Block Cyclic Distribution of Data in pbdR and its Effects on Computational Efficiency

REU Site: Interdisciplinary Program in High Performance Computing

Matthew G. Bachmann[1], Ashley D. Dyas[2], Shelby C. Kilmer[3], and Julian Sass[4],
Graduate Research Assistant: Andrew Raim[4],
Faculty Mentors: Nagaraj K. Neerchal[4] and Kofi P. Adragni[4]
Clients: George Ostrouchov[5] and Ian F. Thorpe[6]

[1]Department of Mathematics, Northeast Lakeview College
[2]Department of Computer Science, Contra Costa College
[3]Department of Mathematics, Bucknell University
[4]Department of Mathematics and Statistics, University of Maryland, Baltimore County
[5]Oak Ridge National Laboratory
[6]Department of Chemistry and Biochemistry, University of Maryland, Baltimore County

Technical Report HPCF–2013–11, `www.umbc.edu/hpcf` > Publications

Abstract

Programming with big data in R (pbdR), a package used to implement high-performance computing in the statistical software R, uses block cyclic distribution to organize large data across many processes. Because computations performed on large matrices are often not associative, a systematic approach must be used during parallelization to divide the matrix correctly. The block cyclic distribution method stresses a balanced load across processes by allocating sections of data to a corresponding node. This method achieves well divided data that each process computes individually and calculates a final result more efficiently. A nontrivial problem occurs when using block cyclic distribution: Which combinations of different block sizes and grid layouts are most effective? These two factors greatly influence computational efficiency, and therefore it is crucial to study and understand their relationship.

To analyze the effects of block size and processor grid layout, we carry out a performance study of the block cyclic process used to compute a principal components analysis (PCA). We apply PCA both to a large simulated data set and to data involving the analysis of single nucleotide polymorphisms (SNPs). We implement analysis of variance (ANOVA) techniques in order to distinguish the variability associated with each grid layout and block distribution. Once the nature of these factors is determined, predictions about the performance for much larger data sets can be made. Our final results demonstrate the relationship between computational efficiency and both block distribution and processor grid layout, and establish a benchmark regarding which combinations of these factors are most effective.

Key Words: pbdR, Block Cyclic Distribution, Grid Layout, Block Size, PCA, covariance, correlation

1 Introduction

The ability to analyze large quantities of data is becoming increasingly necessary as technology to accquire data improves. Our capacity to accquire data has far surpassed our capability to understand it. It is estimated that humankind is able to store 295 exabytes of informtion. Big data is more than just the buzz word of the day: The White House, National Institute of Health, National Science Foundation, and many other prominent organizations all have proposed initiatives emphasizing the big data problem (TBD: citation(s) needed). The scope of this note is how interpeting big data in efficient ways will facilitate progress toward a better understanding of literally any topic that has substantial observational data.

Programming with Big Data in R (`pbdR`) contains numerous packages that allow for manipulation of this type of data [5]. Section 2.1 gives an overview of such libraries. The underlying process `pbdR` uses to organize big data amongst processes is described in Section 2.2. Breaking up a matrix of large dimensions can be dangerous when doing computations that are not embarrassingly parallel. `pbdR` uses block cyclic distribution to meticulously distribute the blocks of data, shown in Figure 2.1, so that the evaluation is not compromised. The process is quite complicated, however block cyclic distribution is primarily implemented without user interaction. `pbdR` integrates simple function calls, see Section 3.2, to make it easy on the user.

Although any block cyclic distribution can be implemented without the user ever interacting with the parameters, `pbdR` also allows the user to alter parameters manually. The different parameters used in this note are discussed in Section 3.3. The primary focus here is to look at how manually adjusting block cyclic distribution can improve efficiency. A thorough test of these parameters is seen in Section 4.1. The results give insight to the best combinations of parameters to maximize efficiency.

Sections 2.3, 3.4, and 4.2 all provide an opportunity to view the block cyclic distribution method applied to a data set from the lab of Dr. Ian Thorpe. The data set we look at is very large and gathered over less than a second. An application to chemistry provides a great moment to stress that this data, if measured over minutes, hours, or even days, has the potential to be incomprehensibly large. In this note, there are 3100 snapshots in intervals of 10 picoseconds containing a 531×3 matrix per snapshot. A brief analysis relevant to this data set is outlined and the results are displayed in Section 4.2. This application only begins to explain the need for computational efficiency in big data.

2 Background

2.1 `pbdR` and Supporting Libraries

Programming with big data in R (`pbdR`) is a package used to implement parallel computing in the statistical software R [2]. `pbdR` consists of several subpackages such as `pbdMPI`, `pbdDMAT`, which build upon established libraries such as MPI, ScaLAPACK, BLAS, and are accessible on the Comprehensive R Archive Network (CRAN). These subpackages assist with the use of different types of mathematical methods on parallel processors. Parallel implementation is

Figure 2.1: Example of block distribution and processor grid layout. TBD: Figure was borrowed from [5]

primarily done through Message Passing Interface (MPI), the widely considered standard for communication amongst processes [1]. As opposed to the master-slave paradigm, `pbdR` uses a single program multiple data (SPMD) parallelism intended by MPI. Further, `pbdR` effectively eliminates the need for a manager node, and all processors are autonomous. This MPI structure is defined by `pbdR` in `pbdMPI`. The packages within `pbdR` include dense linear algebra packages with specific methods that allow complex computations to be implemented on many processors. Of these methods for parallel computing, block cyclic matrix distribution is included.

2.2 Block Cyclic Distribution

Built upon ScaLAPACK, `pbdDMAT` allows for a matrix to be distributed amongst a group of processes by first breaking the matrix into different block sizes. The dimension of these blocks ($b \times b$) has an effect on efficiency of computation. Block sizes can be both inefficiently large or inefficiently small, with respect to the size of the overall matrix. Additionally, block sizes that divide the number of rows and columns evenly are likely more efficient. Figure 2.1 shows, in the case of a 9×9 matrix, $b = 2$ causes some additional smaller matrices (1×2, 2×1, and 1×1) to be given to all processes except 3 and 5 (purple and orange in figure). This division will cause processes 0, 1, 2, and 4 to have more computations to perform than the others. Had the block size been, for example, $b = 3$, the block sizes would divide the over all dimension evenly and it is probable that there would be speed up in compution time.

