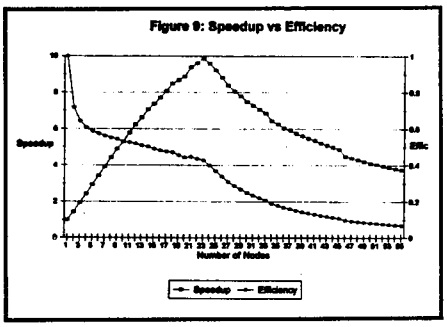
Parallelizing Locally-Weighted Regression

Julia Corbin Fauntleroy
and
Edward J. Wegman

Technical Report No. 102
October, 1994

Center for Computational Statistics



George Mason University
Fairfax, VA 22030

CENTER FOR COMPUTATIONAL STATISTICS
TECHNICAL REPORT SERIES (RECENT REPORTS)

TR 93. Winston C. Chow, Modeling and Estimation with Fractional Brownian Motion and Fractional Gaussian Noise (Ph.D. Dissertation), February, 1994.

TR 94. Mark C. Sullivan and Edward J. Wegman, Correlation Estimators Based on Simple Nonlinear Transformations, February, 1994, To appear **IEEE Transactions on Signal Processing.**

TR 95. Mark C. Sullivan and Edward J. Wegman, Normalized Correlation Estimators Based on Simple Nonlinear Transformations, March, 1994.

TR 96. Kathleen Perez-Lopez and Arun Sood, Comparison of Subband Features for Automatic Indexing of Scientific Image Databases, March, 1994.

TR 97. Wendy L. Poston and Jeffrey L. Solka, A Parallel Method to Maximize the Fisher Information Matrix, June, 1994.

TR 98. Edward J. Wegman and Charles A. Jones, Simulating a Multi-target Acoustic Array on the Intel Paragon, June, 1994.

TR 99. Barnabas Takacs, Edward J. Wegman and Harry Wechsler, Parallel Simulation of an Active Vision Model, June, 1994.

TR 100. Edward J. Wegman and Qiang Luo, Visualizing Densities, October, 1994.

TR 101. Daniel B. Carr, Converting Tables to Plots, October, 1994.

TR 102. Julia Corbin Fauntleroy and Edward J. Wegman, Parallelizing Locally-Weighted Regression, October, 1994.

TR 103. Daniel B. Carr, Color Perception, the Importance of Gray and Residuals on a Choropleth Map, October, 1994.

TR 104. David J. Marchette, Carey E. Priebe, George W. Rogers and Jeffrey L. Solka, Filtered Kernel Density Estimation, October, 1994.

TR 105. Jeffrey L. Solka, Edward J. Wegman, Carey E. Priebe, Wendy L. Poston and George W. Rogers, A Method to Determine the Structure of an Unknown Mixture Using the Akaike Information Criterion and the Bootstrap, October, 1994.

TR 106. Wendy L. Poston, Edward J. Wegman, Carey E. Priebe and Jeffrey L. Solka, A Contribution to the Theory of Robust Estimation of Multivariate Location and Shape: EID, October, 1994.

TR 107. Clifton D. Sutton, Tree Structured Density Estimation, October, 1994.

TR 108. Charles A. Jones, Simulating a Multi-target Acoustic Array on the Intel Paragon (M.S. Thesis), October, 1994.

Parallelizing Locally-Weighted Regression

By

Julia Corbin Fauntleroy

and

Edward J. Wegman

Abstract:

This paper focuses on a nonparametric regression technique known as locally-weighted regression or LOESS. LOESS is a computationally intensive technique which makes it naturally amenable to exploiting high performance computers. In this paper, we explore domain decomposition techniques for LOESS and study the performance of our algorithm on an Intel Paragon XP/S A4 machine. We study both speedup and efficiency as a function of the number of nodes. Certain segments of the LOESS computation are shown to be fruitfully parallelized while others are essentially sequential and cannot be parallelized effectively.

