Parallel Computation of High Dimensional Robust Correlation and Covariance Matrices

James Chilson, Raymond Ng, Alan Wagner and Ruben Zamar†
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada, V6T 1Z4
{chilson,rng,wagner}@cs.ubc.ca

†Department of Statistics
University of British Columbia
ruben@stat.ubc.ca

Abstract

The computation of covariance and correlation matrices are critical to many data mining applications and processes. Unfortunately the classical covariance and correlation matrices are very sensitive to outliers. Robust methods, such as QC and the Maronna method, have been proposed. However, existing algorithms for QC only give acceptable performance when the dimensionality of the matrix is in the hundreds; and the Maronna method is rarely used in practise because of its high computational cost.

In this paper, we develop parallel algorithms for both QC and the Maronna method. We evaluate these parallel algorithms using a real data set of the gene expression of over 6,000 genes, giving rise to a matrix of over 18 million entries. In our experimental evaluation, we explore scalability in dimensionality and in the number of processors, and the trade-offs between accuracy and computational efficiency. We also compare the parallel behaviours of the two methods. From a statistical standpoint, the Maronna method is more robust than QC. From a computational standpoint, while QC requires less computation, interestingly the Maronna method is much more parallelizable than QC. After a thorough experimentation, we conclude that for many data mining applications, both QC and Maronna are viable options. Less robust, but faster, QC is the recommended choice for small parallel platforms. On the other hand, the Maronna method is the recommended choice when a high degree of robustness is required, or when the parallel platform features a large number of processors (e.g., 32).

1 Introduction

Given n samples of v variables, the correlation between two variables measures the strength of the linear relationship between the two variables. Given two columns of samples of length n, the *covariance* between X_i and X_j is:

$$\mathrm{cov}(X_i, X_j) = Ave[(X_i - \mu_i)(X_j - \mu_j)]$$

where $\mu_i = Ave(x_i)$ and $\mu_j = Ave(x_j)$ are the means. A *covariance matrix* measures the relation between all pairs of variables. The *correlation* of two variables is the normalized value of the covariance of the two variables and is related to the covariance as follows:

$$\text{corr}(X_i, X_j) = \frac{\text{cov}(X_i, X_j)}{\sigma_i \sigma_j}$$

where σ_i and σ_j are the standard deviations of X_i and X_j.

The computation of covariance and correlation matrices are critical to many data mining operations and processes. For example, in exploratory data analysis, it is typical to determine which variables are highly correlated. Moreover, covariance and correlation matrices are used as the basis for principal components analysis, for manual or automatic dimensionality reduction, and for variable selection. They are also the basis for detecting multidimensional outliers through computation of Mahalanobis distances.

Unfortunately the classical covariance and correlation matrices are very sensitive to the presence of multidimensional outliers. Even a small fraction of outliers can distort the covariance or correlation values to the extent of rendering them misleading and virtually useless. This is a serious issue for most data mining applications, as the data cannot typically be assumed to be clean (outliers free).

The example shown in Figure 1 illustrates the problem. Figure 1 shows all pairwise scatter plots of the 5-dimensional data set called “Woodmod”. In Figure 1, consider the two rectangles at V1-V2 and V4-V5. In both rectangles there is a small cluster of outliers in the bottom right hand corners. Similar observations can be made in other rectangles as well. Note that although these points are clearly outliers in two dimensional space they are not univariate outliers, i.e., they do not show up as well detached outliers in any of the possible one-dimensional projections of the data. The presence of these outliers causes differences between the classical and robust correlations, sometimes very substantial differences, even including changes of sign. For example, in the previous two rectangles the classical correlations between V1 and V2 and between V4 and V5 are -.14 and -.24, respectively. whereas the robust correlations are significantly different, +.85 and +.65. If you were to delete the small cluster of outliers from the data set the variables, as given by the robust correlation values, are strongly correlated. This example illustrates the inadequacy of classical correlation and covariance matrices in the presence of outliers and the valuable role of robust alternatives.

To improve on the robustness of covariance and correlation, many methods have been proposed to deal with “dirty” large databases. While Section 1.1 will give a more detailed discussion on related work, two state-of-the-art methods are Quadrant Correlation (QC) [2] and the Maronna method [11].

Even though there is value in robust methods, the problem for QC and the Maronna method is that they are computationally expensive, particularly when the size of the matrix (i.e., $v \times v$) is large. Existing algorithms for QC only give acceptable performance when the value of v is in the hundreds. In practise, the Maronna method is rarely used because of its high computational cost. For many data mining applications, the value of v is in the thousands, if not higher. For example, in Section 2, we give a more detailed account of a gene expression data set with v, the number of genes measured in a microarray chip, exceeding

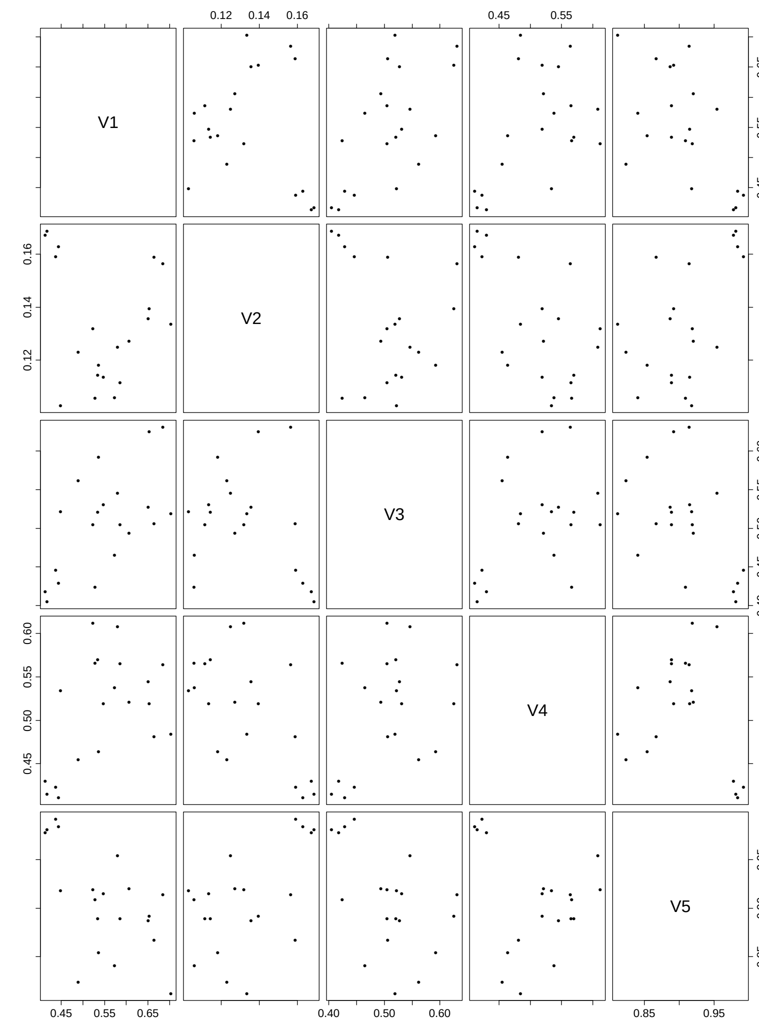

Figure 1: Robustness Results for a Small Five-dimensional Data Set.

6,000. Thus, the problem we tackle in this paper is: *How to compute high dimensional robust covariance and correlation matrices?*

The approach we explore in this paper is based on parallelization. This is motivated by the fact that multi-processor compute clusters have become inexpensive in the past decade, to the extent that even a small organization (e.g., a medical research laboratory) can find such a cluster affordable. For the algorithms we develop here, the target architecture is a compute cluster consisting of commodity processors running MPI/LAM, a public domain version of MPI. MPI (Message Passing Interface) is a standardized communication library for distributed memory machines and makes the programs easy to port to variety of parallel machines [5].

The parallelization of the Maronna and QC methods evolved during the course of the work as we experimented with the software on different clusters. The experimentation on real data sets motivated our investigation of load-balancing schemes, communication overheads on different clusters, and the questions about how best to trade-off computation for accuracy.

This paper makes the following contributions:

- First, we investigate the parallelization of QC on clusters. We find that the key computation is closely related to matrix multiplication. Dense Matrix multiply is a standard parallel computing kernel, which makes it easy to implement using off the shelf parallel numerical libraries such as ScaLAPACK and PLAPACK. As such, QC can be easily implemented on a variety of machines and, as is the case for matrix multiply, can be executed on large parallel machines.

- We also investigate the parallelization of the Maronna method. In the Maronna method the key step is for each pair to compute an iteration approximating the correlation. Each pair can be computed independently, which we can exploit in the parallel version. Although the Maronna method requires more computation than QC, the ability to decompose the computation into a large number of independent tasks make it amenable for parallel computation.

- We examine scalability in dimensionality and in the number of processors, and the trade-offs with accuracy and computational efficiency. For both methods, we provide a componentwise analysis of their execution times to provide insight into the different behaviours of the two methods in a parallel environment and why they behave so differently. We develop and evaluate a predictive performance model for the Maronna method to better understand the trade-offs with respect to scalability in processors and problem size.

- We conducted extensive empirical evaluation of the parallel algorithms with several real data sets. We report here the results based on the gene expression levels of 6,068 genes. We describe some interesting numerical properties of the Maronna method that arose from our ability to experiment on "real" and large data sets.

- We conclude with a "recommended recipe" covering various situations. Less robust but faster, QC is the recommended choice for small parallel platforms. On the other hand, the Maronna method is the recommended choice when a high degree of robustness is required, or when the parallel platform features a high number of processors. In any event, even for matrices of size over 6,000 $\times$ 6,000, covariance and correlation matrices can be computed in only tens of seconds of wall-clock time.

1.1 Related Work

The statistical literature contains a substantial number of papers studying the properties of robust covariance matrix estimation. The robustness of an estimate can be measured by its breakdown point - the maximum fraction of contamination the estimate can tolerate. There has been considerable emphasis on obtaining positive definite, affine equivariant estimators with a high breakdown point, namely a breakdown point of one-half. The best known such estimators are the Minimum Volume Ellipsoid (MVE) and Minimum Covariance Determinant (MCD) estimates [17, 18]. Another important class of affine equivariant high-breakdown point estimates are those based on projections: the Stahel-Donoho estimate (SDE) proposed by [19] and [4], and studied by [13, 9]; and P-estimates [12]. However, all known affine equivariant high-breakdown point estimates are solutions to a highly non-convex optimization problem and as such do not scale up to the large data sets which are commonplace in data mining applications.

The"Fast MCD" (FMCD) method is recently proposed that is much more effective than naive subsampling for minimizing the objective function of the MCD [15]. But FMCD still requires substantial running times for large v, and it no longer retains a high breakdown point with high probability when n is large.

