, 12].

The parallel package development is initially fueled by the increasing computational complexity of the required methodologies. Common examples include matrices decomposition, linear system solver, Markov chain Monte Carlo simulation and bootstrapping. Since the complexity of those solutions typically ranges from linear to polynomial time to the input data, the increasing volume of big data present can increase the computational requirement dramatically. Recent effort has been made specifically for Big data including *R-pbd* package [13], and *Rhadoop* project [14]. A common approach in using R for big data is to break large data sets into chunks and runs each chuck in parallel sessions. Therefore, more parallel processes can improve the throughput of processing big data problems with R.

A common usage model is to rewrite some basic functions or processing flow with the corresponding parallel version provided by the parallel packages. Developing these packages often requires the user to have extensive knowledge in both existing R code as well as the parallel mechanism supported by the additional packages.

R enables linking to other shared mathematics libraries to speed up many basic computation tasks. One option for linear algebra computation is to use Intel® Math Kernel Library (MKL)[15]. MKL includes a wealth of routines to accelerate application performance and reduce development time such as highly vectorized and threaded linear algebra, fast Fourier transforms (FFT), vector math and statistics functions. Furthermore, the MKL has been optimized to utilize multiple processing cores, wider vector units and more varied architectures available in a high end system. Different from using parallel packages, MKL can provide parallelism transparently and speed up programs with supported math routines without changing code. It has been reported that the compiling R with MKL can provide three times improvements out of box [16].

Significant efforts have been made in developing accelerator cards that can easily increase the parallel processing potential in recent years. A general purpose graphic processing unit (GPGPU) extends parallel functions and technologies traditionally embedded in graphic processing units to handle more generic computations. Computational solutions can utilize the parallel features provided by GPU through programing interface such as OPENCL and CUDA. Most recently, the Intel Xeon Phi SE10P Co-processor (Xeon Phi) integrate 60 processing cores and 8GB memory in a single card. A critical advantage of the Xeon Phi co-processor is that, unlike GPU-based co-processors, the processing cores run the Intel x86 instruction set (with 64-bit extensions), allowing the use of familiar programming models, software, and tools. In addition to allowing the host system to offload computing workload partially to the Xeon Phi, it also can run a compatible program independently.

In this paper, we focus on how Xeon Phi could benefit the popular R packages in parallel processing. We evaluated two models of utilizing Xeon Phi: a) as a co-processor to offloading workload from host and b) offloading in combination with existing parallel packages available in R. We used the popular R-25 benchmark for performance evaluation while varying several factors including number of parallel processes and input data sizes. To study the potential benefit to the big data problem, we conducted preliminary runs with two practical problems.

II. Background and Related Work

A. Background on Intel Xeon Phi SE10P Co-processor

The basis of the Xeon Phi is a light-weight x86 core with in-order instruction processing, coupled with heavy-weight 512bit SIMD registers and instructions. With these two features the Xeon Phi die can support 60+ cores, and can execute 8 double precision (DP) vector instructions. The core count and vector lengths are basic extensions of an x86 processor, and allow the same programming paradigms (serial, threaded and vector) used on other Xeon (E5) processors. Unlike the GPGPU accelerator model, the same program code can be used efficiently on the host and the coprocessor. The same Intel compilers, tools, libraries, etc. used on Intel and AMD systems are also available for the Xeon Phi.

The Xeon Phi runs a lightweight BusyBox Operating System (OS), thereby making the Xeon Phi function as a separate Symmetric Multiprocessor (SMP). So, while the Xeon Phi can be used as a work offload engine by the host processors, it is also capable working as another independent (SMP) processor. In the latter mode MPI processes can be launched on the Xeon Phi and/or the E5 processors. In this "symmetric" mode the Xeon Phi appears as an extra node for launching MPI tasks.