Introduction

Regression analysis is a statistical methodology used to predict values of one or more response variables from a group of predictor variables. This methodology results in the following model

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon, \text{ with } \beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

where $\mathbf{Y}$ is the response variables, $\mathbf{X}$ are the predictor variables, β is the vector of regression coefficients, and ε are the error terms. The purpose of regression analysis is to find the estimate of β that best fit the data. The regression coefficients are selected using a least squares criterion and referred to as the least squares estimates of β.[2]

Locally-weighted regression, or loess, is a non-parametric method for fitting a regression surface using multivariate smoothing. Instead of a single fit over all X, a local neighborhood of size k is determined for each x_i and a weighted-regression model is formed for that neighborhood. This model is similar to the regression model and can be expressed as

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon, \text{ where } \beta = (\mathbf{X}^T\mathbf{W}_i\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}_i\mathbf{Y} \text{ and}$$

$$\underset{kxk}{\mathbf{W}_i} = \begin{vmatrix} w_1 & \cdots & & \cdots & 0 \\ & w_2 & & & \\ & & \cdots & & \\ & & & \cdots & \\ 0 & \cdots & & & w_k \end{vmatrix}.$$

W_i is a diagonal matrix of weights where w_j is the weight of the jth observation in the neighborhood of x_i.The weights are determined so that points closer to x_i are weighted more than those further away from x_i. The size of the local neighborhood is a fraction of the total sample size, n.[3]

To calculate a loess estimate, n regression models must be formed. When the sample size is small, the algorithm can quickly calculate the fitted value for each data point. When the sample size becomes large, the number of computations performed requires an extensive amount of computer time. This paper demonstrates the use of parallel computing to reduce the time for computing fitted regression estimates for each data point in large-scale problems.

Constructing a Parallel Algorithm

The code for the local regression routine was obtained from AT&T Bell Labs at netlib.att.com. It originated from research by William S. Cleveland, Eric Grosse, and Ming-Jen Shyu and is the loess() routine used by S-Plus, a statistical software package from StatSci. The code consists of both C and Fortran subroutines. Only the portion of code for computing exact regression estimates was used for this project.

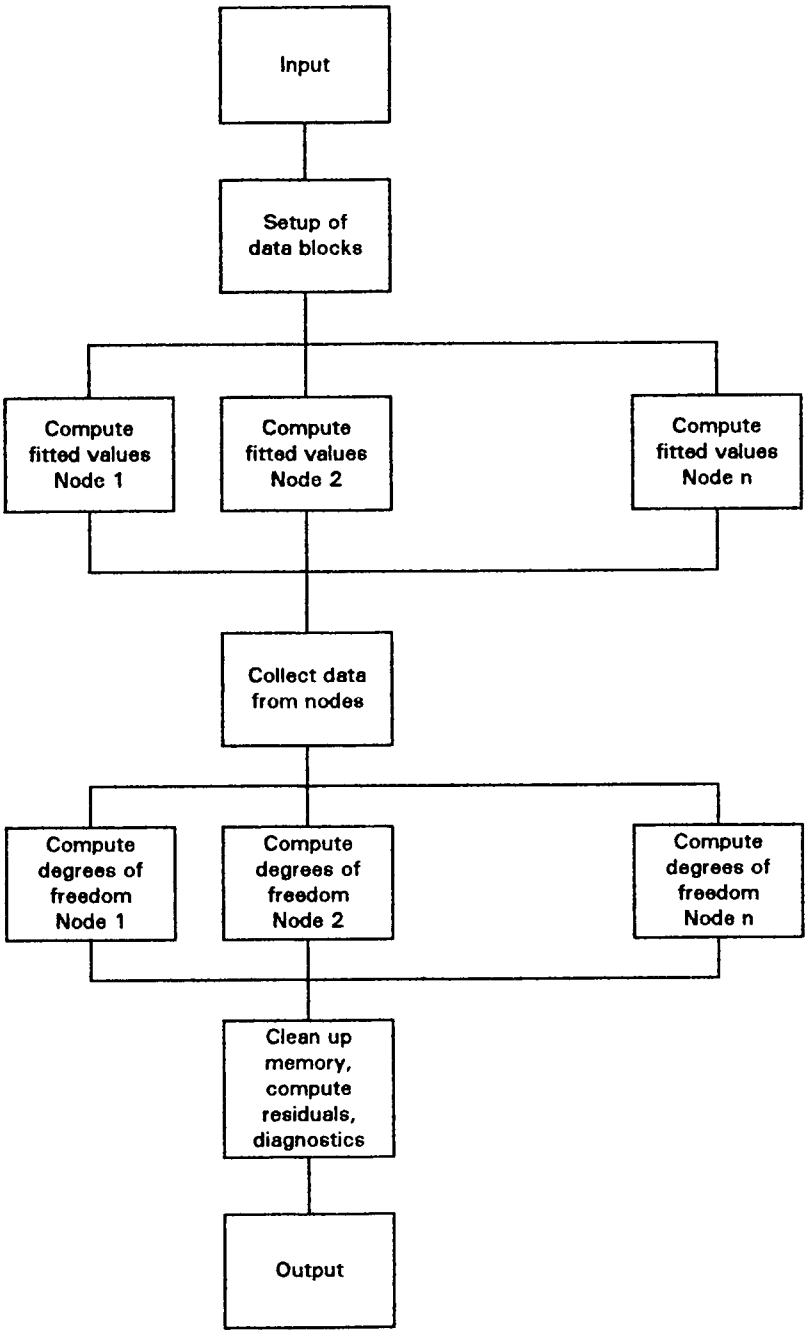
The first step towards parallelizing the problem was to examine the code in a purely sequential setting. The program was split into several sections. As illustrated in Figure 1, each section represents either sequential processes or candidate processes for parallelization. The sequential processes handled the setup and cleanup of internal working storage, computation of residuals, and residual diagnostics. The candidate processes were the computation of the fitted regression values and the degrees of freedom (or effective number of parameters). These portions of code perform repetitive computations, require a longer time to compute, and can easily be translated to a parallel environment. Table 1 shows the breakdown of computation time for the sequential processing of the complete application using a sample size of 500.