The main challenge is to find good initial estimates from which one searches for a nearest optimum in hopes that it produces a global optimum. The initial estimates are invariably obtained by using some form of repeated random sub-sampling of N_s rows of the original data table, with the number of samples N_s determined in order to achieve a high-breakdown with high probability, e.g., with probability .99 or .999 (see for example [16]). It turns out that achieving this latter condition results in algorithms that have exponential complexity of order 2^p in terms of the dimension p of the data. This rules out the use of such estimates for many data mining applications where one has in excess of 20 - 30 variables. In addition, robust covariance matrix estimates based on projections have the problem of having quadratic computational complexity n^2 in the number of observations. Since many data mining applications involve hundreds of thousands if not millions of rows, the current projection estimators are not feasible for data mining.

Much faster estimates with high breakdown points can be computed if one is willing to drop the requirements of affine equivariance of the resulting covariance or correlation matrix. The simplest such methods are based on pairwise robust correlation or covariance estimates such as : (i) classical rank based methods, such as the Spearman's ρ and Kendall's τ (see for example [1]); (ii) classical correlations applied after coordinate-wise outlier insensitive transformations such as the quadrant correlation (QC) and 1-D "Huberized" data (see [7], p. 204); and (iii) bivariate outlier resistant methods such the method proposed by [6] and studied by [3]. The pairwise approach is appealing in that one can achieve high breakdown point on a pairwise basis that results in a high breakdown point for the overall covariance or correlation matrix, and at the same time reduces the computational complexity in the data dimension p from exponential to quadratic (from 2^p to p^2). This greatly increases the range of data mining problems to which robust covariance and correlation estimates can be applied, e.g., 200-300 variables becomes quite feasible.

In recognition of this opportunity, [10] and [2] recently proposed new pairwise methods based on a modification of approaches (iii) and (ii) respectively, that preserve positive definiteness and have computational complexity $\mathcal{O}(np^2)$. However, these pairwise methods are not affine equivariant and may be upset by the so called two-dimensional structural outliers. The example shown in Figure 2 illustrates the problem. If the data were perfectly clean, the classical Pearson correlation coefficient would be 0.95. However, a small percentage of

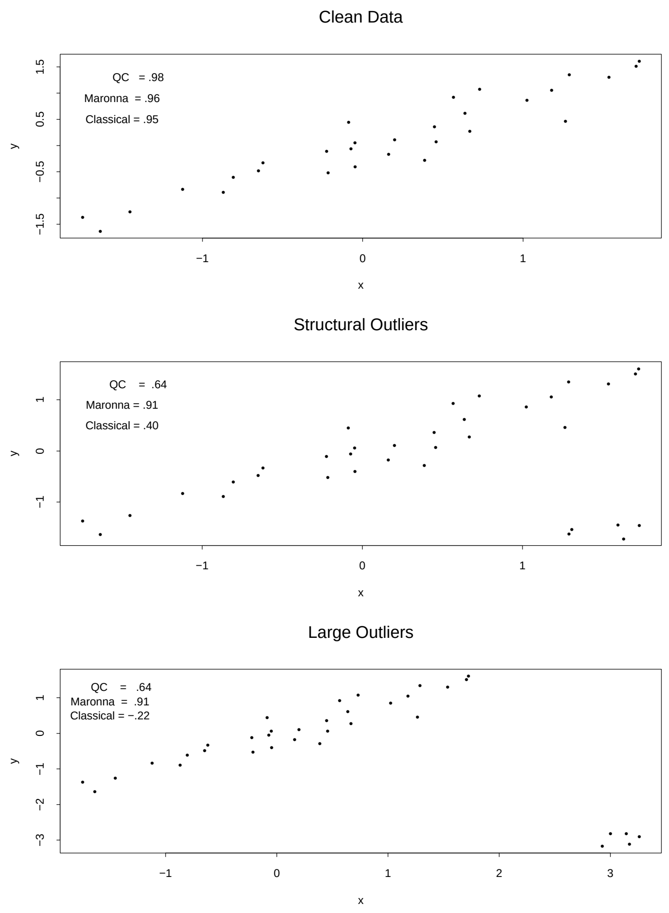

Figure 2: Advantage of Maronna over QC and the classical Pearson Correlation.

outliers (in this case, around 10%) was sufficient to create disaster for the classical coefficient, as it drops to -0.22. The effect of structural outliers on the Pearson correlation causes it to give a 0.40 correlation. The correlation based on QC drops from 0.98 when the data were perfectly clean, to 0.64 with a small percentage of outliers. Also, QC was unable to detect the two-dimensional structural outliers, and calculates a correlation of 0.64 in their presence as well. The Maronna method is even more robust, as the value only changes slightly from 0.96 to 0.91, even for the data with the structural outliers.

This can be explained because the Maronna method is positive definite and affine equivariant, and is thus more robust than QC. The problem, of course, is that the extra robustness requires a lot more computational effort. Thus, a key question to be addressed in this paper is how to develop a load-balanced parallel algorithm for the Maronna method. It will then be interesting to see if the parallel Maronna algorithm can eventually catch up with QC.

2 An Application: Gene Expression Analysis

In this section, we show a real-life application that requires the computation of a high-dimensional robust covariance matrix. This application arises from our strong ties with the cardiovascular research laboratory at the St Paul's Hospital in Vancouver (www.icapture.ubc.ca). Rheumatic valves in the heart cause heart failures, and represent one of the most common reasons for heart transplants. To try to understand how rheumatic valves are formed, researchers at the hospital collect gene expression data (i.e., using microarray technologies) for a number of rheumatic valves and normal valves. Specifically, for each sample/valve, each gene is associated with a non-negative count representing the number of times the gene has expressed itself in the valve. Based on these counts, we compute the covariance and correlation matrices.

These matrices are useful for a variety of reasons. One usage is that for any given gene G, we can find a ranked list of genes which are the most positively or negatively correlated with G. Another usage is depicted in Figure 3, where the correlation matrix is used to form a dissimilarity function for clustering a given collection of genes. Figure 3 gives a dendrogram for the given collection, showing clusters of correlated genes. The height is defined as $1 - r$, where r is the correlation coefficient between a pair of genes. Correlated genes help medical researchers to identify the biological pathways that are heavily involved in producing rheumatic valves.

There are, however, a number of problems in computing the covariance matrix. First, even though microarray technologies have improved dramatically in recent years, gene expression data are noisy (i.e., contain many outliers). Thus, robust methods for computing the covariance matrix are valuable. Second, as usually the case for many biomedical applications, n, the number of samples, may not be large. In our case each sample corresponds to a heart valve, and it takes a long time to collect even 10 rheumatic values from heart transplant patients. Minimizing the negative impact of noise is all the more important because of the small number of samples. Last but not least, the dimensionality of the matrix is very high.

The main data set used in our experiments was the rheumatic heart valve data consisting of 20 samples where each sample contained the gene expression counts for 6068 genes. There were 10 control samples and 10 rheumatic heart samples. Our experiments found the presense of outliers in the data and the need to minimize the impact of this noise on the final results. We also experimented with variations of our data set where we added random samples or new samples based on weighted combinations of other samples.

The 6068 gene data set corresponds to a 6068 $\times$ 6068 correlation or covariance matrix, with over 18 million entries. This magnitude far exceeds the capability of state-of-the-art algorithms for computing robust covariance matrix. In fact, we recently received a new version of the data set with over 12,000 expressed genes. Thus, there is an urgent need to develop algorithms for computing high dimensional robust covariance and correlation matrices.

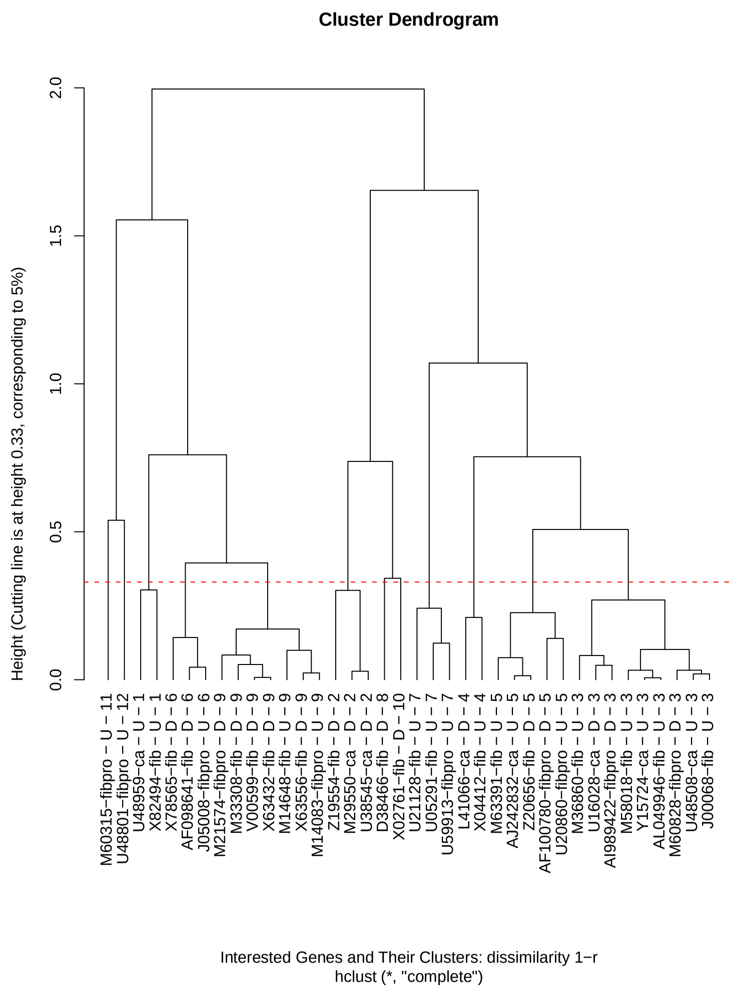

Figure 3: Dendrogram of the Most Correlated Genes

3 Parallel Correlation and Covariance Methods

The Maronna and Quadrant Correlation (QC) methods take as input a $n \times v$ matrix X with v variables and n cases, and compute as output a $v \times v$ matrix, which is either the covariance or correlation matrix. In general, both algorithms perform the following steps:

1. Calculate the median/MAD for each variable in X, where MAD stands for the Median Absolute Deviation the data from its median (cf: Section 3.3).

2. Compute the pairwise covariance/correlation for X.

3. Restore the positive definiteness of the matrix, if required.

Our main focus for this paper is step two, the covariance/correlations computation. The Maronna method is discussed in Section 3.1, while QC is discussed in Section 3.2. Step one is covered briefly in Section 3.3. Step three, restoring the positive definiteness of the covariance and correlation matrices, may or may not be necessary depending on the applications. The general restoration process is to determine the negative eigenvalues of the matrix and calculate replacement eigenvalues to recreate a positive definite matrix. In some cases, when the matrix has only a few nonzero eigenvalues, it can be quicker to solve for only the positive eigenvalues, then use these to rebuild the positive definite matrix. In any event, whether this optional step is required does not depend on whether QC or the Maronna method is chosen. Since this part of the calculation is optional, we omit any details here on step three.