These co-processors contain a large number of (relatively) simple cores running at lower frequency to deliver much higher peak performance per chip than is available using more traditional multi-core approaches. In the case of the Intel Xeon Phi SE10P Co-processor used in this paper, each coprocessor chip has a peak performance of roughly 1070 GFLOPS, approximately six times the peak performance of a single Xeon E5 processor, or three times the aggregate peak performance of the two Xeon E5 processors in each Stampede compute node. Each coprocessor is equipped with 8GB of GDDR5 DRAM with a peak bandwidth of 352GB/s, also significantly higher than the 51.2GB/s peak bandwidth available to each host processor chip.

B. *Related work in Enabling Parallelism with R*

There are nearly 30 packages that are related in enabling parallelism listed in CRAN task view for high performance computing. Among them, some are designed to provide explicit parallelism where users control the parallelization (such as *Rmpi* and *snow*); some are specially designed to provide implicit parallelism so that the system can abstract parallelization away (such as *multicore*); others are high level wrapper for other packages and intended to ease the use of parallelism, such as *snowfall* and *foreach*. Here we only reviewed some of the major packages that are directly related to our investigation. *Rmpi* is one of the earliest parallel package developed for R and is still used today and is built upon by other packages[9]. *Rmpi* provides an interface between R and Message Passing Interface and can link to an existing MPI implementation. The users need to link the R package with a MPI library installed separately, then the package enables users to use mpi-like code in R scripts. The package also includes parallel implementations of apply-like functions. The *snow* package utilizes *Rmpi* and several other existing parallel packages to expand the parallel support through a simple interface[10]. There are also several packages for exploiting parallelism within a single compute node. *Fork* is based on the system processing management interface to generate additional threads for computations[9]. *Pnmath* uses the OPENMP to implement many common mathematic functions to run in parallel. R/parallel provides support for running loops in parallel using a master-slave model[8]. *Multicore* package has been developed for utilize multiple cores available on the system. In addition, there are projects related with big data but not directly compared here, e.g. *pbdR*, *Rhadoop* etc. For a more comprehensive reviews of the parallel packages, interested reader can refer [17, 18].

III. EXPERIMENTAL DESIGN AND EVALUATION

A. *Objectives of the Experiment*

The primary objective for this investigation is to investigate the benefit of using Xeon Phi with R. We tested two usage models and compare to a baseline model of running R serially. The first model is to utilize the MKL on the host CPU only. We then experiment using the Xeon Phi processor to co-process with the host process. In the latter model, some of the computation can be passed to the Xeon Phi co-processor, also known as workload offloading. Theoretically, with an appropriate library, any computational workload could be offloaded to Xeon Phi for co-processing. Through the usage of MKL, we can run an R script in several different models without changing the script. Specifically, our tests include the following different models of computation.

- M1) Plain R without any parallelism. In this model, R was compiled and built with MKL, but with available threads set to 1. Therefore, all computations are run in single thread. This serves as a baseline for comparison.
- M2) R with MKL utilizing host CPU only. In this model, R was compiled and built with MKL as the library for the linear algebra computation. MKL can automatically utilize the available cores in the host system for parallel processing. In our test, the host computing node has 16 cores.
- M3) R with MKL utilizes both host CPU and Xeon Phi co-processor. In this model, R was compiled and built with MKL as well. However, the offloading to Xeon Phi is enabled. A configurable portion of the workload can be specified to use Xeon Phi, ranging from 0 percent to 100 percent.

For each of the above tests, we tested the factors affecting the performance including, functions affected by MKL, number of threads used, and size of input data. Furthermore we always used a "scatter" thread distribution on the Xeon Phi, as that always gave the best performance.

To evaluate how the automatic parallel offloading comparing to explicitly parallelism with other packages, batch runs of benchmarks are also conducted. When using with the MKL/Phi, the batch runs are initiated sequentially. While with other parallel packages, the batch runs are distributed out for explicitly parallel processing. Lastly, preliminary tests were also conducted with domain application to further evaluate potential benefit in practice.