Section of Application	Time (in sec.)
1. Setup of working storage	0.6805
2. Compute fitted values	4.1304
3. Compute degrees of freedom	111.2922
4. Compute residuals, residual diagnostics, cleanup	0.6672
Total Time	116.7703

Table 1: Breakdown of sequential time for 500 data points.

As seen in the table, the largest computation time occurs when the degrees of freedom are computed. In Figure 2, it also can be seen that the time required for computing the degrees of freedom increases at a much faster rate than that of computing the fitted values. This is because the

Figure 1: Breakdown of Algorithm



Sequential Process

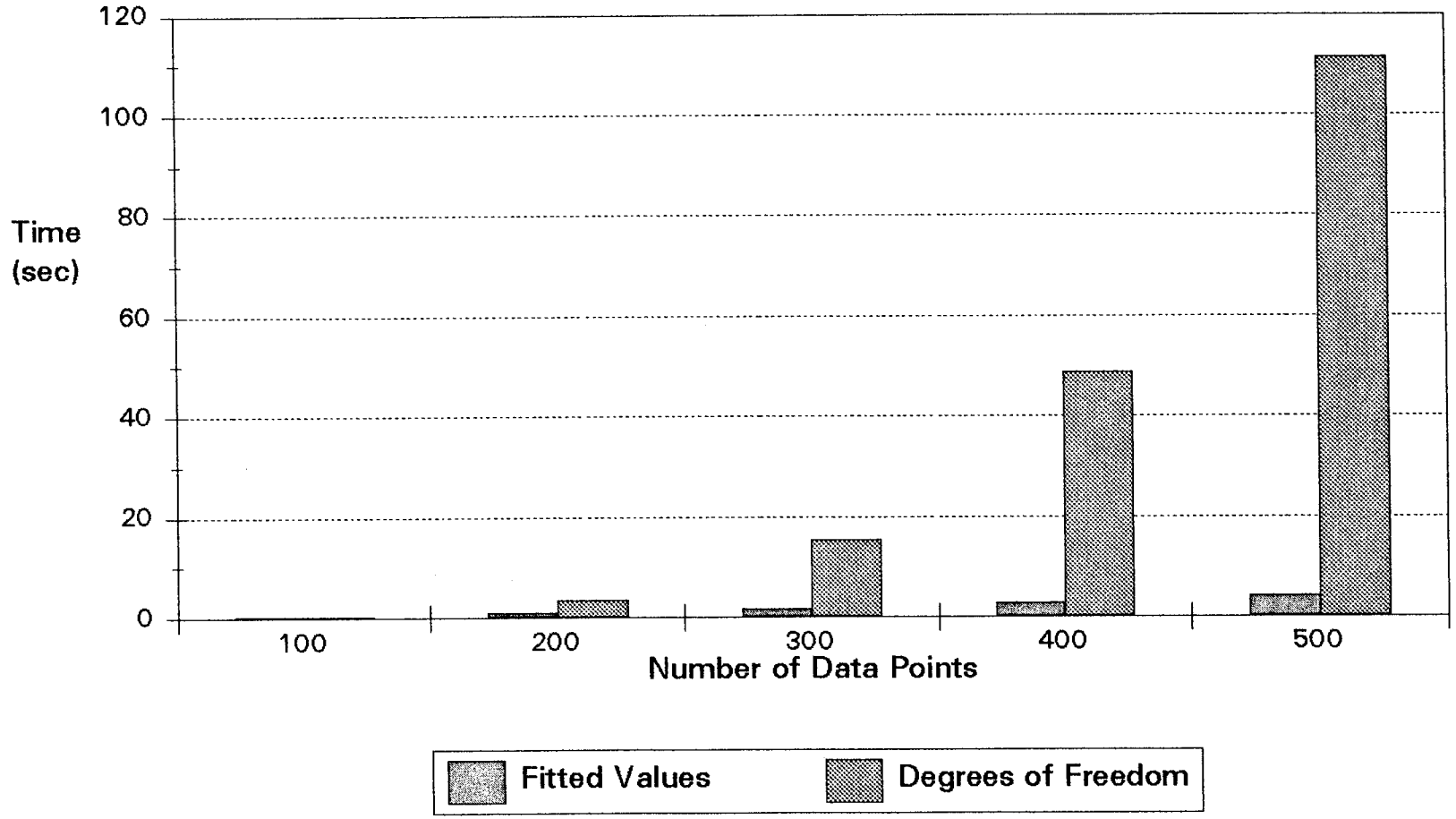
Candidate Process #1

Sequential Process

Candidate Process #2

Sequential Process

Figure 2: Computation Time of Candidate Processes



degrees of freedom computation requires $O(n^2)$ calculations. The number of calculations for this process grows at a rapid rate as n gets large (Figure 3). When this process is parallelized a significant improvement in the calculation times should be seen as the calculations are spread across the nodes.

On the other hand, the computation of the fitted values requires a much smaller number of calculations. This may be surprising since a regression model is being fitted for every data point, but given the fact that the model is fitting only a small number of points in the local neighborhood, the number of calculations is of $O(n_k)$ where n_k is the size of the local neighborhood. This keeps the computation time small. For purposes of this paper, both the processes will be moved to a parallel environment. However, the benefit in parallelizing the computation of fitted values will be small.

After the candidate processes were determined, each section of code was examined to decide the best approach to take in parallelizing the routines. As the outlines show, the data are processed in a sequential manner in both candidate processes; i.e. a loop from 1 to n. In addition, no information from a calculation in one iteration of the loop is required at the next iteration. This means that the calculations can be performed on separate nodes.

Candidate Process #1 - Compute Fitted Values

The loess estimate for a point x_0 is computed in the following way.[5]

1. Identify the k-nearest neighbors of x_0, for k = N*span where the span is percentage of the data provided by the user.
2. Compute the maximum distance of the k-nearest-neighbors and x_0.
3. Compute the weights for the k-nearest-neighbors.
4. Fit the k-nearest-neighbors using weighted least-squares.