3.1 The Maronna Method

A description of the parallel version of Maronna is shown in Figure 4. The algorithm begins with the calculation of median and MAD for each variable, which is discussed in more detail in Section 3.3. Each of the $\mathcal{O}(v^2)$ pairwise calculations can be computed independently. Each independent computation for a given v_i and v_j is iterative and converges at different rates. The inner sequential part of each computation, beginning on step 6 in Figure 4, does the following.

First, the values involved in the iteration are initialized in step 7 and 8. μ is a vector of length two, and is initialized with the median of the data variables involved in this correlation calculation. σ is a 2 x 2 matrix that will hold the estimate values for the correlation upon convergence. It is initialized as a diagonal matrix holding the MAD of the correlation variables in the diagonal. After initialization, the algorithm repeats the following process. The Mahalanobis distance is used to measure the distance between the samples of the pair of variables in step 14. The Mahalanobis distance measures the distance between a data point and the centroid of all the data points. A weight function is then applied to the distance values in step 16 to decrease the influence of outliers in the data. Our weight function uses Huber's score function as the robust M-estimate to score the influence of the sample points to the median and variance.

$$\begin{aligned}
\text{HSF}(y) &= \begin{cases} y & |y| \le c \\ c \cdot sign(y) & |y| > c \end{cases} \\
\text{is used to define} & \\
\text{weight}(y) &= \begin{cases} HSF(y)/y & y \neq 0 \\ 1 & y = 0 \end{cases} + \\
&= \begin{cases} 1 & |y| \le c \\ c/|y| & |y| > c \end{cases} \qquad (1)
\end{aligned}$$

The weight function gives weights between zero and one that are applied to the data. The weight function will weigh normal data variables near one and down-weigh the outlier values with weights closer to zero.

The weighted data is used to calculate new values for μ and σ for the next iteration in steps 18 and 19. The loop continues until the change in covariance from one step to another

```
Input:   $n$ by $v$ matrix $X$ with $v$ variables and $n$ cases
Output:  $v$ by $v$ matrix cor, the correlation matrix
1.  In parallel For each column $X_i$
2.      Calculate the median and MAD.
3.  Let median[$i$] be the median of column $i$.
4.  Let MAD[$i$] be the MAD calculation of column $i$.
5.  In parallel For each pair of variables $i$, $j$
6.      Initially
7.          $\mu^{(0)} = \left[\text{median}[i],\ \text{median}[j]\right]$
8.          $\sigma^{(0)} = \begin{bmatrix} (\text{MAD}[i])^2 & 0 \\ 0 & (\text{MAD}[j])^2 \end{bmatrix}$
9.      Let $\mathbf{x}_q$ be the vector $[X_i[q],\ X_j[q]]$.
10.     ITERATE
11.         Given $\mu^{(k)}$ and $\sigma^{(k)}$
12.         For $q = 1$ to $n$
13.             // Calculate the Mahalanobis distance.
14.             $\text{mah}[q] = [\mathbf{x}_q - \mu^{(k)}] \cdot [\sigma^{(k)}]^{-1} \cdot [\mathbf{x}_q - \mu^{(k)}]^T$
15.             // Down weight the outliers.
16.             $W[q] = \text{weight}(\text{mah}[q])$
17.         Calculate
18.             $\mu^{(k+1)} = \dfrac{\sum_{q=1}^{n} W[q] \cdot \mathbf{x}_q}{\sum_{q=1}^{n} W[q]}$
19.             $\sigma^{(k+1)} = \frac{1}{n} \cdot \sum_{q=1}^{n} (W[q])^2 \cdot \left[\mathbf{x}_q - \mu^{(k+1)}\right]^T \cdot \left[\mathbf{x}_q - \mu^{(k+1)}\right]$
20.     // Check for convergence.
21.     UNTIL (determinant $|((\sigma^{(k)})^{-1}(\sigma^{(k+1)}) - 1| < \epsilon$)
22.     Let $\sigma^*$ denote the converged value for columns $X_i$ and $X_j$.
23.     $\text{cor}[i][j] = \sigma^*[0][1] / \sqrt{(\sigma^*[0][0] \cdot \sigma^*[1][1])}$
```

Figure 4: Parallel Maronna Method

is within the desired tolerance. The algorithm is known to converge, but the rate can vary depending on the input.

The sequential version of Maronna uses a single processor to perform all of the pairwise computations. The independence of these $\mathcal{O}(v^2)$ pairwise computations makes Maronna an excellent candidate for parallelization. The parallel version divides the pairwise computations into p groups, one for each processor. The challenge in computing the covariances efficiently is to ensure that the work is distributed and equally shared among the processors. As we will show in Section 4 the number of iterations varies significantly and can potentially slow down the computation while some processors wait for others to finish. As well, care must be taken in the distribution and gathering of the results since, for large problem sizes, there are a large number of pairs to be distributed. The experiments in Section 4 show that Maronna can achieve significant speed-up on large problem sizes and can effectively use a large number of processors.

3.2 Quadrant Correlation

Figure 5 describes the parallel version of QC. Again, QC also begins with a calculation of the median and MAD values for the variables. Although there are several vector operations, the major computation in QC is a large matrix multiplication. In the algorithm, we represent the matrices in column-major order, where the columns are variables. Thus $X[i]$ refers to the *ith* column (variable). After calculating the median and MAD for all the variables, the algorithm creates a temporary matrix to hold the normalized values:

$$X[i][j] = \frac{X[i][j] - \text{median}(X[i])}{\text{MAD}(X[i])}$$

The X matrix is then used to create a matrix Y of all 1's, -1's, and near zero values by applying a function, ψ, that is similar to the sign function, to all the elements in X.

$$\psi(x, c) = \begin{cases} \text{sign}(x) & \text{if } |x| > c \\ \frac{x}{c} & \text{otherwise} \end{cases}$$

Our sign function cuts off the values within c of zero and assigns them to the value $\frac{x}{c}$. Our choice for c in the code was 0.00001. In actuality, by our ψ function, we are using a Huberized estimator, which in the limiting case is Quadrant Correlation [2]. The limiting case here would be to use the sign function in the place of ψ.

In the next step, the algorithm calculates the following equation to fill in each entry of the correlation matrix:

$$\text{cor}(i, j) = \frac{\frac{1}{n}\sum_k(\psi(X[j][k])) \cdot (\psi(X[i][k]))}{\sqrt{(\frac{1}{n}\sum_k(\psi(X[j][k]))^2 \cdot \frac{1}{n}\sum_k(\psi(X[i][k]))^2)}} \tag{2}$$

The computationally expensive part of the calculation is the part where the numerator is calculated using a matrix multiplication between Y and its transpose in steps 8 and 10 of Figure 5. The operations involved are approximately $\mathcal{O}(v^3)$. The denominator is the geometric mean of the average number of nonzero elements for a pair of columns i and j. This part of the calculation takes $\mathcal{O}(v^2)$ time and is set up in step 13. The equation finishes in step 16 where the denominator divides the numerator. Again, this division occurs for every element in the matrix. Thus, step 16 requires $\mathcal{O}(v^2)$ time.

```
Input:   $n$ by $v$ matrix $X$ for $v$ variables and $n$ cases
Output: $v$ by $v$ matrix cor, the correlation matrix
1.  In parallel For each column $X_i$
2.        Calculate the median and MAD
3.  Let median[$i$] be the median of column $i$.
4.  Let MAD[$i$] be the MAD calculation of column $i$.
5.  // Find "sign" ($\psi$) of normalized values.
6.  Construct $Y$ where $Y[i][j] = \psi(X[i][j] - \text{median}[i])/\text{MAD}[i], C)$;
7.  // Parallel matrix multiply.
8.  In parallel compute matrix $\text{cor} = Y \cdot Y^T$;
9.  // Scale the matrix in parallel.
10. In parallel $Y = \frac{1}{n} \cdot Y$; (element-wise)
11. For all $i$, $j$, set $Y[i][j] = (Y[i][j])^2$;
12. Construct vector $D$
13.       $D[i] = \frac{1}{\sqrt{\text{avg}(Y[i])}}$;
14. // Parallel operations on the distributed matrix object.
15. In parallel For all $i$, $j$ set
16.       $\text{cor}[i][j] = \text{cor} \times D[i] \times D[j]$
17.       $\text{cor}[i][j] = \sin(\frac{\pi}{2} \cdot \text{cor}[i][j])$;
```

Figure 5: Parallel Algorithm for Quadrant Correlation.

The parallelization of QC is complicated by the number of different types of vector operations that it performs. These operations and a matrix multiply need to be performed on matrices and vectors that are distributed across the p processors. Rather than create our own vector and matrix library we implemented QC using the PLAPACK library [20]. PLAPACK is a well-known parallel numerical library from the University of Texas at Austin that provides a variety of vector and matrix operations. The library is used to construct a processor mesh and partition the linear algebra objects, vectors and matrices, into blocks that are distributed to the processors. Once the objects are distributed the operations can be done in parallel with each processor working on their pieces of the distributed objects. There is some communication between the processors during this computation when the values residing on other processors are needed, so the processors do not work independently.

The difficulty in implementing QC using PLAPACK was to determine the block size and distribution patterns to avoid undue communication between the processors necessary to perform the various vector, matrix operations. It is possible to reduce the communication by replicating the matrix in each processor. However, this is impossible for larger problem sizes. Memory size was an issue on the problem sizes that we experimented with in this paper.

3.3 Median and MAD Calculation

Maronna and QC use the median and MAD values for each variable (i.e., column). MAD, which stands for the median absolute deviation, measures the deviation of the data from the median and is a more robust measure than the standard deviation. The MAD value of a variable can be directly calculated from the median using the formula below.

$$\mathrm{MAD}(X[i]) = \frac{\mathrm{median}(|X[i][j] - \mathrm{median}(X[i])|)}{0.6745}$$

The constant .6745 appearing in the formula is the inverse of the third quartile of the normal distribution. The numerator alone tends to underestimate the standard deviation and dividing by .6745 leads to a better estimate [14]. It is possible, with slight modifications to the median finding algorithm, to directly calculate the MAD values for use by Maronna and QC.

The choice of whether or not to do the MAD calculations in parallel depends on v and n. For large v, since each variable can be computed independently, we can distribute the work in a similar fashion as we did in the Maronna method. It does differ from the correlations calculations in that there are far fewer independent calculations. For small v and large n, it becomes necessary to do the median-finding computation in parallel. Parallel median finding algorithms are well-known [8]. We choose not to implement a parallel median finding algorithm partly because our focus is high dimensional data where the former case is more important, and partly because median-finding time is dominated by the correlation/covariance calculations.