B. *Testing Workloads:*

We first tested using the R-25 benchmark script available at http://r.research.att.com/benchmarks/. The testing script includes fifteen common computational tasks grouped into three categories: Matrix Calculation, Matrix function and "Programmation" (R scripting). The fifteen tasks are listed in Table-1. The choice of using the R-25 benchmark is mostly based on the fact that it appears to be the most popular, widely acknowledged benchmark. We are reaching out to R users on Stampede to improve their code performance with the Xeon Phi's and actively modifying code inside R modules to make use of the Xeon Phi. Early results of these efforts (including extensive scalability results) are promising and will appear in subsequent publications. The R-benchmark might not be the absolute best representative benchmark of Big-Data applications, it is however representative of typical R usage and does contain elements of Big-Data applications on Stampede.

In addition to testing the base R-25 benchmark script across the different usage models, we also explore the speed up gained when varying the size of the matrices and number of elements used in the benchmark to compare the results between host only MKL vs. Xeon Phi offloading.

We next turn R-benchmark into a wrapper function, and examine how to execute parallel calculations via the high-level interfaces provided by *multicor*e and *snowfa*ll packages for more compute-intensive tasks (e.g., when "nruns -> 100" or more in R-25 benchmark script).

We then investigate how parameters controlling offloading computations to Xeon Phi effect speed up of computation heavy matrix based computations. This investigation includes timing DGEMM based calculations while varying work offloading parameters to achieve maximum speed up for these operations.

TABLE I. TRANSLATION OF BENCHMARK NUMBER TO R-25 BENCHMARK DESCRIPTION FOR ALL R-25 PLOTS.

Task Number	R25 Benchmark Description
1	Creation, transp., deformation of a 2500x2500 matrix (sec)
2	2400x2400 normal distributed random matrix
3	Sorting of 7,000,000 random values
4	2800x2800 cross-product matrix
5	Linear regression. over a 3000x3000 matrix
6	FFT over 2,400,000 random values
7	Eigenvalues of a 640x640 random matrix
8	Determinant of a 2500x2500 random matrix
9	Cholesky decomposition of a 3000x3000 matrix
10	Inverse of a 1600x1600 random matrix
11	3,500,000 Fibonacci numbers calculation (vector calculation)
12	Creation of a 3000x3000 Hilbert matrix (matrix calculation)
13	Grand common divisors of 400,000 pairs (recursion)
14	Creation of a 500x500 Toeplitz matrix (loops)
15	Escoufier's method on a 45x45 matrix (mixed)
16	Total time for all 15 tests (not averaged)
17	Overall mean (sum of means of all tests)

Finally, we tested the benefit with offloading to Xeon Phi with one practical application in large-scale Bayesian multiple testing for time-series data. The project concerns nonparametric statistical methods and regularization techniques in high dimensional volatility matrix estimation. The computations are implemented in R, with each of 5 runs taking more than 5 hours to iterate through 5000 combinations of input parameters, and the whole batch processing originally taking more than 24 hours without exploiting parallelism of any kind.

C. *System Specification*

To evaluate the applicability of different methods for improving R performance in a high performance compute environment, we used the Texas Advanced Computing Center Stampede cluster. Stampede provides several different techniques for achieving higher performance computations which include using its Xeon Phi accelerators and/or NVIDIA Kepler 20 GPUs for large matrix calculations. In this test, each compute node has two Intel Xeon E5-2680 processors each of which has eight computing cores running @2.7GHz. There is 32GB DDR3 memory in each node for the host CPUs. The Xeon Phi SE10P Coprocessor installed on each compute node has 61 cores with 8GB GDDR5 dedicated memory connected by an x16 PCIe bus. The NVIDIA K20 GPUs on each node have 5GB of on-board GDDR5. All compute nodes are running CentOS 6.3. For this study we used the stock R 3.01 package compiled with the Intel compilers (v.13) and built with Math Kernel Library (MKL v.11). In addition to this version of R, a second version was compiled to run natively on the Xeon Phi coprocessors. The *gputools* package (version 0.28), which supplies the ability to offload linear algebra calculations to the NVIDIA Kepler GPUs, was linked with the CULA dense library free version R17 which in turn was linked to CUDA 5.0 toolkit.