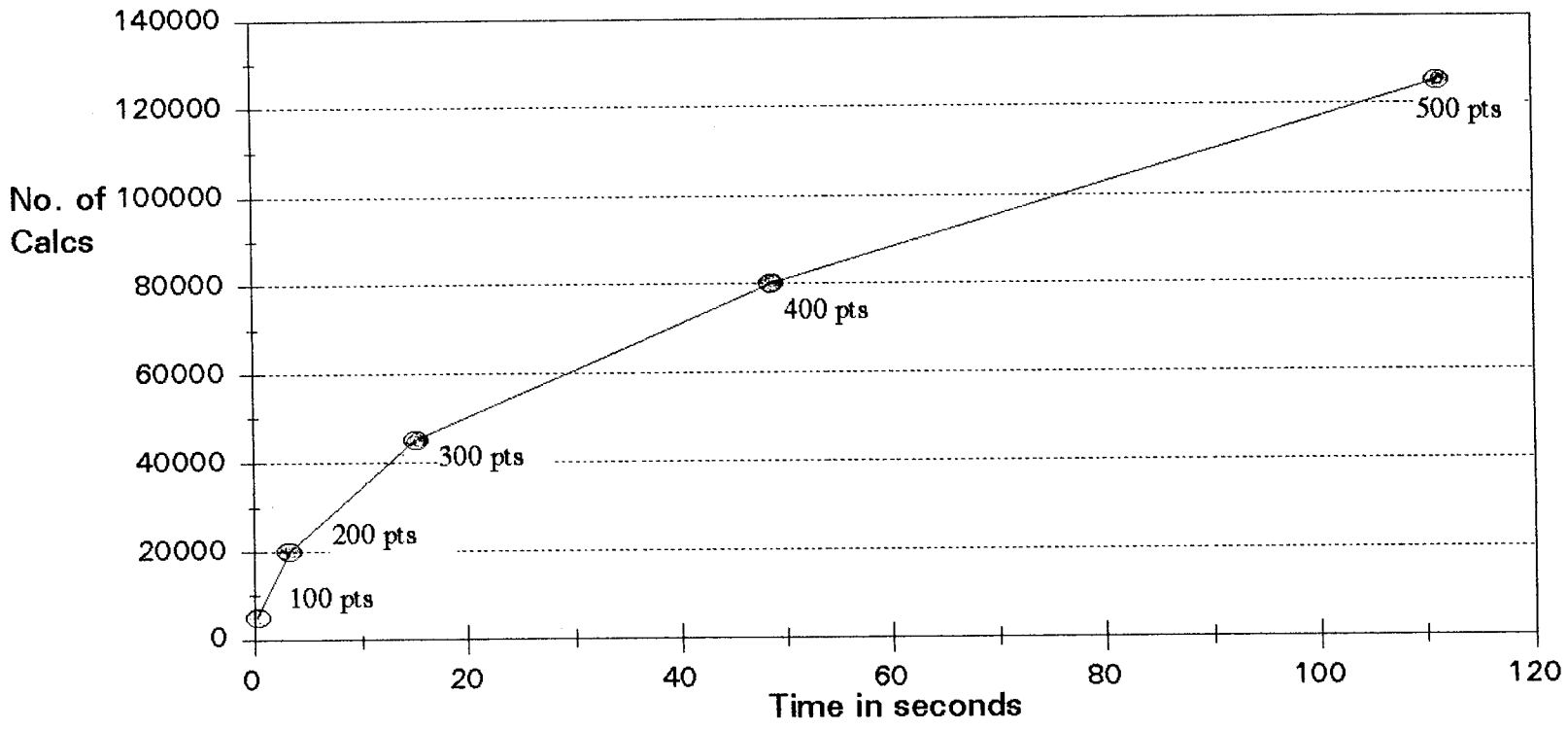
Candidate Process #2 - Compute degrees of freedom

Using Hat Matrix (H) compute

1. $H_0 = I - H$ where I is the Identity matrix
2. $\delta_k = tr(H_0'H_0)^k$
3. $\rho = \delta_1^2/\delta_2$ where ρ is the degrees of freedom [6]

There are several ways to parallelize these algorithms. The calculations for each data point requires the multiplication of several matrices. The matrix algebra could be vectorized but the time required for the matrix calculations at each data point is minimal and the communication overhead for number of calculations required would probably increase the computation time. Since vectorization is not practical, the best approach would be to distribute the data across the nodes if the data set is extremely large.

Figure 3: Calculations vs Time



Implementation of the Algorithm

To determine the computation time of the parallelized routines, five samples of 100, 200, 300, 400, and 500 data points with two parameters were used. These points were generated from random variables where $X_1 \sim N(0,1)$, $X_2 \sim E(1)$, and $Y \sim N(0,1)$. Each sample was processed across N nodes for N=1,2,..,56. The timings that were collected during the runs were based on elapsed system time. The candidate processes to be parallelized are the computation of fitted regression values and the computation of degrees of freedom.