Once `pbdR` breaks a matrix into blocks, these blocks must be distributed amongst different processes. Therefore, block cyclic distribution employs a grid layout ($r \times c$) of processes to

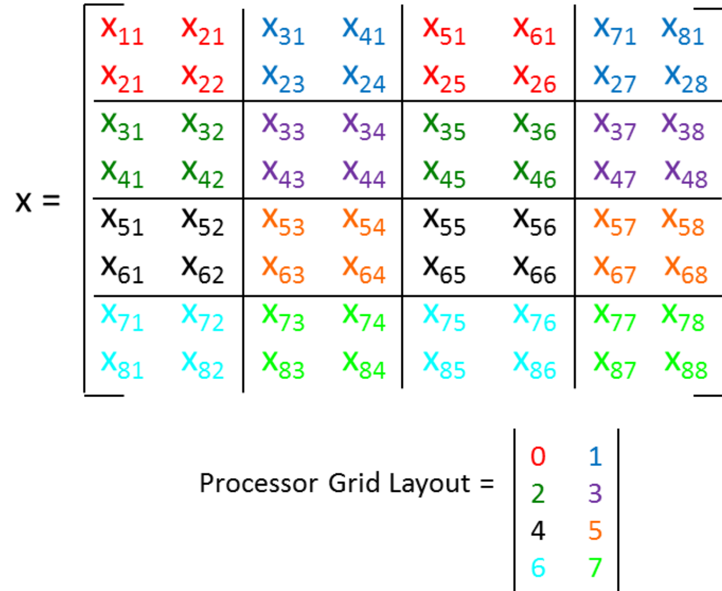

Figure 2.2: Example of block distribution and processor grid layout. TBD: Figure was borrowed from [5]

partition the blocks. Processor grid layout attempts to maximize efficiency by determining which groups of blocks are given to each process, and how the layout of data given to each process is presented. In Figure 2.1, note that processor grid layout is represented by groups of color. For example, process 0 receives the red blocks of data, process 1 receives the blue blocks, process 2 receives the green block, and continues in $r \times c = 2 \times 3$ fashion till it reaches the next group of each color. The same process is shown in Figure 2.2, though here we see an example of when load balancing is perfectly even. Preliminary studies by our client Dr. G. Ostrouchov demonstrate that there exists a relationship between the efficiency of parallel execution and both grid layout and block size.

2.3 Application to Protein Movement

In an application of our study, we use data from the lab of Dr. Ian Thorpe. This data focuses on the movement of the NS5B protein created by Hepatitis C virus polymerase. This polymerase is responsible for the replication of the viral RNA of the virus. The data is formatted as 3100 snapshots of the protein at intervals of 10 picoseconds. Each snapshots contains the x-, y-, and z-coordinates of different amino acid residues in 8290 different atoms in the protein. In order to reduce the size of the data set, we organize the data by taking the average x-, y-, and z-coordinates of each atom in the 531 residues, creating a 531×3 matrix for every snapshot. We then are able to combine all of the data into one 531×9300 data matrix.

2.4 Computational Environment

This efficiency study is made possible by the cluster tara in the UMBC High Performance Computing Facility (`www.umbc.edu/hpcf`). Tara consists of 86 nodes: 82 compute nodes, 2 develop nodes, 1 user node, and 1 management node. All nodes have two quad-core Intel Nehalem X5550 processors and 24 GB memory. All nodes are connected by a quad-data rate InfiniBand interconnect. We may access files, write and compile code, and submit jobs from the user node. Documentation on running `pbdR` programs on tara is available in [3].

3 Methodology

The factors of variability we define in our method are based on prelininary studies presented to us by our client Dr. G. Ostrouchov. This research makes us aware of four factors that directly affect efficiency of parallel programming in `pbdR`: number of observations ($n \times k$), number of processes (p), processor grid layout, and block size. We study more closely the relationships between computational speed and these four factors in order to develop a benchmark for further work with big data and block cyclic distribution.

3.1 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a statistical method used most commonly for dimensionality reduction; see [4] for a general overview. This multivariate analysis can be used to create a lower dimensional picture of a data set by omitting nonsignicant principal components. PCA determines the directions of maximum variability. When the first few principal components contribute to the majority of variability in the data, these components are used to transform the data. PCA can be defined as an orthogonal linear transformation of data. Performing PCA on a large data set provides opportunities for parallelization through block cyclic distribution.

Here PCA is implemented on the $k \times k$ sample covariance matrix

$$\mathbf{S} = \frac{1}{n-1}\mathbf{Y}^T \left(\mathbf{I} - \frac{1}{n}\mathbf{J}\right)\mathbf{Y},$$

where $\mathbf{J} = \mathbf{1}\mathbf{1}^T$ is a matrix of all ones, and

$$\mathbf{Y} = \begin{bmatrix} y_{11} & \cdots & y_{1k} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nk} \end{bmatrix}$$

is a matrix of n observations each having k attributes. $\mathbf{S}$ is positive semi-definite and therefore can be diagonalized by

$$\mathbf{\Lambda} = \mathbf{O}^T\mathbf{S}\mathbf{O},$$

where $\mathbf{O}$ is an orthogonal matrix; this decomposition is possible with any positive semi-definite symmetric matrix. Each row of $\mathbf{O}$ is an eigenvector of $\mathbf{S}$. These eigenvectors are

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}^T$$

(a) 1×6

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

(b) 2×3

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

(c) 3×2

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

(d) 6×1

Figure 3.1: Possible grid layouts for 6 processes.

called principal components. $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues of $\mathbf{S}$. The number of significant eigenvalues determines the true underlying dimensionality of the data.

`pbdR` has integrated both a PCA and covariance function defined by `prcomp` and `cov` respectively. Because both functions are important in analyzing big data, they provide a great way to test block cyclic distribution on data of very large dimension. Therefore, `prcomp` is timed in this note to determine how different parameters of block cyclic distribution affect performance.

3.2 Block Cyclic Distribution in `pbdR`

The `pbdDMAT` sub-package in `pbdR` provides the `ddmatrix` function for applying block cyclic distribution to matrices for higher level programming [5]. By implementing the `ddmatrix` function defined in `pbdDMAT`, b is easily manipulated. Furthermore, $r \times c$ can be arranged using `init.grid`, also found in `pbdDMAT`.