IV. RESULTS AND DISCUSSION

A. *Summary of Different Methods Comparison*

Figure 1 shows the speed up of three different methods over the single thread execution for tasks listed in Table-1. The default R running with single thread is shown in blue as our base line for comparison with a speed up equal to one. We also exploited parallelism with GPU with the *gputools* module. However, the result (purple column) showed little benefit and even worse performance for some computing tasks. Despite our best efforts to improve the performance of the CUDA code in the *gputools* module, we ran into issues that required contacting the module developers. We chose to temporarily halt evaluating the *gputools* while we work with the module authors on improving performance. Since this is an early evaluation, this seemed appropriate: R users who get on the system today will see some transparent, immediate benefit from using the Xeon Phi cards for some codes, but will need to work on getting good performance from the *gputools* (beyond changing function calls). This transparent, immediate performance improvement is the focus of this evaluation.

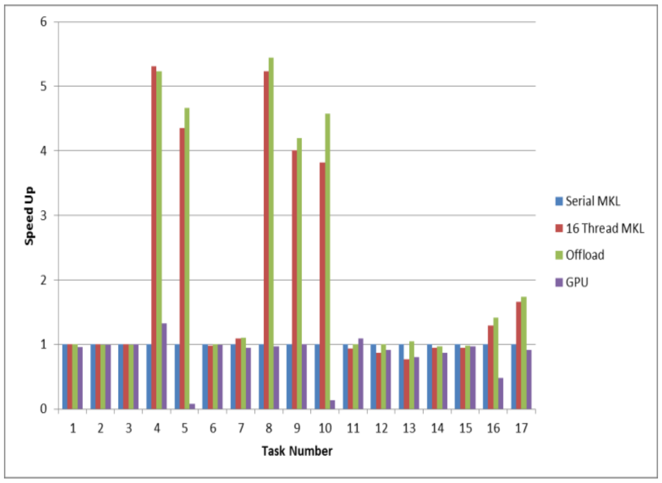


Figure 1. Plotted are the relative acceleration of the different threading and offloading techniques relative to results for the single threaded MLK benchmark results on the host.

The results of using MKL for automatic parallel execution with 16 threads are shown in red. There are five subtasks showing a significant benefit from the automatic parallelization with MKL. Those tasks include cross product between matrices, linear regression, matrix decomposition, computing inverse and determinant of a matrix. Other computing tasks received very little performance gains from parallel execution with MKL. The green column depicts the results when 70% workload is computed through Xeon Phi. Offloading computation to Xeon Phi can benefit the same set of computing tasks and has about 10% additional increase over MKL with host CPU alone. In both cases, the maximum speed up we observed is over five times more than single thread. While this is provides significantly faster wall clock times for those computing tasks, the speed up is sub-linear to the additional number of core/processes used. We think this is due to the extra overhead introduced with MKL library for automatic threading.

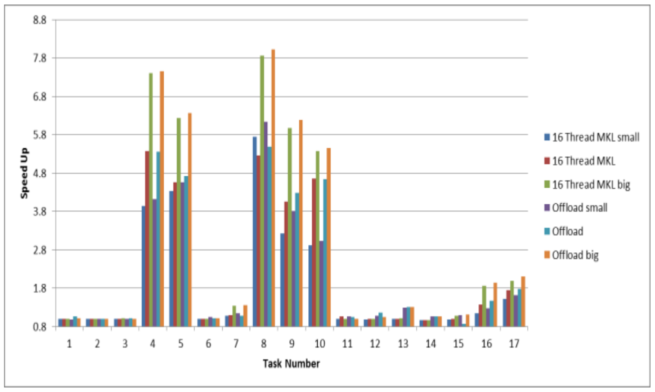


Figure 2. Comparison of speed up for different size of matrices or number of elements with each benchmark tasks.