4 Experimental Results

4.1 Experimental Setup

We evaluated the parallel Maronna and QC methods in two different cluster environments:

- a small platform: a collection of eight 500MHz Pentium-3 processors running Linux using MPI-LAM on a 100Mbps LAN; and
- a grid platform: WestGRID [21], a recently installed compute cluster consisting of 504 dual processor 3 GHz Xeon processors running Linux with 2 GB of RAM on Gigabit Ethernet (www.westgrid.ca).

The majority of our experiments were performed on the Pentium-3 system which provided a dedicated and controlled environment to evaluate and test different versions of the program. The WestGRID facility was used to evaluate the scalability of the two methods for large number of processors and increasing problem sizes.

We experimented with several real data sets. The results reported below are based on the gene expression levels of 6,068 genes on rheumatic and normal heart valves, as described in Section 2. We repeated each experiment ten times and report the best results because these more closely represent what performance would be in an ideal setting.

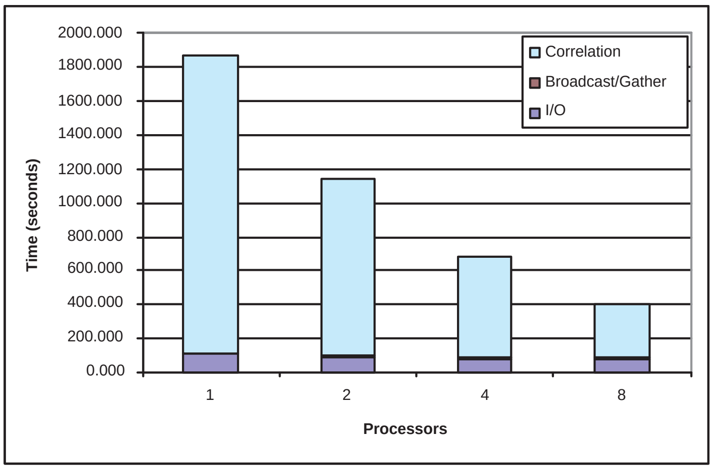

Figure 6: Maronna Performance for large problem size (6000 variables)

4.2 Parallel Maronna Method

The Maronna method was parallelized using a task farm organization where a manager process distributes the pairs to be correlated to worker processes. The input $n \times v$ matrix was distributed to all the workers so that each could independently work on different portions of the correlation matrix.

Figure 6 shows the total time (wall clock time) taken for the Maronna method using the small platform. As expected, the total time decreased as the number of processors increased. On 8 processors the wall clock time was about 400 seconds, representing a speed-up of 4.5 out of 8. The results for Figure 6 used a static load-balancing scheme. The relatively poor speed-up lead us to investigate the affect of load-balancing on the computation and to implement the dynamic load-balancing scheme discussed in Section 4.3.

The total execution time can decomposed into several different time components: (a) correlation calculation, (b) median/MAD calculation, (c) communication overhead, (d) I/O time, and (e) miscellaneous overheads. In Figure 6 each bar is divided into the major time components that make up the total time. The two most dominant components in Figure 6 are the correlation component and the I/O and communication component. The other components are so small they do not appear on the chart.

Given the different time components we attempted to model the performance of the Maronna method by measuring, where possible, the time for individual components and to use this data along with knowledge of the algorithms to find an expression for the total execution time. The model attempts to predict the execution time based on v, the number of variables, n, the number of cases, p, the number of processors.

$$\text{Time}(n, v, p) = \mathcal{O}(v^2 n/p) + \mathcal{O}(vn/p) + \mathcal{O}(v^2 p + v^2 + p) + \mathcal{O}(v^2) + \mathcal{O}(1) \tag{3}$$

The first two terms in $\text{Time}(n, v, p)$ are calculation times for correlation and the median/MAD. The next terms are the communication, I/O and miscellaneous overheads, respectively. In the following we describe each of the components in more detail and provide experimental evidence to validate the performance model given by Equation 3.

Figure 7 shows the main computational part of the method, calculating the correlation. In the method the correlations are all independent and once they have been distributed to the worker processors, the processors can compute until the results are returned.

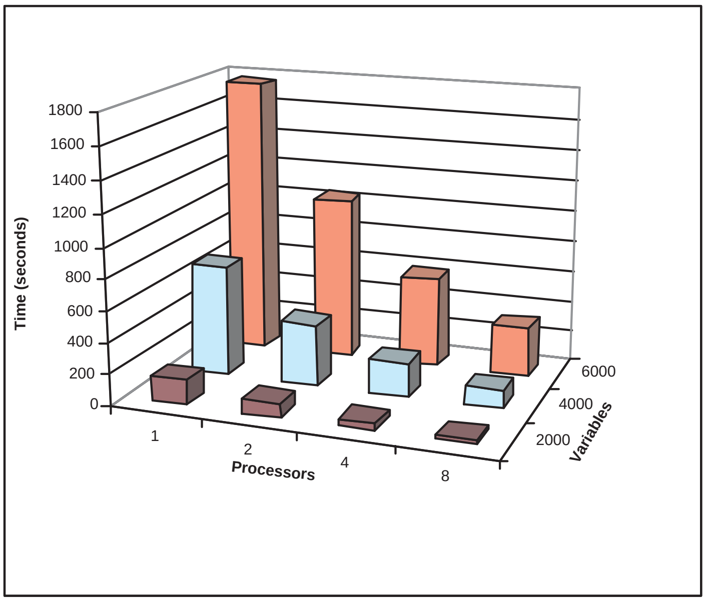

Figure 7: The Correlation Component

A linear least-squares approximation was calculated for the correlation component. We were concerned because our model did not accurately predict the experimental results. The $\mathcal{O}(\frac{v^2 n}{p})$ term in our model assumes that all the work for computing the correlation is equally distributed among the processors. In Table 1, we test this hypothesis by showing how close the actual speed-ups for Maronna's correlation component are to linear.

v	p	Linear Speed-up	Static Speed-up	Static Error	Dynamic Speed-up	Dynamic Error
2000	2	2.00	1.87	7.13%	1.99	0.31%
2000	4	4.00	3.66	9.38%	3.96	1.06%
2000	8	8.00	6.99	14.40%	7.95	0.62%
4000	2	2.00	1.81	10.58%	2.00	0.03%
4000	4	4.00	3.33	20.20%	3.91	2.24%
4000	8	8.00	6.31	26.80%	7.98	0.30%
6000	2	2.00	1.69	18.40%	2.05	2.45%
6000	4	4.00	2.99	33.72%	4.06	1.46%
6000	8	8.00	5.54	44.36%	8.21	2.53%

Table 1: Static Maronna Correlation Speed-up vs Linear Speed-up and Dynamic Speed-up

The static Maronna's correlation speed-up values are reasonable, but there is significant error between them and perfectly linear speed-up. This makes it difficult to create a linear predictive model for the correlation component. When we compare the error terms for static Maronna to the dynamic load balanced version we discuss in Section 4.3, we see there is a great difference. The error for the dynamic Maronna is small, so the speed-up is very near linear. This lends support to our prediction of a $\mathcal{O}(\frac{n^2}{p})$ term for the correlation component because the dynamic version of Maronna eliminates the unbalanced work load, and thus shows us a better view of the correlation components behaviour.

The speed-up for the Dynamic Maronna on 6000 variables in Table 1 seems to be super linear. There are two reasons for this. First, for our experiments, there is a small amount of variation in running time, and the running times for two, four, or eight could have varied on the fast side or the single processor runs could have been slow. Another reason for this performance is that the 6000 variable correlation matrix is large enough to put a strain on memory resources, especially if the computation is limited to a single processor as in the sequential case. This causes more page faults to happen, resulting in a slower sequential time, which inflates the speedup figures for two or more processors. We believe that the speedup figures are really linear, not super linear, and the small error terms for the dynamic version support our position.

As one might expect, a key factor that affected the speed-up was the communication time as shown in Figure 8. The communication component included the time needed to

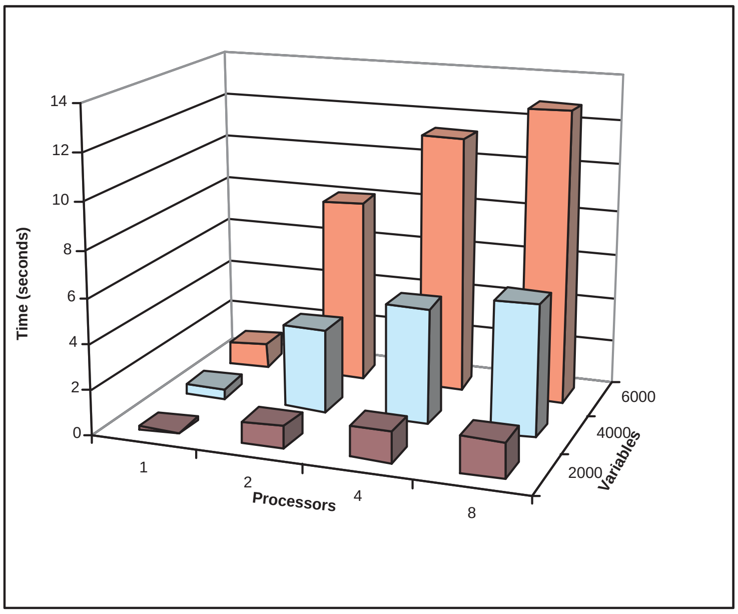

Figure 8: The Communication Component

distribute the pairwise correlations to the processors and the time to gather the results back to the manager processor. It increased with both problem size and number of processors, and resulted in slightly lower speed-up with regard to the total time.

We did not model the scatter portion of the communication. The scatter time was a couple of orders of magnitude smaller than the gather time. Because each worker processor has a copy of the initial data set the amount of data to distribute tasks to processors is extremely small. In reality, there are probably one or more n terms in the equation, but the applications we are interested in had n so small that it was easily lost in the constants of the modelling equation. The predicated communication time for the gather phase is given by the expression $\mathcal{O}(v^2p + v^2 + p)$, which depends on the number of processors and the size of the output. This expression for communication arose from fitting the experimental result to the data. A linear least-square approximation of the experimental data for each problem size was done to obtain the lines for $p = 2, 4, 8$. These expressions were then used to find linear expressions for the co-efficients with increasing number of p. The final expression we obtained was:

$$0.019v^2p + 0.201v^2 + 0.0045p - 0.0038$$

where v^2 is the problem size given as {1,4,9,16,25}.