The program used first generates independent random data on each process. This process assists in avoiding memory limitations imposed by the hardware. By changing the class of the data from “matrix” to “ddmatrix” we communicate that each process has a portion of a distributed matrix. This is crucial because the functions of `pbdDMAT` will only work on a distributed matrix. This distribution, however, is not in the correct format to begin computation. As stated, much care must be taken before beginning computation to ensure that the evaluation is correct. The `redistribute` function allows the format of distribution to be changed to a block cyclic format, ready for computation.

$r \times c$ must match p in the sense that the rc divide p evenly. So for example, in the case $p = 6$, we can study $r \times c = 1 \times 6$, 2×3, 3×2, and 6×1, as shown in Figure 3.1.

3.3 Efficiency Studies

As discussed in Section 3.1, we use the `prcomp` function provided by `pbdR` to perform our efficiency study. Our initial runs are performed to study the effects of $n \times k$. Results from this study are expected to have a predictable effect on efficiency. We perform the study by holding the block size constant at $b = 2$, testing for $n = 1000, 3000, 5000, 10000, 30000, 50000$, with $k = \frac{n}{10}$. With $p = 64$, we can compare grid layouts of $r \times c = 4 \times 16$, 8×8 and 16×4. If the results show $n \times k$ having a predictable effect on efficiency, as expected, we can observe

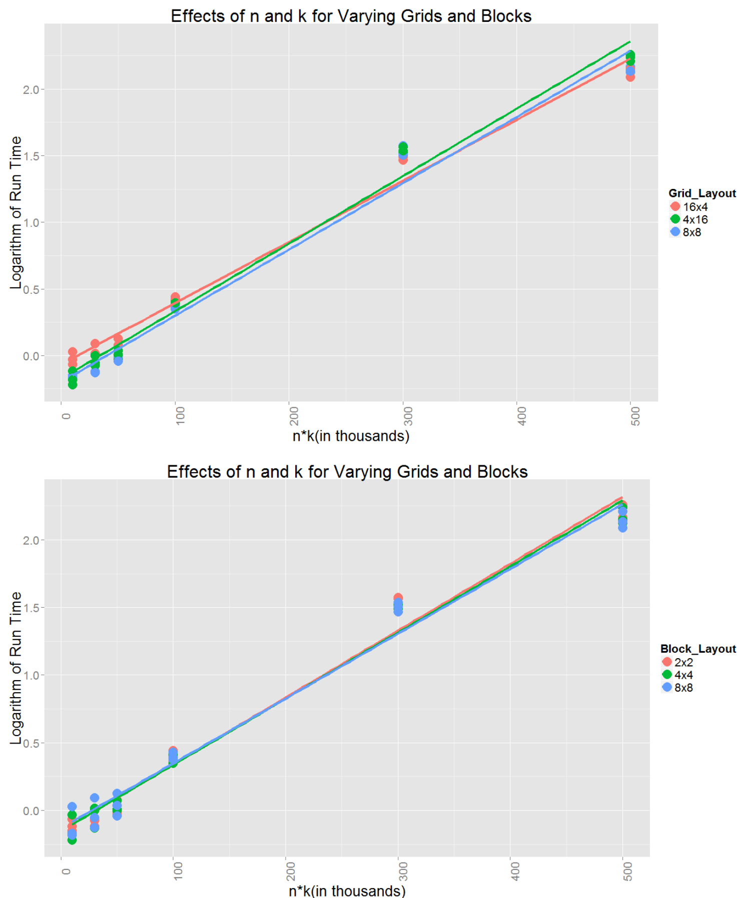

Figure 3.2: Effects of $n \times k$ on run time.

other factors while n and k remain constant.

The results from this $n \times k$ study are shown in Figure 3.2. In the $n \times k$ study we observe times that appear to have a logarithmic relationship. By taking the logarithm of the times we observe a linear trend shown in Figure 3.2. These results confirm the anticipated predictability of the dimensions affect on efficiency. This conclusion allows further study to hold $n \times k$ constant and focus on the variability associated with $r \times c$ and b. We explore this idea further in Section 4

3.4 PCA and Correlation Matrix for SNPs

In the application of block cyclic distribution and grid layout, we create a covariance matrix from the data on protein movement. From there we implement PCA on the correlation matrix. We were also able to compute a 531×531 correlation matrix from the movement data, showing how each residue correlates to the other residues. From there, we create a level plot of the correlation matrix in order to see correlations of the data with itself, with the red showing negative correlation and the blue showing positive correlation. From there

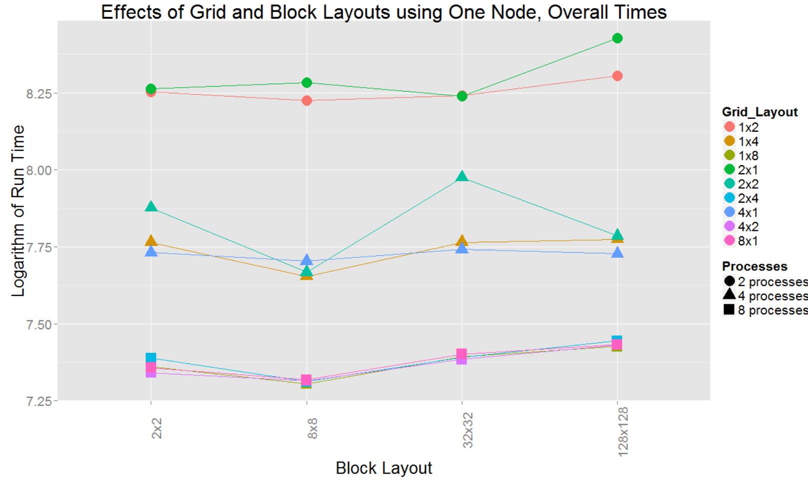

Figure 4.1: Overall run time for 1 node: 5120×5120.

we use a Fisher Transform, defined as

$$z = \ln\left(\frac{1 + R_{ij}}{1 - R_{ij}}\right),$$

where R_{ij} represents the (i, j)th entry in the correlation matrix. This transformation is used to find which correlations were statistically significant to the movement of the protein. On the next level plot of the data, we can grey out all data points with a Fisher Transform value of less than or equal to 2, which shows the statistical insignificance of the correlation to overall movement.