Each candidate process was evaluated to determine the speedup and efficiency due to the parallelization. The speedup (S) of the program is defined as the ratio of the time it takes for the application to execute on a single node (T_1) and the time it takes for the application to execute over p nodes (T_p). This says that the ideal time for each node to process is T_1/p, which implies that $S_{ideal} = T_1/T_p = T_1 / (T_1/p) = p$. The speedup is effected by algorithmic constructs, overhead created by node initialization and implementation, load balancing, and communication overhead. The efficiency of the application is determined by the ratio of the speedup to the number of nodes used; $\varepsilon = S/p$.[1] Efficiency describes how well the algorithm was parallelized. The more work a single processor has to do because of the parallelization, the lower the efficiency.[4] If ideal speedup is achieved then the efficiency of the application is 1.0. Since the ideal speedup will most likely not be achieved, there becomes a tradeoff between loss of efficiency and increased speedup.[1]

Results

As illustrated in Figure 4, the computation of fitted values showed some improvement as the number of nodes increased but after the addition of more than four nodes, the speedup of the process began to decline. Eventually, running the process across a large number of nodes resulted in worse times than running it on a single node. It is suspected that the communication overhead is having an effect on the computation time. It is interesting to note that as the sample size increases the speedup decreases over all nodes. This indicates that this process has better performance in a sequential environment.