As shown in Table 2 our predictive model for the gather time is within 20% of the experimental data. The value for the $p = 4$ is consistently too large. There are not enough data points to determine whether this indicated a problem with the model. Another interesting trend is that the error appears to be decreasing as the problem size and number of processors increases.

v	p	Experimental Time	Predicted Time	Percent Error
2000	2	0.926	0.9612	3.66%
	4	1.336	1.1222	19.04%
	8	1.517	1.4442	5.04%
4000	2	3.624	3.8292	5.35%
	4	5.106	4.4462	14.83%
	8	5.804	5.6802	2.17%
6000	2	8.115	8.6092	5.74%
	4	11.350	9.9862	13.66%
	8	12.875	12.7402	1.06%

Table 2: Maronna Communication Time versus Predicted Values

The most significant term in the above expression is the $0.201v^2$, which relates to bandwidth. There is a v^2p term that was necessary to accurately model the data. Although it is relatively small in comparison to the v^2 term, it begins to dominate the equation as p increases. We were surprised that the gather time did not behave linearly in v^2 and p (i.e., $\mathcal{O}(v^2 + p)$). One possible explanation is network congestion. We did investigate the gather time more closely by monitoring the network traffic (using SNMP from a router). This confirmed the fact that the large burst of communication at the end of the Maronna method was the likely culprit. Later implementations of the software attempted to alleviate this problem by returning intermediate results back to the manager processor thus avoiding overloading the network.

Apart from the correlation and the communication components, the remaining components included the median/MAD calculation, matrix fill time, I/O time and miscellaneous other operations to initialize and manage memory. Section 4.7 will discuss the median/MAD time.

The matrix fill time was the time required to copy the results from the message buffers into the final result matrix. We could have eliminated much of this time by gathering the result directly into the matrix. In general, the time was small and constant. It did increase the time substantially when memory constraints resulted in page faults. This explains the relatively large time on one and two processors, 41 seconds and 22 seconds respectively. These page faults did slightly inflate the apparent speed-up in Figure 6. The I/O time remained relatively constant for a fixed program size. The time to write the $v \times v$ matrix to disk was the major portion of the I/O time.

We combine our analysis of the I/O and matrix fill portion of the model because they both rely on the output matrix size, v^2. A least-squares fit of the experimental data resulted in a linear equation with v^2 terms that had constants of .1842 for the matrix fill part and 1.75 as the constant for the I/O time. As expected, disk I/O is an order of magnitude larger than memory copy time. The predicted times using the above equation was always within 6% of the actual measured time and the accuracy improved with increasing problem size.

4.3 Load Balancing Maronna

Initially, we statically divided the set of pairs into p tasks, distributed one task to each of the p processors, and gathered the results at the end. If the number of iterations for each pair was constant or randomly distributed the processors all finished at the same time. However, this was not the case for the gene data set. While 99.9% of pairs converged within 200 iterations, the remaining pairs took up to 2400 iterations. The variation in convergence rates resulted in load imbalance when tasks were statically allocated. As a result we implemented a dynamic scheme.

Our dynamic load-balancing implementation used a demand-driven task allocation scheme. We divided the work into tasks where each task was a block of pairs to be correlated. We experimentally determined the optimal task size to ensure that there were enough tasks to equally distribute the load and the tasks were large enough to amortize the overhead of distributing and gathering the tasks and results. When a processor finished a task it requested a new task from the manager. A double buffering scheme was used to overlap the execution of a task with the request for a new one.

We actually have two different versions of the load balanced Maronna. The two versions differ in how they gather the results into the correlation matrix. One version follows a similar approach to static Maronna and has the processors store their results until they all finish calculating, then they all perform a giant gather operation to combine all the results. This version still has the benefit of balancing the calculations between the processors as can be seen where its values were used in Table 1. However, the giant collective gather at the

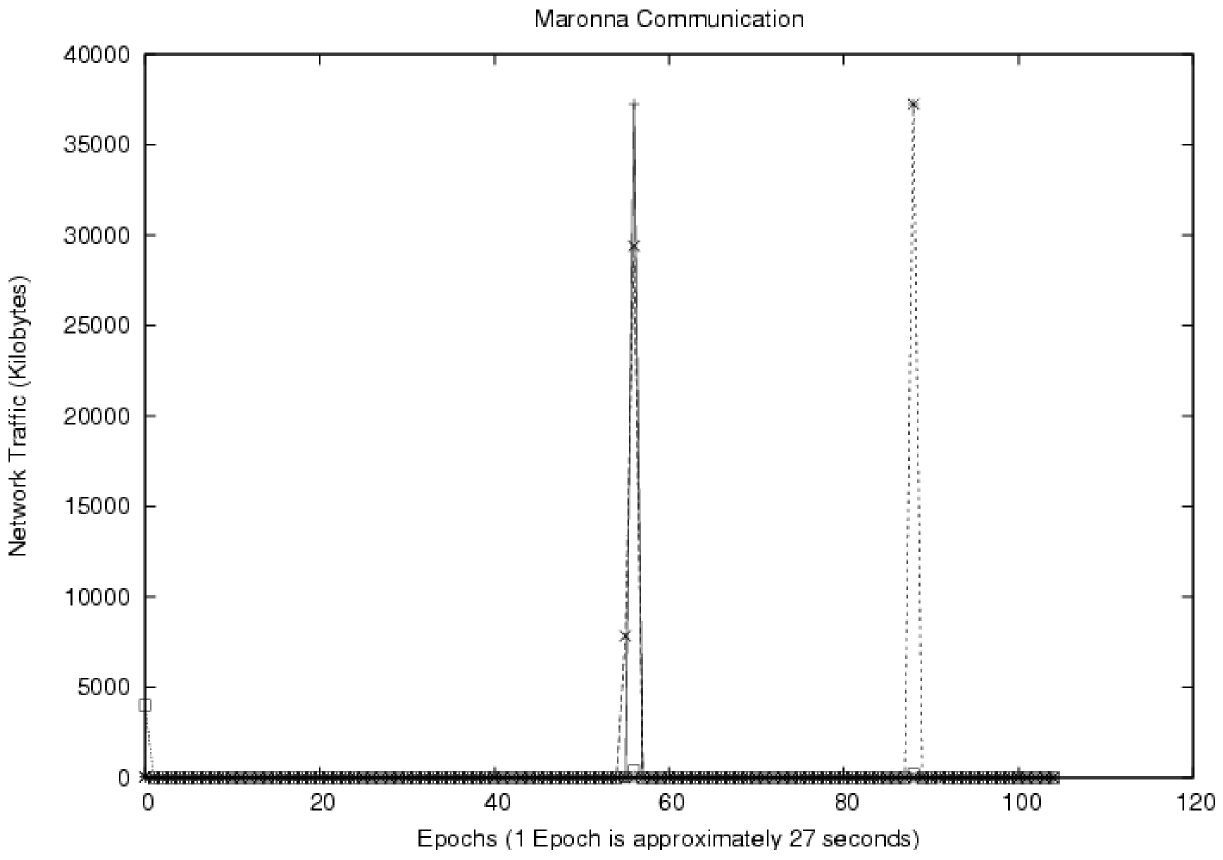

Figure 9: Maronna Communications Profile

end could saturate the root processor and cause the gathering time to be long. Our second version deals with both the load balancing and alleviating the large gather at the end by having each processor send back its results after calculating each block. This evens out the gather time so that it is not one massive operation so that the root processor can more easily handle the load.

We further explore the communication portion of the different Maronna versions in the following graphs. We were able to create rough communications profiles for the algorithms using a setup with several machines connected to a Juniper M5 router where each machine was configured to be on its own network. Then, by SNMP, we used the management port to query for the total TCP traffic. This setup allowed us to independently monitor the traffic without perturbing the execution. Using this, we created a communication profile by repeatedly querying the router for the traffic through the connections to monitor the activity as time passed. The communications traffic is measured in bytes over the queries we made. We could query the router at approximately 37 times per second. We grouped the results of 1000 of these queries together and report the sum of the traffic that the router reported for this period. The x-axis in the graphs represents a rough estimation of time, where we report total router traffic about every twenty-seven second interval. The reported traffic is measured in kilobytes.

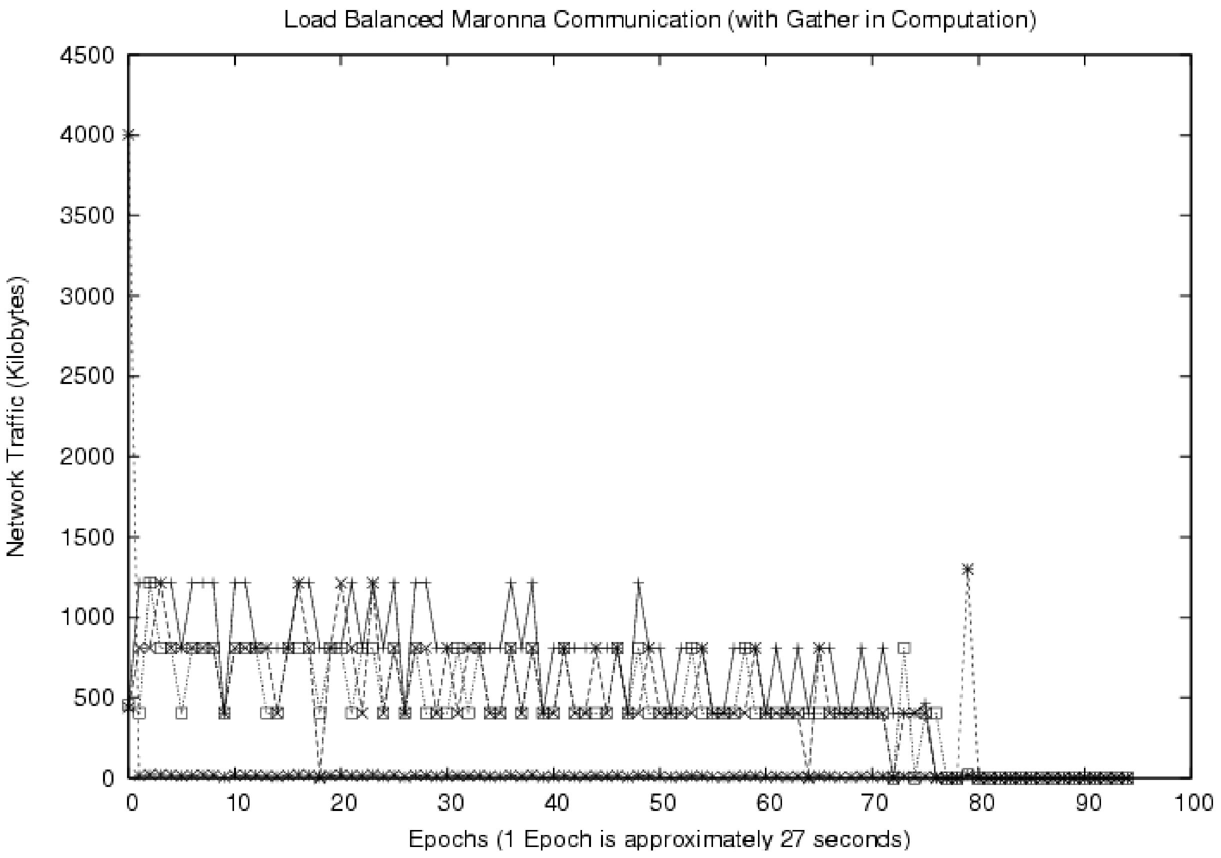

Figure 10: Load Balanced Maronna with Block Gather Communications Profile

Figure 9 shows the Maronna algorithm's profile. Maronna also has the heaviest traffic during the gather stage. There are two large communication points on the graph because one of our machines was slower than the rest. When the faster machines completed earlier, they had to wait for the slowest one to finish and meanwhile sat idly. This is one reason why load balancing is an improvement.