4 Results

4.1 Efficiency Study Results

Because Figure 3.2 shows a predictable trend, from there we can hold n and k constant in order to analyze the more pertinent factors: grid layout and block size. In order to produce the widest range of timing resutls possible, we determine our constant n and k as the largest possible values within memory restrictions. In our first study we analyze the effects on one node of processes in an effort to avoid noise caused by communication time. We determine that with one node, the largest matrix `R` will allow is approximately 250 million entries. In order to obtain a wide range of data points, we look at all $r \times c$ combinations for 2, 4, and 8 processes. For example, the grid layouts for 4 processes include a 1×4, a 2×2, and a 4×1 grid. Additionally we wish to study both small and relatively large block sizes. We study $b = 2$, 8, 32 and 128. However, due to proper load balancing, we need to ensure that the combination of block size and grid layout divdes the matrix evenly:

$$n = m\,b\,r, \quad k = n\,b\,c,$$

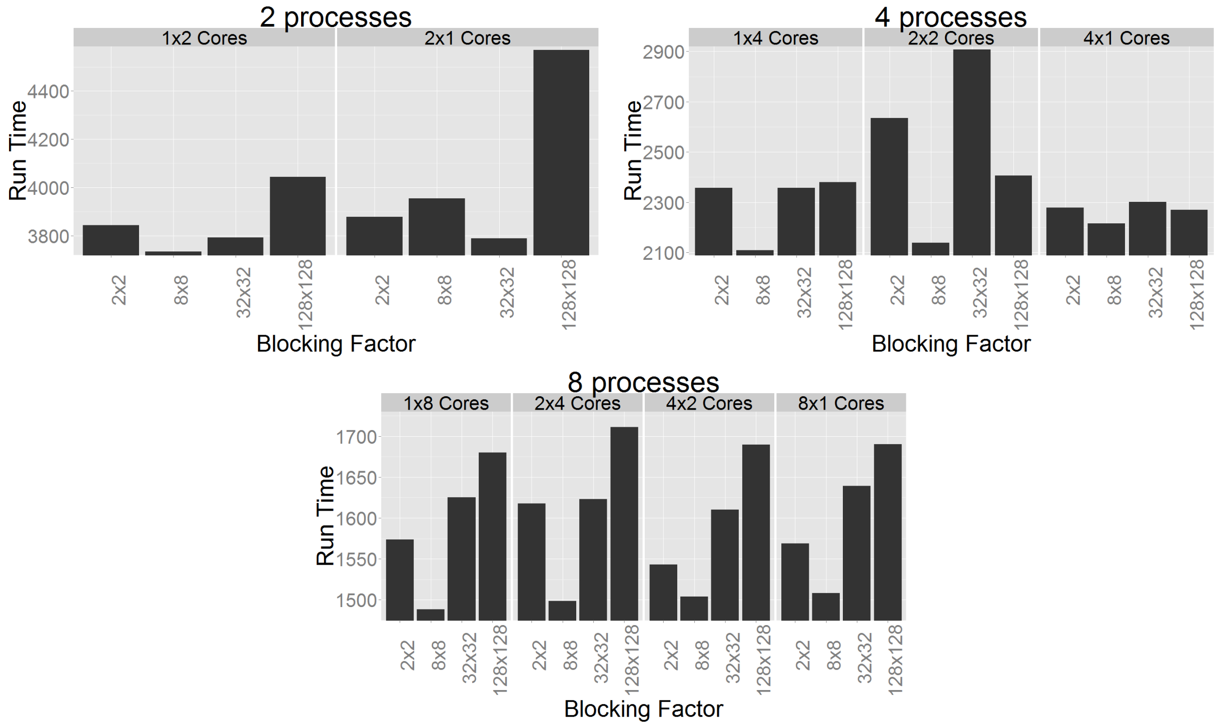

Figure 4.2: Overall run time for 1 node: 51200×5120.

where m, n are some constants. From this we choose to use a 51200×5120 matrix, keeping with our desire for n to be ten times larger than k. Figure 4.2 displays the overall run time for this study.

The results shown in Figure 4.2 clearly demonstrate that in all cases except the 2×1 grid layout, the 8×8 blocking factor is the most efficient, often substantially so. Both the computations of the covariance matrix and the principal component analysis are included in these results. From this it would be easy to conclude that $b = 8$ is most efficient, essentially independent of grid layout. However, Figure 4.3, which display the covariance and PCA run times seperately, give a more insightful description of how the factors are truly affecting efficiency.

When observing PCA and covariance together, the results are more complex. Our timing results for PCA alone (excluding the covariance matrix computation) look very similar to Figure 4.2. This is to be expected because the PCA computation takes about 10 times longer than the covariance computation. This imbalance causes the trends in the PCA results to dominate the overall results. The times in Figure 4.3 do not seem to have an easily distinguishable pattern.

As a result of finding very different trends for PCA and covariance, it is crucial to investigate the cause. The fundamental difference between these computations is the dimension of the matrix they compute. The covariance computation accepts an $n \times k$ data matrix and reduces it to a $k \times k$ positive semi-definite covariance matrix. PCA, on the other hand, performs its computation on the new $k \times k$ matrix. At first glance, this matrix dimension