Figure 2 compares the speed up gained over serial MKL for both host only vs. Xeon Phi offloading when varying the size for each dimension of matrices or number of elements of the R-25 benchmark tasks from half (small), normal and double (big). Offloading parameters were set at 30% host and 70% Xeon Phi. The results show that for certain tasks, speed up improves as the size of the data being processed increases.

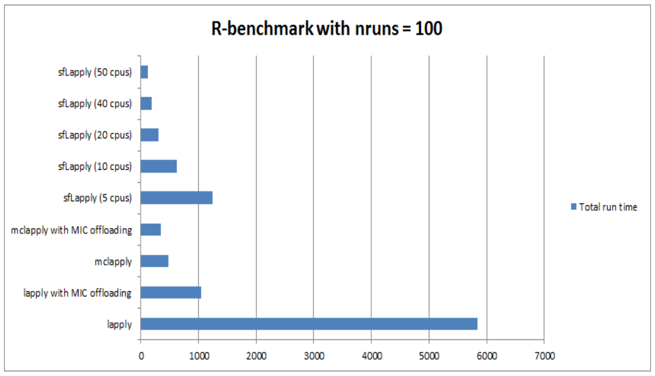


Figure 3. Comparison of computational time over 100 runs on R-benchmark using different parallel packages including Snowfall, multicore, and MKL.

To compare the parallelism supported by MKL and Intel Xeon Phi processor with other parallel packages, we conducted batch runs with the R-25 benchmark in parallel using Snowfall and Multicore package. The first five bars in Figure-3 shows the total running time using Snowfall package (sfLapply) with different number of processing cores which scales near linearly to the number of computing cores used. The bottom four bars compare using Multicore package (mclapply) with the serial run (lapply) with and without offloading to Intel Xeon Phi Processors. The comparison indicate that the benefit of using Xeon Phi processor diminish when used with other explicit parallelism based packages. For batch runs, explicit parallelism is more effective than using automatic parallelization provided through MKL.

B. Investigation of Different Models of Xeon Phi

The percentage of workload assigned to the Xeon Phi co-processor is an adjustable parameter. We investigated how different sizes and different workload sharing affect performance. Figure-4 shows the speed-up factors for different problem sizes at different workload sharing percentages. Note that matrices have to have a minimum size before automatic offloading starts paying off, in this case 8192x8192. Speedup is highest in the range of50%-70% workload on the coprocessor. As indicated in Figure 4 and Figure 5, the benefit of using parallelism through automatic offloading improves as the input data grows.

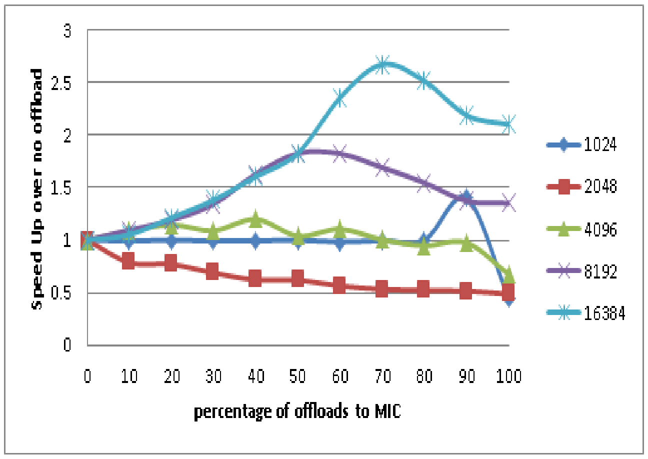


Figure 4. Speedup factor as a function of percentage of work on the coprocessor and matrix size. Note that forced offloading of small matrix operations reduces performance.