We have profiled two versions of the load balanced Maronna. The first has the processors returning their results after they calculate a block of correlations, and is shown in Figure 10. The height of all the communications here and their thickness in the graph are all related to the block size of the algorithm. Small blocks make the messages smaller and more frequent, while larger blocks make for larger messages that are not as frequent. Thus, the total traffic the network can handle is something to consider when choosing the block size.

The profile for the second load balanced Maronna is in Figure 11. It is similar to the original Maronna in that all the correlations are saved up until the end for one massive gather. The processors still send messages throughout the algorithm, but they are small and only serve as requests for more work.

Figure 12 shows the effect of static versus dynamic load balancing on the overall compu-

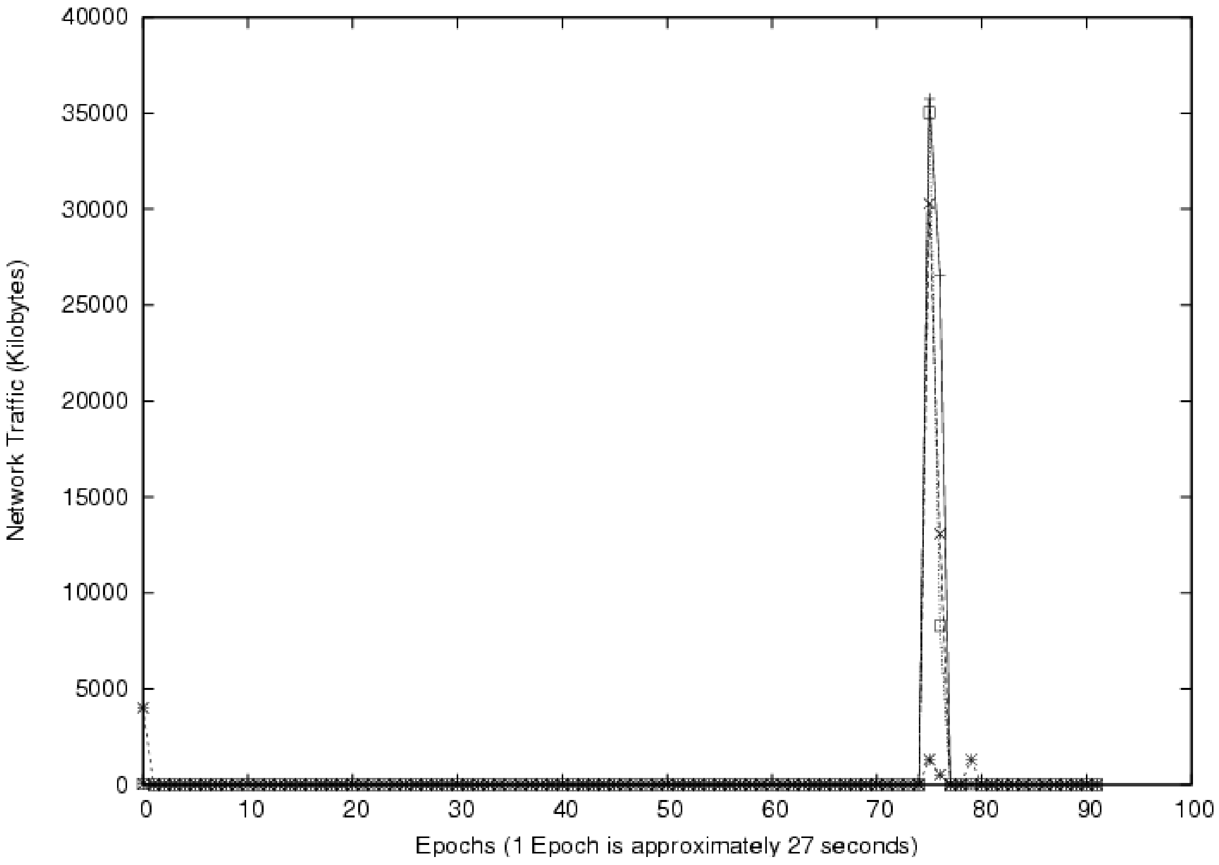

Figure 11: Load Balanced Maronna with End Gather Communications Profile

tation time. For this graph we used the block gather version of dynamic Maronna. The dynamic load balancing scheme and the spread-out gather operation combined for a $30\% - 40\%$ improvement in the overall runtime of the Maronna method. Thus, balancing the workload and spreading the gather operation are effective optimizations for the parallel Maronna algorithm.

4.4 Parallel QC Method

Next we turn our attention to the parallel QC algorithm. QC calculated its correlation using several vector operations and matrix multiplication. The performance of QC for a large problem is shown in Figure 13.

On 8 processors the wall clock time was 105 seconds, which was a speed-up of only 1.6 out of 8. Notice that QC executed faster than the Maronna method on the same problem. Again, the bars in Figure 6 show the major time components that made up the total time. A clear observation is that QC was not able to use the processors as effectively as the Maronna method, and at 8 processors there was little to be gained by adding more processors. It is evident from the figure that the correlation calculation was dominated by the communication

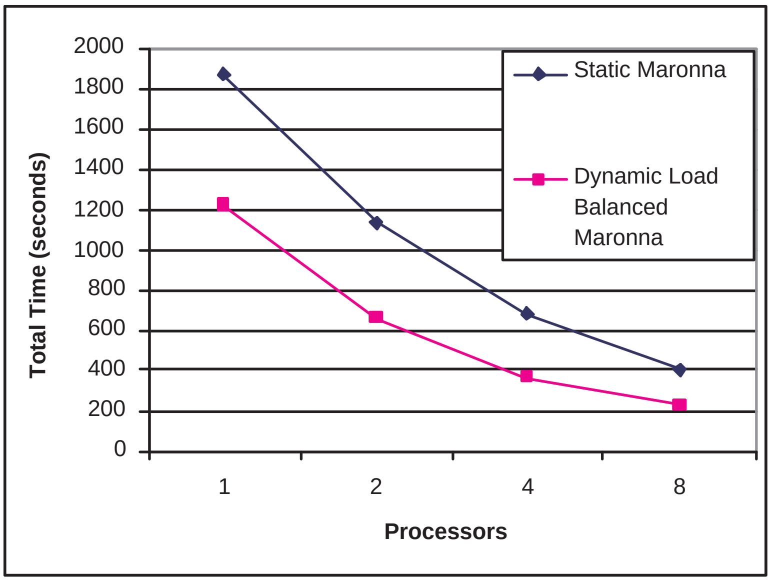

Figure 12: Static versus Dynamic Load-balancing

time and I/O time. At 8 processors the correlation time was only taking 5 seconds, a fraction of the overall execution time of 105 seconds. Figure 14 shows in greater details the correlation component of the calculation of a variety of problems sizes and machine sizes.

Except for the small problems sizes the speed-up was better than expected, in some cases super linear. The reason for the super linear speed-up related to how the PLAPACK library distributed matrix blocks to processors. It assumed a mesh formation and in these experiments attempted, as best possible, to arrange the processors in a square mesh. This distribution affected the performance, making it more difficult to determine exact speed-up numbers.

The distribution of blocks to processors also affected communication as well. The communication component for QC is shown in Figure 15. The gather portion of QC used a PLAPACK primitive call to assemble the distributed matrix into a continuous buffer on one processor. On one processor, the time shown for communication was actually a memory-to-memory copy. The large times for the copy was due to page faults and was not present when executed on a cluster with more memory.

It is difficult to find a model that accurately predicts the time performance of QC. Some components are the same as Maronna, such as the I/O routines and the Median and MAD calculations. Others are a bit more tricky because we do not really know what is happening inside the PLAPACK library. The distribution of data pieces between the processors can effect the performance, such as the communication and correlation computation components. Also, the correlation calculation, more specifically the matrix multiply, likely contains some hidden communication calls that are not apparent from the outside. Without knowing exactly how the matrix and vector objects are specifically distributed between the processors and the types of communications going on in the computation components, it is not likely

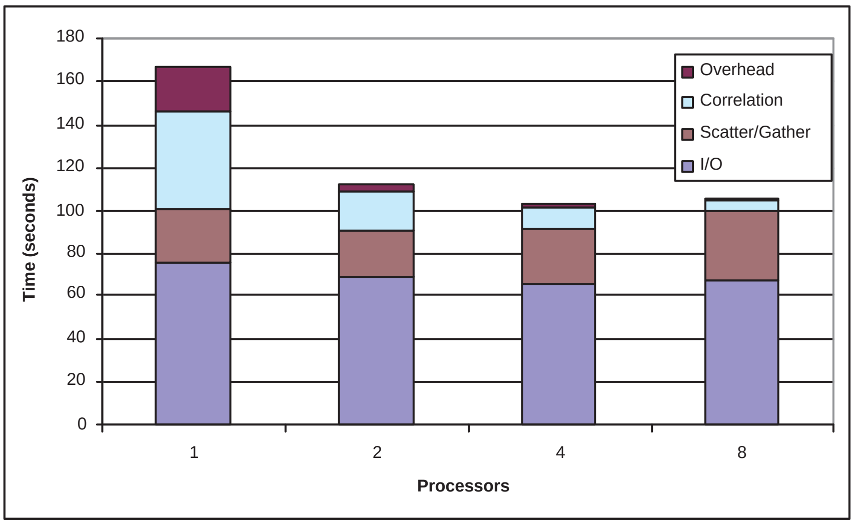

Figure 13: QC Performance for large problem size (6000 variables)

we can find an accurate model for QC.

4.5 Trading Accuracy for Time

The iterative nature of the Maronna method also lended itself to an alternative approach to speeding up the computation.

1. Iterate only a constant number of times.
2. Change the tolerance (ϵ) so that it terminates sooner.

These techniques are discussed in Sections 4.5.1 and 4.5.2.