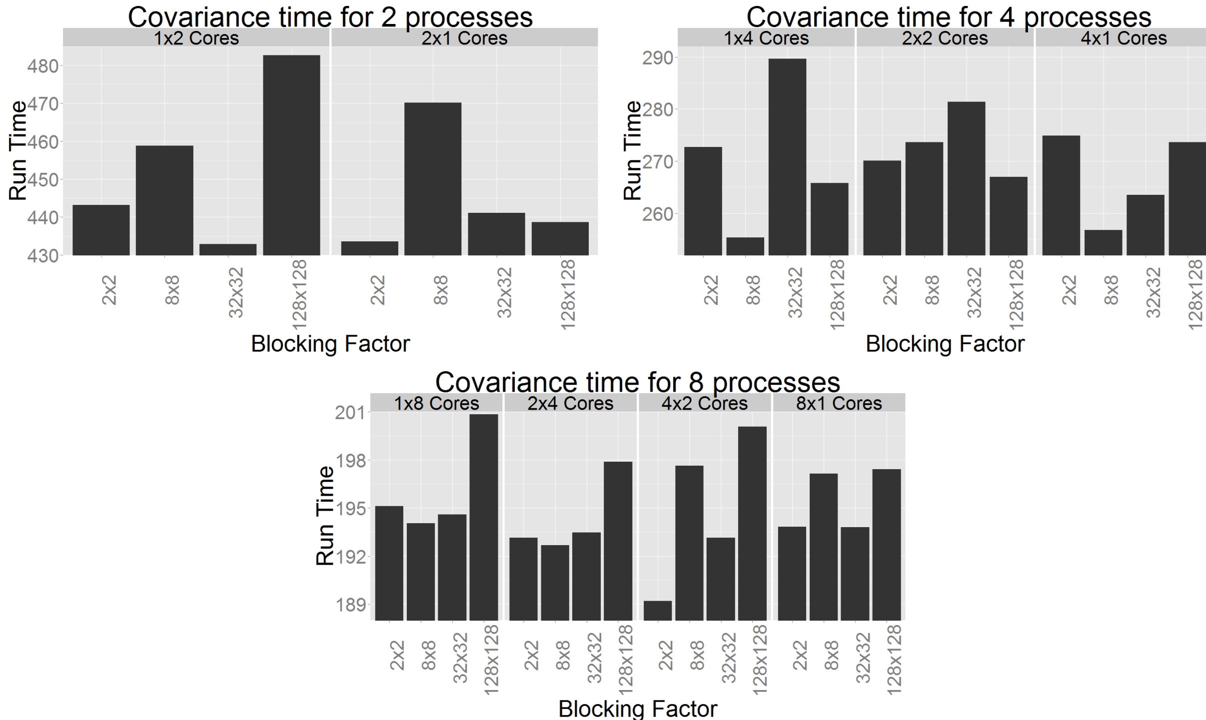

Figure 4.3: Covariance run time for 1 node: 51200×5120.

characterstic stands out as a possible explanation for the different trends. Nonetheless, before making any definitive conclusions, we extend our study to a larger $n \times k$ case in hopes that the trends will become more defined.

By extending our study to four nodes ($p = 32$), more memory from three additional nodes grants the ability to compute a matrix with nearly four times more entries. R, however, restricts vector allocation beyond 4 GB. Therefore, the four node study is restricted to a maximum dimension of 81920×8192. Although this is not four times larger, the 81920×8192 matrix is about 670 million entries whereas the 51200×5120 was only about 250 million. We still see a reasonable increase in size; this should surely be adequate to add clarity to our trends. Figure 4.4 now shows a similar, yet more distinct relationship.

4.2 Protein Data Analysis Results

From the plots of the principal component analysis, it is clear that there are three major principal components of the data. This shows that there are three axes that the data can be rotated around such that we cannot see the insignificant correlations. From the level plots of the data, we are clearly able to distinguish which correlations are statistically significant to the overall movement of the protein. Since there are much fewer points of significant correlations, we can state that few residues give substantial contribution to movement. This gives new insight to the study of the Hepatitis C virus, since very few of the protein components contribute to its movement throughout the body.

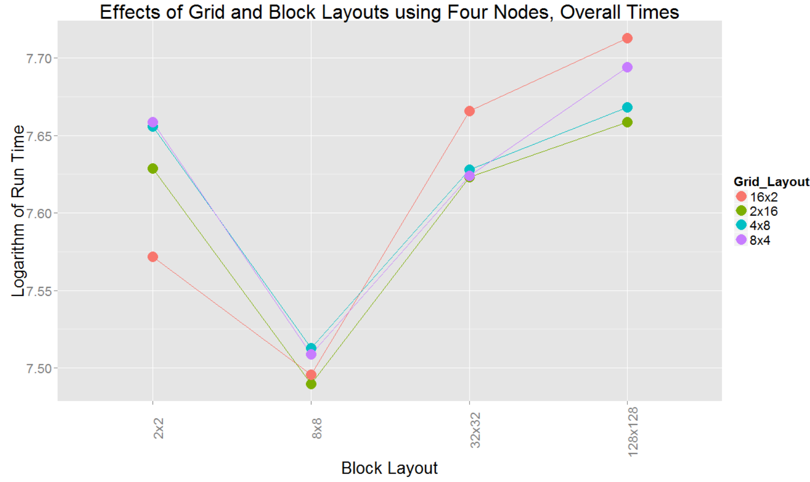

Figure 4.4: 4 node overall.

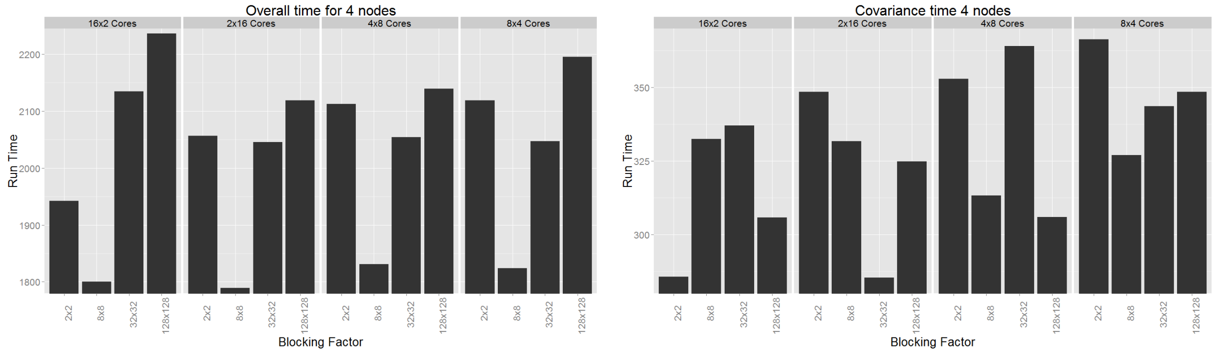

Figure 4.5: Overall and Covariance times for 4 node study.

5 Conclusions

One potential for further research involves the n and k values we were able to use for the four node study. Ideally we should have been able to use vectors four times as large, given that four nodes provides four times as much memory as does one node. However, due to vector allocation limits imposed by R, we were only able to use vectors up to 4 GB of data. This severely limited our study. We would like to be able to investigate the potential of parallelizing vector allocations for the k vector in order to be able to handle even larger data matrices. Perhaps for even larger matrices our results would be quite different.