There are 60 physical processing cores available on each Intel Xeon Phi. Each computing core has four hardware threads. We can therefore run with 60 or 240 parallel threads on each Xeon Phi coprocessor. Furthermore, we can work share across the host the Xeon Phi, or across the host and two Xeon Phi coprocessors with different workloads as shown in Figure 5. Figure 4 shows the speedup factor for a basic matrix multiply in R with 1 thread on the host, 16 threads on the host, 60 threads on a Xeon Phi, 240 threads on a Xeon Phi, work-sharing at the 30% host (16 threads) 70% coprocessor (240 threads) sweet spot and work-sharing at the 20% host (16 threads) and 40% coprocessor (240 threads) for each of the two Xeon Phi's. The work-sharing sweet spots were chosen based on the data shown in Figure 5 and its equivalent for two Xeon Phi co-processors.

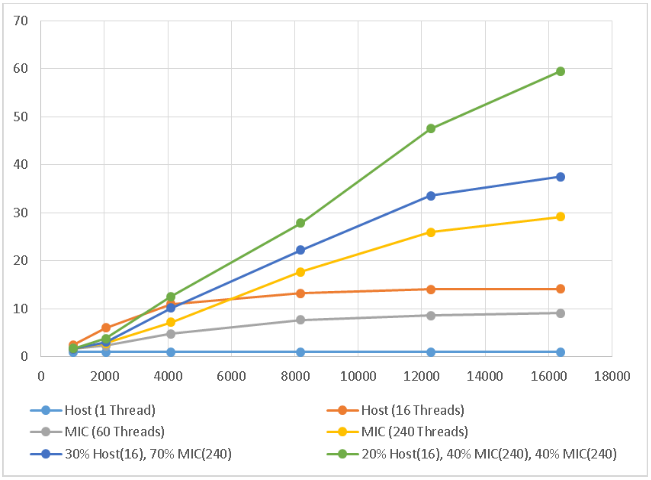
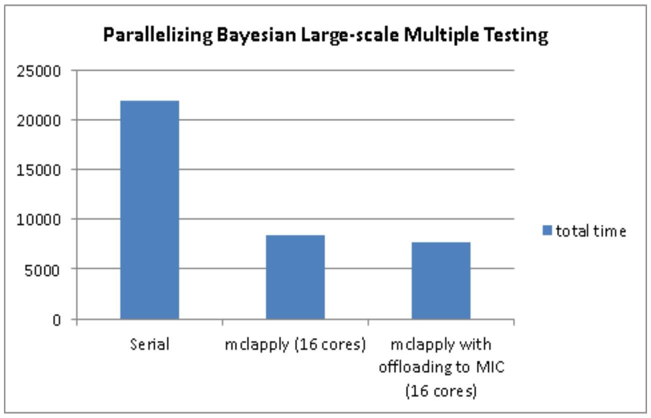


Figure 5. Speedup factors for different configurations. The highest possible speedup is almost 60X for the largest matrix size (16384x16384) with two Xeon Phi coprocessors (240 threads) each with 40% of the workload. Note that at small matrix sizes, the cost of offloading to the Xeon Phi is high enough to eliminate much of the speedup gained by using the Xeon Phi.