4.5.1 Fixing the Number of Iterations

In experimenting on random data we noticed that all pairs converged in less than 5 iterations. The absence of slow converging pairs in the random data led us to hypothesize that it may be the highly correlated pairs that were converging slowly. However, this proved not to be the case. Further experiments showed that the rates of convergence were normally distributed between -1 and 1 and matched the distribution of the correlation values. Instead, by analyzing the slow converging pairs, we discovered that it was the outliers in the data that were taking longer to compute.

The outliers in the data are the values that have a large Mahalanobis distance and these distances were reduced according to the weight function given by Equation 1. The value of c used in this equation was Huber's constant (9.21) and this constant controlled the rate of convergence. The slow converging corrections had very large distances in comparison to the distances that converged quickly. We also experimented with a stricter weight function, where we down-weighted the outliers immediately to zero, and one that used $2 \cdot c$ rather

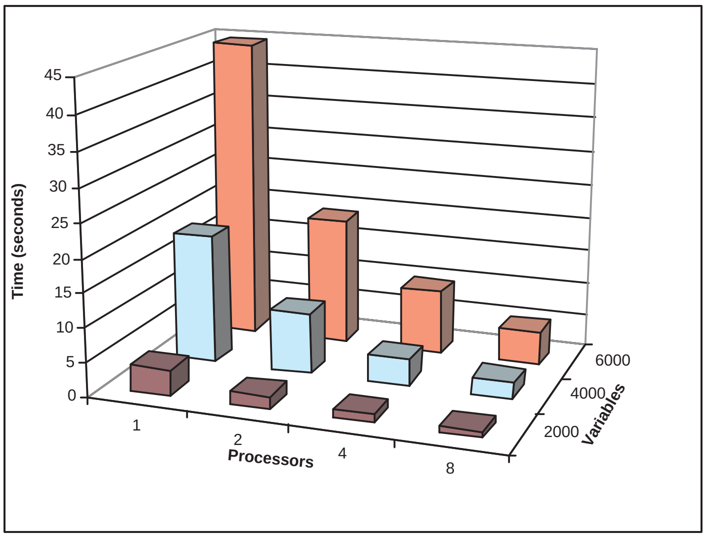

Figure 14: QC Correlation Component

than c. The strict weight function converged quickly but did not produce accurate values for the outliers. The $2 \cdot c$ decreased the iteration time and obtained good results but it may not hold in general.

Iterations	Correlations	Percent of Total Corrs
0-200	18396529	99.9416%
201-400	8665	0.04707%
401-600	1362	0.00740%
601-800	379	0.00206%
801-1000	127	0.00069%
1001-1200	78	0.00042%
1201-1400	53	0.00029%
1401-1600	28	0.00015%
1601-1800	22	0.00012%
1801-2000	9	0.00005%
2001-2200	7	0.00004%
2201-2400	4	0.00002%
>2400	19	0.00010%

Table 3: Correlation Convergence for Maronna on 6068 by 20 Gene Dataset with $\epsilon = 10^{-7}$

In conclusion, the good news from these experiments is that Maronna in only a few iterations computed the vast majority of the correlations. The bad news is that the small fraction of remaining pairs corresponded to outliers and cannot easily be truncated if accurate correlated values are required for these entries.

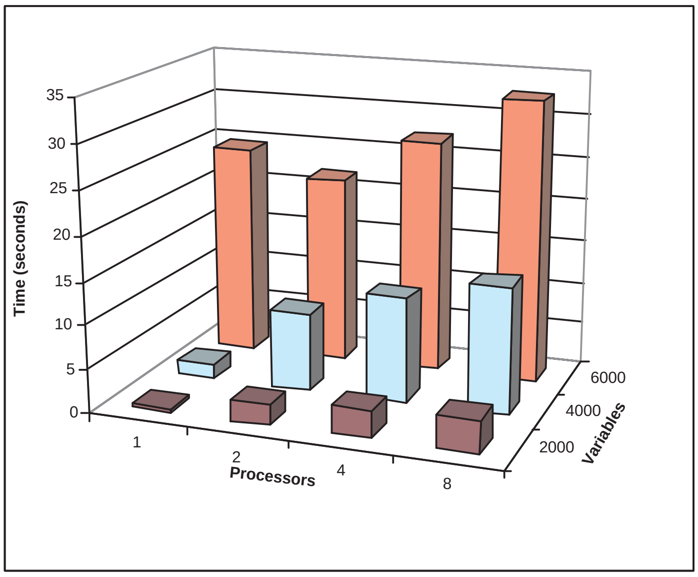

Figure 15: QC Communication Component

4.5.2 Changing the Tolerance

The Maronna algorithm contained a tolerance argument (ϵ) as a stopping condition that determined how close the iterated correlation estimate was to the real correlation value. By increasing the value of epsilon, the correlation values, including the slow converging ones, converged faster at the cost of accuracy.

We experimented with varying ϵ from 0.1 to 10^{-13}. At the most accurate setting, as expected, more correlations took more time to converge. At $\epsilon = 0.1$ the majority of correlations converged within 5 iterations and all converged within 30 iterations.

We defined the accuracy to be the absolute difference between a correlation estimate and the correlation's real value. We defined the accuracy of a correlation matrix estimate to be the largest of the accuracy values for the matrix's individual correlation entries compared to the corresponding entries in the real correlation matrix. To calculate the accuracy in practise, we used the correlation matrix estimate with the smallest ϵ for the machine, 10^{-13} for the gene data.

Figure 16 displays the trade off between changing epsilon and the convergence and accuracy for Maronna on the full gene data set with 6068 variables using 8 processors. By convergence, we mean how much does the resulting matrix differ from the matrix calculated with $\epsilon = 10^{-13}$. We considered entries in the computed matrix to be converged if they were within 10^{-7} of the corresponding entry in the $\epsilon = 10^{-13}$ matrix. Starting at the "gold standard" of $\epsilon = 10^{-13}$, the convergence holds steady as epsilon increases, but then hits a point and drops dramatically. The time improvement seems to increase linearly with epsilon. The data in this graph helped us determine what value of epsilon to choose for our experiments with Maronna. We chose $\epsilon = 10^{-7}$ because at this point, the near 30% time improvement comes at the cost of little accuracy since the matrix is 99.9% convergent with this epsilon.

Compared to the $\epsilon = 10^{-13}$ matrix, the matrix computed by QC had 0.0168% of its entries within our convergence range. QC did have good time improvement at 40.3%, which is comparable to Maronna using $\epsilon = 10^{-4}$.

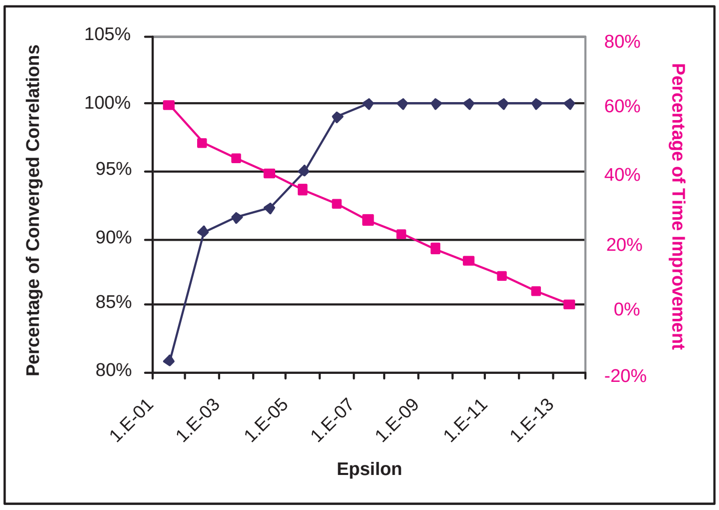

Figure 16: Accuracy versus Time Improvement and Convergence to $\epsilon = 10^{-13}$ Matrix

Figure 17 shows a graph showing how the different values of ϵ and truncated iterations affected the running time. The differences in time for the various ϵ show that we gained some improvement with a careful choice of epsilon. This improvement helps in both cases of slow converging correlations, whether a single slow converging correlation at the end of a processor's batch, or an unbalanced load to a processor since all the correlation calculations benefited with faster convergence for a smaller epsilon.

4.6 Scalability on a Grid Platform

From the previous figures, it is clear that the Maronna method was more amenable to parallelization than QC. The question to be answered is when the speed-up will stop for the Maronna method. To answer this question, we ran the parallel Maronna and QC on the WestGrid cluster using up to 128 processors on the gene data set. There was some variation in the experiments. Most of the variation came in the category of I/O time, and times varied by as much as 170 seconds for Maronna and 30 seconds for QC. Each experiment was run ten times and used the smallest repeatable value. Although the processors were not shared, the network and file system were shared, resulted in varying times. The smallest value best reflected what would be possible on a dedicated machine.

Figure 18 shows the speedup for QC and Maronna on up to 128 processors. It also has the data points labelled with the total runtimes for Maronna and the runtimes for QC on eight or more processors. As an example, the Maronna method using 128 processors has a speedup of 24.1 and a total runtime of 15.5 seconds. For the Maronna method in Figure 18, the total runtime continued to decrease as we add processors. The time decreases from 374.7

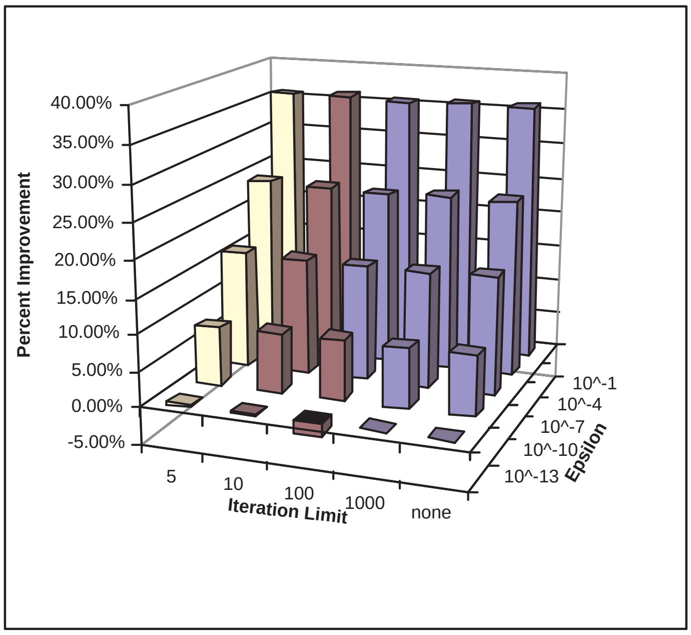

Figure 17: Time versus Number of Iterations versus Tolerance

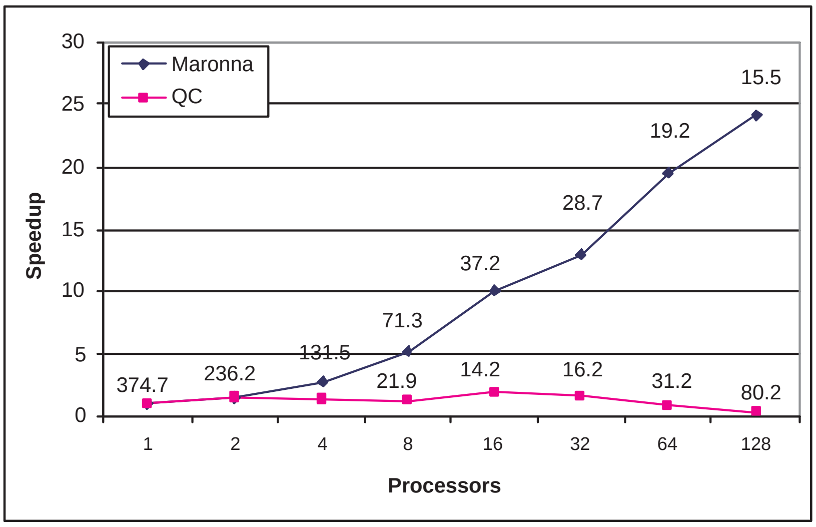

Figure 18: Speed-up on WestGRID

seconds on a single machine to 15.5 seconds with 128 processors. We were surprised at how well Maronna performed on a large cluster. One may expect to have saturated the manager processor since it is the only one allocating and distributing tasks. The fact that this did not occur, even when the total execution time was 15.5 seconds, suggests that the Maronna method will continue to scale well to larger problems and processor sizes. For a problem size of 6068, at 128 processors, there is little to be gained by adding more processors.