Based on our findings in Section 4, we notice that 8×8 block sizes consistently perform the most efficiently for the PCA and overall run time studies. We analyze the relationship more quantitatively through a statistical technique called analysis of variance (ANOVA). ANOVA allows us to quantify how much variability is being contributed by each factor. According to our ANOVA results in Table 5.7, we notice that the effects of grid layout on computational efficiency for PCA are much less significant than those of block layout. For

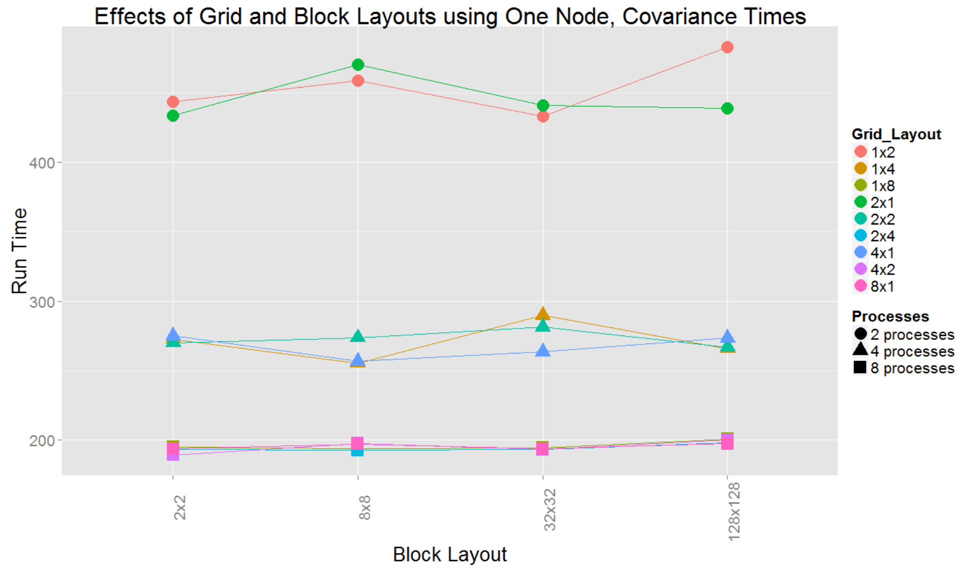

Figure 4.6: Covariance times for 1 node study.

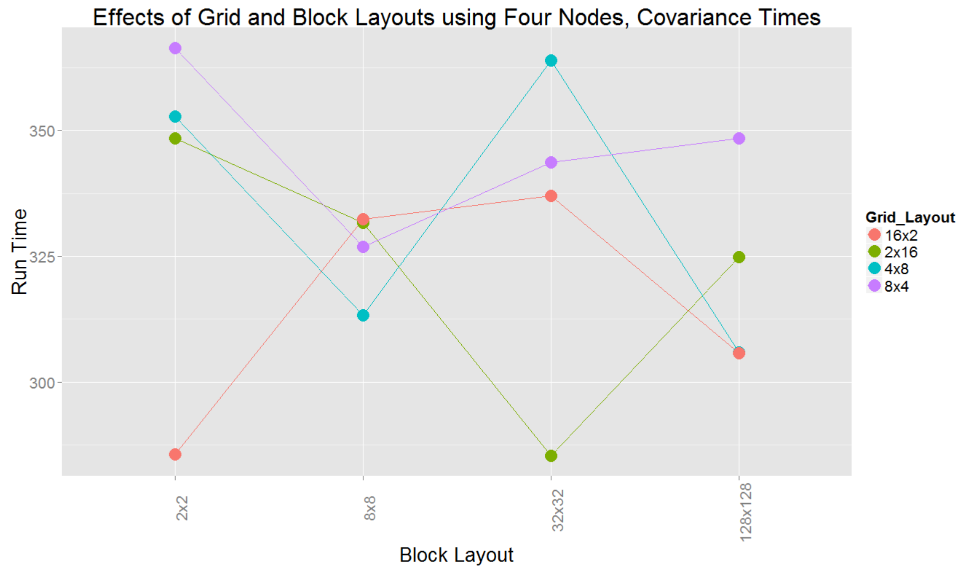

Figure 4.7: Covariance times for 4 node study.

example, in the one node study using 8 processes, block size contributes 95.388% of the variability, computed by dividing the sum of squares (SS) of the block size by the total sum of squares, whereas grid layout only contributes 1.942%. This leads us to believe that for computing PCA, block size serves as the main contributor to the variance.

However, the same conclusions cannot be drawn for the covariance matrix computation. We analyzed the covariance computation separate from PCA due to the relatively small amount of computation time needed for covariance vs. PCA, where n is only used in the

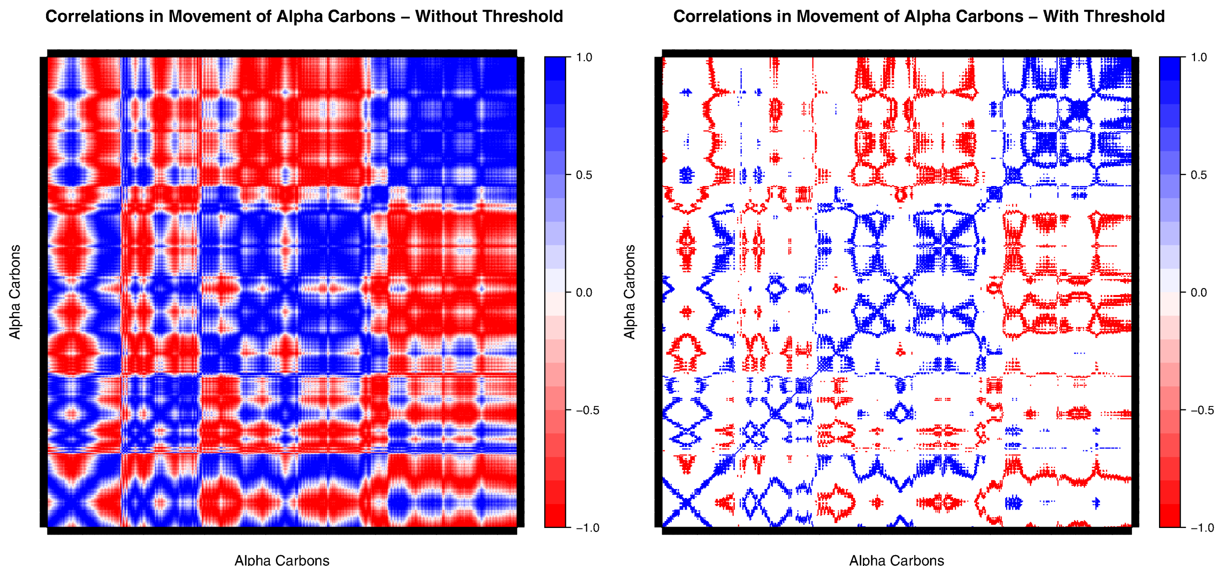

Figure 4.8: Level plot before and after greying out correlations below threshold.