C. Exemplar Application Tests



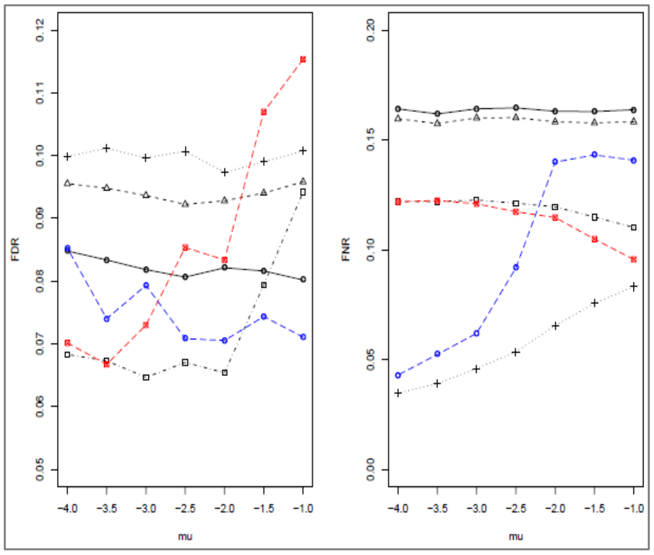


Figure 6 (Top) Parallelizing multiple testing tasks. Computations are distributed over multiple cores (using *multicore* package) and multiple nodes (using *snowfall* package). Enabling offloading to Xeon Phi coprocessors gives us an extra 10% speedup when using *multicore* package (from 8347.630s to 7675.636s). (Bottom) Simulations run to compare multiple methods including BH (Benjamini and Hochberg), AP (adaptive p-value procedure), OR (assume the true parameters are known), LIS (Sun and Cai), FB (full Bayesian method) and NPB (nonparametric Bayesian method). (Bottom-left: FDR (False Discovery Rate) versus μ; Bottom-right: FNR (False Non-discovery Rate) versus μ (control steps).

Multiple testing is one of the two fundamental statistical issues to address in the tasks of detecting and identifying the "outbreaks" in time series data. This study initiated by a group of researchers in IUPUI Mathematical science department investigates statistical solutions to examine and predict the outbreaks from data streams in Indiana Public Health Emergency Surveillance System, which are collected in an almost-real-time fashion and contain daily counts of ILI, ICD9, patent chief complaints, etc. The compute-intensive component of the project specifically considers the problem of massive data multiple testing under temporal dependence. The observed data is assumed to be generated from an underlying two-state hidden Markov model (HMM) - 'aberration' or 'usual'. Bayesian methods are applied to develop the independent testing algorithm by optimizing the false negative rate while controlling the false discovery rate. The testing procedures use a length of 1000 as burn-in period, with 5000 MCMC iterations. The original implementation were coded using R in a serial fashion and a 20-run procedure roughly took about 5 hours on desktop. The parallel solution run on Stampede exploits parallelization with *multicore* package to achieve a ~3-fold speedup, and automatic offloading to Xeon Phi coprocessors indicates another 10% performance improvement. While encouraging, we believe we can increase the performance boost and will discuss our approach and results in detail in subsequent publications.

As shown in Figure 6, computations are distributed over multiple cores (using *multicore* package) and multiple nodes (using *snowfall* package). Enabling offloading to Phi coprocessors gives us an extra 10% speedup when using *multicore* package (processing time was decreased from 8347.630s to 7675.636s). The study runs pairs of such simulations to compare multiple methods including BH (Benjamini and Hochberg), AP (adaptive p-value procedure), OR (assume the true parameters are known), LIS (Sun and Cai), FB (full Bayesian method) and NPB (nonparametric Bayesian method) [19].

V. CONCLUSION

In this paper, we report our experience in using new Xeon Phi co-processor to speed up the R computation. We tested workload offloading potential with a popular R benchmark and a preliminary test on a practical project. The results indicated that the performance improvement varies depends on the type of workloads and input data size.

As our initial investigation relied on the Intel Math Kernel library to coordinate usage of Xeon Phi co-processor, the types of parallel computations are limited by the features supported by the library. Therefore, we observed significant speed up with matrices operations, such as transformation, inverse and cross product. For this type of computations, the workload can be executed in parallel automatically and transparent to end users. Therefore, there is no need to modify the existing R script. Future programming efforts are required to enable parallelism manually for other types of computation.

The Xeon Phi coprocessor was designed for vector processing, therefore it came as no surprise that those functions can be vectorized and offloaded show promising performance boost. A key factor to this performance however is the size of the problem: offloading to the Xeon Phi can be costly, and for small problem sizes, any speedup gains can be cancelled out by offloading costs. On the other hand, if a function does not lend itself to vectorization or parallelism, there is nothing the Xeon Phi coprocessor can do that will improve performance without major code modification.