On the other hand, the computation of the degrees of freedom showed a dramatic had a dramatic improvement in speedup, as seen in Figure 5. But like the first process, the speedup begins to drop off as the number of nodes increases past some maximum speedup value. As the

Figure 4: Speedup
Computed Regression Values

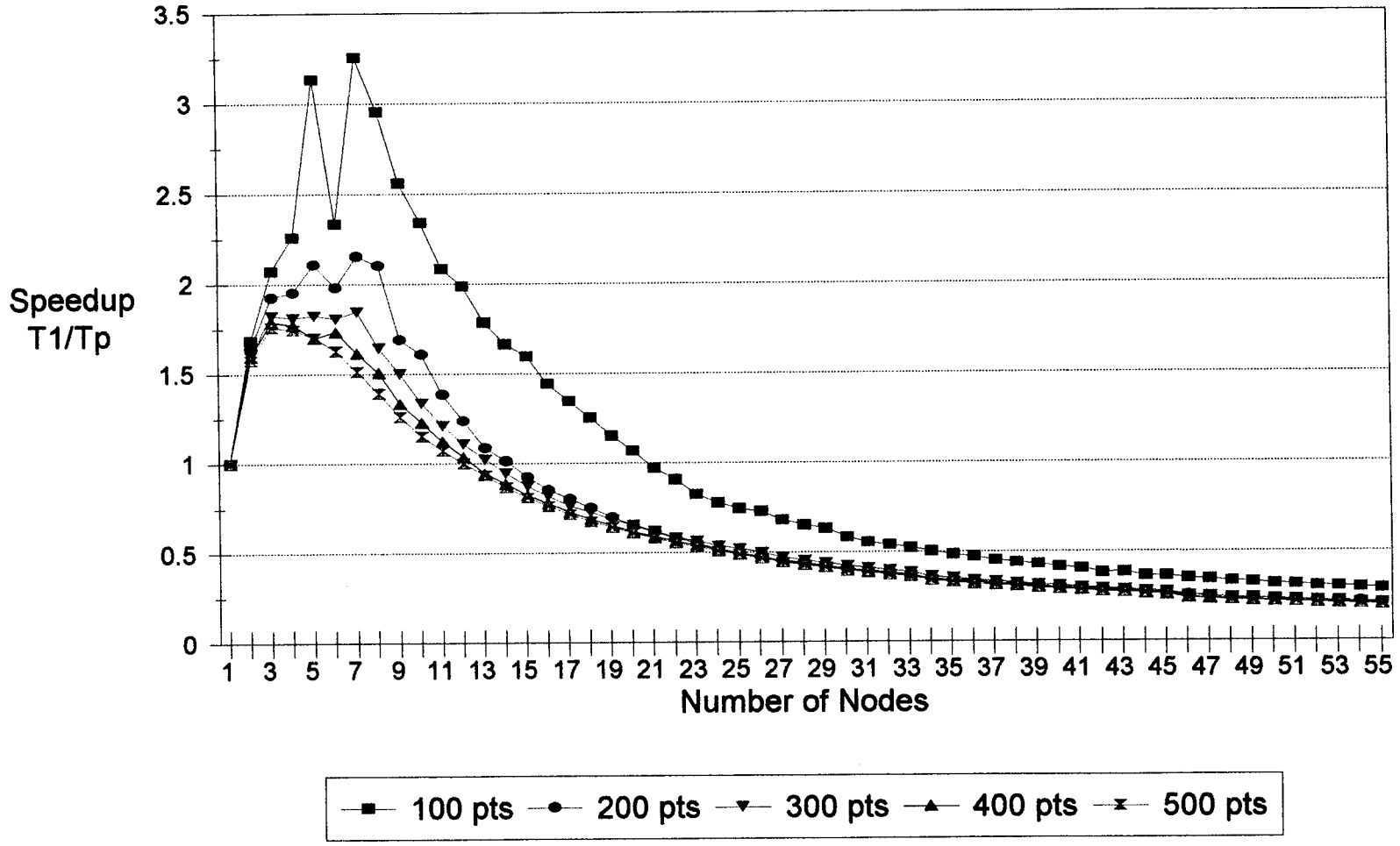