covariance evaluation. The covariance matrix is a square $k \times k$, and therefore our large n values do not affect PCA hugely. When we isolated the run times for covariance matrix computation, we noticed very different results, which can also be noted in the ANOVA tables. For example, for the same one node study using 8 processes, now block size contributes 65.714% and grid layout contributes 5.714%. Though block size still seems to have a larger effect, these percentages are drastically different. We believe there to be some sort of interaction between grid layout and block size for computations that involve an n much larger than k, i.e., large rectangular matrices. Another potential for further research would be to investigate the nature of this interaction.

Acknowledgments

These results were obtained as part of the REU Site: Interdisciplinary Program in High Performance Computing (`www.umbc.edu/hpcreu`) in the Department of Mathematics and Statistics at the University of Maryland, Baltimore County (UMBC) in Summer 2013. This program is funded jointly by the National Science Foundation and the National Security Agency (NSF grant no. DMS–1156976), with additional support from UMBC, the Department of Mathematics and Statistics, the Center for Interdisciplinary Research and Consulting (CIRC), and the UMBC High Performance Computing Facility (HPCF). HPCF (`www.umbc.edu/hpcf`) is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from UMBC. Co-author Jordan Ramsey was supported, in part, by the UMBC National Security Agency (NSA) Scholars

Program though a contract with the NSA. Graduate RA Andrew Raim was supported by UMBC as HPCF RA.

References

[1] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, 1997.

[2] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[3] Andrew M. Raim. Introduction to distributed computing with pbdR at the UMBC High Performance Computing Facility. Technical Report HPCF–2013–2, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2013.

[4] Alvin C. Rencher. *Methods of Multivariate Analysis*. John Wiley & Sons, Inc., 2002.

[5] Drew Schmidt, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel. Guide to the pbdDMAT package. Version 2.0, `http://cran.r-project.org/web/packages/pbdDMAT/vignettes/pbdDMAT-guide.pdf`, 2012.

Pilot study to analyze effects of n and k on computational efficiency							
n	k	Block	Grid	Overall time	Cov. time	PCA time	log(Overall time)
1000	100	2×2	16×4	0.857	0.337	0.520	-0.067
1000	100	2×2	8×8	0.709	0.259	0.450	-0.149
1000	100	2×2	4×16	0.764	0.300	0.464	-0.117
1000	100	4×4	16×4	0.930	0.321	0.609	-0.031
1000	100	4×4	8×8	0.668	0.231	0.437	-0.175
1000	100	4×4	4×16	0.606	0.317	0.289	-0.213
1000	100	8×8	16×4	1.067	0.299	0.768	0.028
1000	100	8×8	8×8	0.678	0.347	0.331	-0.169
1000	100	8×8	4×16	0.658	0.248	0.410	-0.182
3000	300	2×2	16×4	1.003	0.306	0.697	-0.001
3000	300	2×2	8×8	0.759	0.268	0.491	-0.120
3000	300	2×2	4×16	0.845	0.323	0.522	-0.073
3000	300	4×4	16×4	1.037	0.300	0.737	-0.016
3000	300	4×4	8×8	0.741	0.224	0.517	-0.130
3000	300	4×4	4×16	0.606	0.307	0.299	-0.213
3000	300	8×8	16×4	1.234	0.339	0.895	0.091
3000	300	8×8	8×8	0.746	0.274	0.472	-0.127
3000	300	8×8	4×16	0.884	0.374	0.510	-0.053
5000	500	2×2	16×4	1.191	0.344	0.697	0.076
5000	500	2×2	8×8	0.967	0.286	0.491	-0.015
5000	500	2×2	4×16	0.923	0.360	0.522	-0.035
5000	500	4×4	16×4	1.187	0.388	0.737	0.074
5000	500	4×4	8×8	0.950	0.338	0.517	-0.022
5000	500	4×4	4×16	1.017	0.498	0.299	0.007
5000	500	8×8	16×4	1.337	0.475	0.895	0.126
5000	500	8×8	8×8	0.907	0.311	0.472	-0.042
5000	500	8×8	4×16	1.091	0.346	0.510	0.038
10000	1000	2×2	16×4	2.759	0.771	0.697	0.441
10000	1000	2×2	8×8	2.264	0.696	0.491	0.355
10000	1000	2×2	4×16	2.533	0.776	0.522	0.404
10000	1000	4×4	16×4	2.611	0.771	0.737	0.417
10000	1000	4×4	8×8	2.228	0.650	0.517	0.348
10000	1000	4×4	4×16	2.591	0.701	0.299	0.413
10000	1000	8×8	16×4	2.691	0.899	0.895	0.430
10000	1000	8×8	8×8	2.389	0.744	0.472	0.378
10000	1000	8×8	4×16	2.480	0.724	0.510	0.394
30000	3000	2×2	16×4	32.341	5.675	0.697	1.508
30000	3000	2×2	8×8	37.628	5.189	0.491	1.576
30000	3000	2×2	4×16	36.829	6.339	0.522	1.566
30000	3000	4×4	16×4	30.977	5.436	0.737	1.491
30000	3000	4×4	8×8	32.947	5.113	0.517	1.518
30000	3000	4×4	4×16	33.820	5.482	0.299	1.529
30000	3000	8×8	16×4	29.354	5.257	0.895	1.468
30000	3000	8×8	8×8	31.979	4.935	0.472	1.505
30000	3000	8×8	4×16	34.405	5.535	0.510	1.537
50000	5000	2×2	16×4	134.771	18.340	116.431	2.130
50000	5000	2×2	8×8	146.750	18.392	128.358	2.167
50000	5000	2×2	4×16	181.826	28.085	153.741	2.260
50000	5000	4×4	16×4	130.050	17.534	112.516	2.114
50000	5000	4×4	8×8	134.036	19.481	114.555	2.127
50000	5000	4×4	4×16	173.336	26.695	146.641	2.239
50000	5000	8×8	16×4	122.932	17.326	105.606	2.090
50000	5000	8×8	8×8	136.511	17.826	118.685	2.135
50000	5000	8×8	4×16	161.763	26.812	134.951	2.209