Our results showed a modest additional performance increase when offloading to the Xeon Phi. The performance improvement further increases as the size of the input data increases. However, the speedup is not a linear scale up with the number of threads used. This indicates an overhead cost in the process of parallelization. We also found an optimal usage is to offload about 70% workload with Xeon Phi.

We are continuing investigation on running R packages natively on the Xeon Phi instead of running as a co-processor. We are also interested in comparing the performance with other similar hardware technology such as using GPGPU and further exploring the use of existing parallel R packages to expand types of workloads benefiting from Xeon Phi offloading. We are encouraged by these initial results and feel there is great potential to utilize hybrid parallel models, running across multiple nodes and multiple Xeon Phi co-processors, in tackling large scale data problems with R.

ACKNOWLEDGEMENT

This project has been funded by grants from National Science Foundation. Experiments were conducted using Stampede computing cluster at Texas Advanced Computing Center.

REFERENCE

[1] R. C. Team, "R: A language and environment for statistical computing," ed. Vienna, Austria: R Foundation for Statistical Computing, 2013.

[2] J. FoX. (Aug. 18). *CRAN Task View: Statistics for the Social Sciences*. Available: http://cran.r-project.org/web/views/SocialSciences.html

[3] C. Gondro, L. R. Porto-Neto, and S. H. Lee, "R for Genome-Wide Association Studies," in *Genome-Wide Association Studies and Genomic Prediction*, ed: Springer, 2013, pp. 1-17.

[4] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, and J. Gentry, "Bioconductor: open software development for computational biology and bioinformatics," *Genome biology,* vol. 5, p. R80, 2004.

[5] E. Grunsky, "R: a data analysis and statistical programming environment–an emerging tool for the geosciences," *Computers & Geosciences,* vol. 28, pp. 1219-1222, 2002.

[6] A. Ohri, *R for business analytics*: Springer, 2013.

[7] S. Pyne, X. Hu, K. Wang, E. Rossin, T.-I. Lin, L. M. Maier, C. Baecher-Allan, G. J. McLachlan, P. Tamayo, and D. A. Hafler, "Automated high-dimensional flow cytometric data analysis," *Proceedings of the National Academy of Sciences,* vol. 106, pp. 8519-8524, 2009.

[8] M. Li and A. J. Rossini, "{RPVM}: Cluster Statistical Computing in {R}," *R News,* vol. 1, pp. 4-7, 2001.

[9] H. Yu, "Rmpi: Parallel Statistical Computing in R," *R News,* vol. 2, pp. 10-14, 2002.

[10] A. J. Rossini, L. Tierney, and N. Li, "Simple parallel statistical computing in R," *Journal of Computational and Graphical Statistics,* vol. 16, 2007.

[11] G. Vera, R. Jansen, and R. Suppi, "R/parallel - speeding up bioinformatics analysis with R," *BMC Bioinformatics,* vol. 9, p. 390, 2008.

[12] G. R. Warnes. (2007). *fork: R functions for handling multiple processes*. Available: http://cran.r-project.org/web/packages/fork

[13] D. Schmidt, W.-C. Chen, G. Ostrouchov, and P. Patel, "A Quick Guide for the pbdBASE package," *R Vignette, URL http://cran. r-project.org/package=pbdBASE,* 2012.

[14] J. Adler, *R in a Nutshell*: O'Reilly Media, 2010.

[15] Intel. *Intel® Math Kernel Library 11.0*. Available: http://software.intel.com/en-us/intel-mkl

[16] A. M. Wilson. (2012). *Speeding up R with Intel's Math Kernel Library (MKL)*. Available: http://www.r-bloggers.com/speeding-up-r-with-intels-math-kernel-library-mkl/

[17] 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,* vol. 47, 2009.

[18] D. Eddelbuettel. (2013). *CRAN Task View: High-Performance and Parallel Computing with R*. Available: http://cran.r-project.org/web/views/HighPerformanceComputing.html

[19] X. Wang, A. Shojaie, and J. Zou , "Bayesian Large-Scale Multiple Testing for Time Seires Data," Manuscript in press, 2013.