As expected from previous discussions, the speed-up curve for QC in Figure 18 shows that QC quickly reached the point that it can not effectively use more processors. The time did continue to decrease with up to 16 processors but at that time communication overheads began to dominate and the time began to increase. The good news is that QC executed quickly on large problems and was able to exploit a small degree of parallelism. We see that the total times for QC are much smaller than Maronna up to the point where QC is overcome by too much overhead. One may be able to use more processors to solve larger problems using QC. However, the communication overheads in QC were more significant than in Maronna.

4.7 Median/MAD Calculation

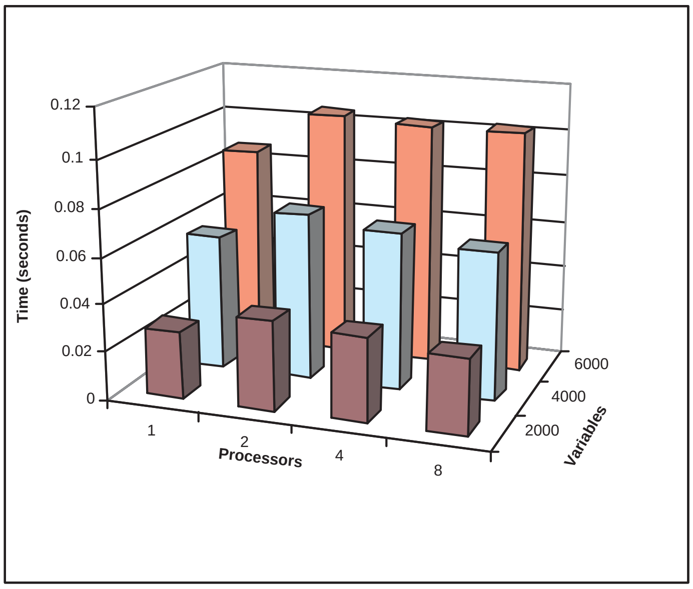

Figure 19: The Median/Mad Component

The Median/MAD component is shown in Figure 19. It did not achieve the same speed-up as the correlation calculation. However, note that even on the largest problem size, the Median/MAD calculation took less than one second, which is insignificant compared to the correlation calculation. Given the amount of computation required, the time overhead of communicating between processors outweighed the advantages of attempting to do the calculation in parallel. The latter may be more useful for very large data sets.

We performed experiments where the variables were fixed at 6000 and the cases in the data set were varied by generating cases based on a weighted sum of other rows of data chosen at random. This allows us to suggest a model for the Median/MAD component, though it is difficult because the Median/MAD contains both calculation and communication mixed together. The Median/MAD times for our experiments and the predicted and error values are listed in Table 4.

		Experimental	Predicted	Percent
v	p	Time	Time	Error
100	1	2.056	2.414	14.82%
	2	0.949	1.143	16.96%
	4	0.706	0.596	18.45%
	8	0.579	0.500	15.92%
1000	1	26.197	26.894	2.59%
	2	12.504	14.193	11.90%
	4	8.558	8.741	2.09%
	8	6.641	7.812	15.00%

Table 4: Dynamic Load-balanced Maronna Communication Time versus Predicted Values

This model is based on the equation $\mathcal{O}(\frac{nv}{p})+\mathcal{O}(np)+\mathcal{O}(p)$. The $\mathcal{O}(\frac{nv}{p})$ term relates to the actual Median finding work. The other terms are mixtures of overhead and communication, as the processors must share the Median and MAD values after they are calculated, and must divide up the data matrix to find them. Again, the error values show this equation is not perfectly accurate, but it suffices to show that the Median/MAD calculation contains several different terms.

5 Conclusion

This paper has shown that robust methods for calculating high dimensional correlation and covariance matrices are feasible when implemented in parallel. These methods now make it possible to not only solve for large correlation and covariance matrices in a timely fashion, but also compute them with a more robust approach.

Our experiments were performed on a real data set with 6068 variables representing the expression levels of genes in 20 patients. The results show that QC scales well for up to 8 processors. Maronna is able to scale up much further beyond since the computation portion requires no communication between processors. This helps Maronna to achieve speed-up on more than 8 processors, up to 128 as can be seen from the WestGRID results. QC is still faster, but Maronna is more robust and scalable to more processors. We examined Maronna closely and found that some correlations, namely the ones involving outlier data values, converge at a slower rate. In response, we developed a load balanced version of Maronna which provided a vast improvement in running time and also gives us a more efficient algorithm for heterogeneous clusters of machines.

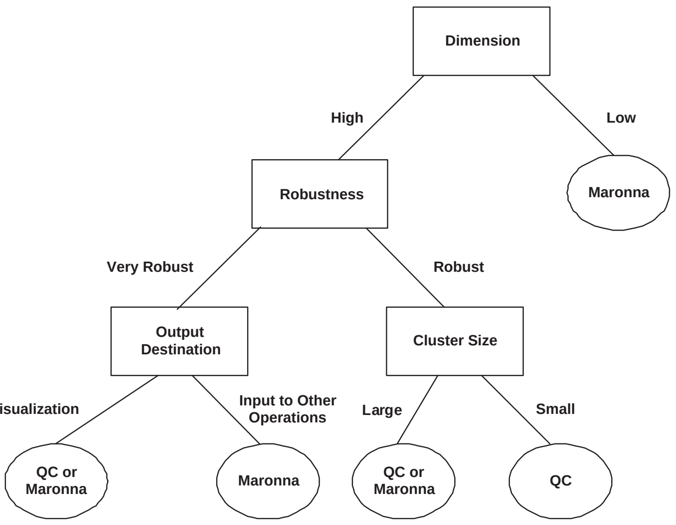

Figure 20: Recipe for Choosing a Parallel Robust Correlation/Covariance Algorithm

We also analyzed the algorithms' component-wise performance in order to generate an equation describing their behaviour. This showed us the reasons why QC and Maronna gave different results as parallel algorithms.

The QC and Maronna algorithms are good for solving different types of problems. We have created a recipe in Figure 20 to suggest which algorithm to use based on the given resources and needs. With low dimensional data, the Maronna method gives robust results and is not overloaded in large amounts of computation. With high dimensional data however, the choice depends on the application's need for robustness. If only a moderate degree of robustness is necessary, and resources are limited to small clusters, then QC works best. If a large cluster is available, either QC or Maronna works well. On the other hand, for very robust applications, the purpose of the calculation may be considered. If the covariance matrix is needed for other calculations, it is best to use the Maronna method because of its higher quality results. If the output is intended only for preliminary exploration or visualization, then QC may be chosen for its performance.

References

[1] M. B. Abdullah. On a robust correlation coefficient. *The Statistician*, 39:455–460, 1990.

[2] F. A. Alqallaf, K. P. Konis, and R. D. Martin. Scalable robust covariance and correlation estimates for data mining. In *Proceedings of the Seventh ACM SIGKDD*, pages 455–460, 1990.

[3] S. J. Devlin, R. Gnanadesikan, and J. R. Kettenring. Robust estimation of dispersion matrices and principal components. *Journal of the American Statistical Association*, 76:354–362, 1981.

[4] D. Donoho. *Breakdown properties of multivariate location estimators.* PhD thesis, Harvard University, 1982.

[5] M. P. I. Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Department of Computer Science, University of Tennessee, 1994.

[6] R. Gnanadesikan and J. R. Kettenring. Robust estimates, residuals, and outlier detection with multiresponse data. *Biometrics*, 28:81–124, 1972.

[7] P. J. Huber. *Robust Statistics.* John Wiley & Sons, 1981.

[8] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

[9] E. M. Knorr, R. T. Ng, and R. H. Zamar. Robust space transformations for distance-based operations. In *Knowledge Discovery and Data Mining*, pages 126–135, 2001.

[10] R. Maronna and R. Zamar. Robust estimates of location and dispersion for high dimensional data sets. *Technometrics*, 2002. to appear.

[11] R. A. Maronna. Robust m-estimators of multivariate location and scatter. *The Annals of Statistics*, 4(1):51–67, 1976.

[12] R. A. Maronna, W. A. Stahel, and V. Yohai. Bias-robust estimation of multivariate scatter based on projections. *Journal of Multivariate Analysis*, 42:141–161, 1992.

[13] R. A. Maronna and V. Yohai. The behaviour of the Stahel-Donoho robust multivariate estimator. *Journal of the American Statistical Association*, 90(429):330–341, 1995.

[14] R. D. Martin and R. H. Zamar. Asymptotically min-max bias-robust M-estimates of scale for positive random variables. *Journal of the American Statistical Association*, 84:494–501, 1989.

[15] P. Rousseeuw and V. Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41:212–223, 1999.

[16] P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection.* John Wiley & Sons, 1987.

[17] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, pages 871–880, Dec 1984.

[18] P. J. Rousseeuw. Multivariate estimation with high breakdown point. In *Mathematical Statistics and Applications*, pages 283–297. Reidel Publishing, 1985.

[19] W. Stahel. Breakdown of covariance estimators, 1981. Research Report 31, Fachgruppe fur Statistik, ETH, Zurich.

[20] R. A. van de Geijn. *Using PLAPACK*. Scientific and Engineering Computation Series. MIT Press, 1997.

[21] Westgrid: Western canada research grid. www.westgrid.ca.