Table 5.1: TBD: Needs a caption

One node study to analyze effects of grid layout and block size on computational efficiency					
Block	Grid	Overall time	Cov. time	PCA time	log(PCA time)
2×2	1×2	3842.167	443.235	3398.932	3.531
2×2	2×1	3877.976	433.529	3444.447	3.537
2×2	1×4	2356.561	272.741	2038.820	3.319
2×2	2×2	2635.694	270.108	2365.586	3.374
2×2	4×1	2279.065	274.894	2004.171	3.302
2×2	1×8	1573.560	195.102	1378.458	3.139
2×2	2×4	1617.721	193.146	1424.575	3.154
2×2	4×2	1543.913	189.201	1354.712	3.132
2×2	8×1	1568.729	193.828	1374.901	3.138
8×8	1×2	3733.481	458.797	3274.684	3.515
8×8	2×1	3955.449	470.131	3485.318	3.542
8×8	1×4	2109.792	255.371	1854.421	3.268
8×8	2×2	2138.616	273.644	1864.972	3.271
8×8	4×1	2215.427	256.748	1958.679	3.292
8×8	1×8	1488.145	194.049	1294.000	3.112
8×8	2×4	1498.672	192.685	1305.987	3.116
8×8	4×2	1504.172	197.640	1306.532	3.116
8×8	8×1	1508.197	197.125	1311.072	3.118
32×32	1×2	3792.556	432.900	3359.656	3.526
32×32	2×1	3788.126	441.100	3347.026	3.525
32×32	1×4	2356.403	289.640	2066.763	3.315
32×32	2×2	2907.591	281.409	2626.182	3.419
32×32	4×1	2302.033	263.508	2038.525	3.309
32×32	1×8	1625.494	194.599	1430.895	3.156
32×32	2×4	1623.021	193.474	1429.547	3.155
32×32	4×2	1610.474	193.139	1417.335	3.151
32×32	8×1	1639.114	193.786	1445.328	3.160
128×128	1×2	4044.472	482.679	3561.793	3.552
128×128	2×1	4569.753	438.712	4131.041	3.616
128×128	1×4	2380.398	265.830	2114.568	3.325
128×128	2×2	2405.942	266.965	2138.977	3.330
128×128	4×1	2269.469	273.623	1995.846	3.300
128×128	1×8	1680.099	200.834	1479.265	3.170
128×128	2×4	1711.300	197.873	1513.427	3.180
128×128	4×2	1689.560	200.055	1489.505	3.173
128×128	8×1	1690.248	197.403	1492.845	3.174

Table 5.2: TBD: Needs a caption

Four node study to analyze effects of grid layout and block size on computational efficiency					
Block	Grid	Overall time	Cov. time	PCA time	log(PCA time)
2×2	2×16	2056.173	348.499	1707.674	3.232
2×2	4×8	2112.691	352.826	1759.865	3.245
2×2	8×4	2119.155	366.385	1752.770	3.244
2×2	16×2	1942.410	285.605	1656.805	3.219
8×8	2×16	1789.567	331.640	1457.927	3.164
8×8	4×8	1830.972	313.225	1517.747	3.181
8×8	8×4	1823.879	326.904	1496.975	3.175
8×8	16×2	1800.444	332.380	1468.064	3.167
32×32	2×16	2045.125	285.324	1759.801	3.245
32×32	4×8	2054.518	364.027	1690.491	3.228
32×32	8×4	2046.963	343.612	1703.351	3.231
32×32	16×2	2134.329	337.023	1797.306	3.255
128×128	2×16	2118.554	324.817	1793.737	3.254
128×128	4×8	2139.09	305.858	1833.232	3.263
128×128	8×4	2195.315	348.422	1846.893	3.266
128×128	16×2	2236.488	305.781	1930.707	3.286

Table 5.3: TBD: Needs a caption

PCA time for One node 2 processes				
Source	DF	SS	MS	F
Grid	1	0.0060	0.0060	2.6087
Block	3	0.0239	0.0080	3.4783
Residuals	3	0.0070	0.0023	
Total	7	0.0369	0.0163	6.0870

PCA time for One node 4 processes				
Source	DF	SS	MS	F
Grid	2	0.0286	0.0143	2.6981
Block	3	0.0440	0.0147	2.7736
Residuals	6	0.0316	0.0053	
Total	11	0.1042	0.0343	5.4717

Table 5.4: TBD: Needs a caption

PCA time for One node 8 processes				
Source	DF	SS	MS	F
Grid	3	0.0008	0.0003	3.0000
Block	3	0.0393	0.0131	131.0000
Residuals	9	0.0011	0.0001	
Total	15	0.0412	0.0135	134.000

PCA time for Four nodes				
Source	DF	SS	MS	F
Grid	3	0.0007	0.0002	0.2593
Block	3	0.1040	0.0347	38.5556
Residuals	9	0.0316	0.0053	
Total	15	0.1363	0.0402	38.8149

Table 5.5: TBD: Needs a caption

Covariance time for One node 2 processes				
Source	DF	SS	MS	F
Grid	1	0.0007	0.0007	0.4667
Block	3	0.0061	0.0020	1.3333
Residuals	3	0.0046	0.0015	
Total	7	0.0114	0.0042	1.8000

Covariance time for One node 4 processes				
Source	DF	SS	MS	F
Grid	2	0.0010	0.0005	0.4167
Block	3	0.0057	0.0002	0.1667
Residuals	6	0.0073	0.0012	
Total	11	0.0140	0.0019	0.5834

Table 5.6: TBD: Needs a caption

Covariance time for One node 8 processes				
Source	DF	SS	MS	F
Grid	3	0.0002	0.0001	1.0000
Block	3	0.0023	0.0008	8.0000
Residuals	9	0.0010	0.0001	
Total	15	0.0035	0.0010	9.0000

Covariance time for Four nodes				
Source	DF	SS	MS	F
Grid	3	2223.8890	741.2963	1.0290
Block	3	669.9260	223.3087	0.3100
Residuals	9	6483.8900	720.4322	
Total	15	9377.705	1685.0372	1.339

Table 5.7: TBD: Needs a caption