Figure 5: Speedup
Degrees of Freedom

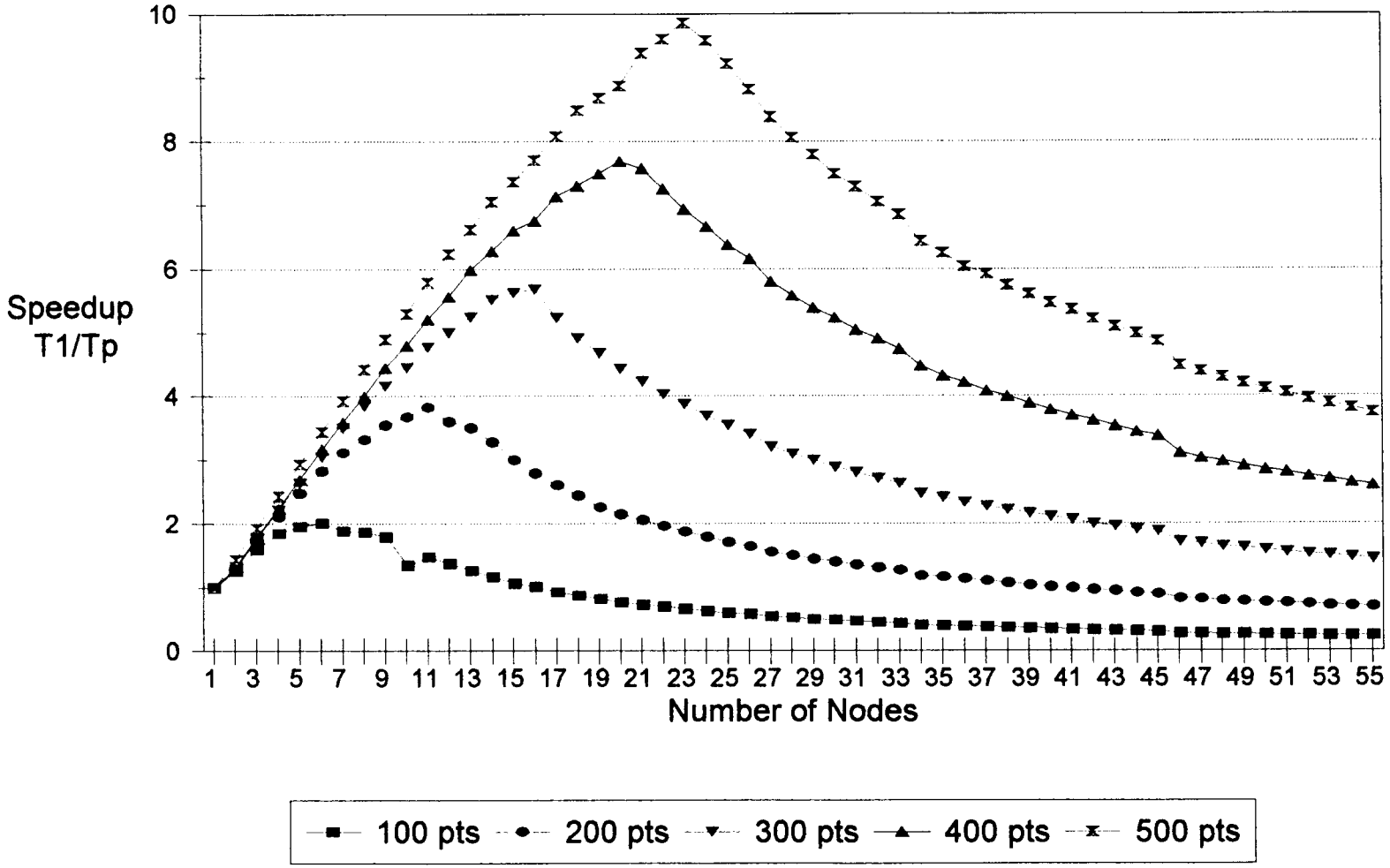


figure also shows, when the sample size increases, the number of nodes required for the maximum speedup also increases.

While the speedup of the process has been greatly improved, at no time does it reach its ideal value. Since the code was not optimized for a parallel environment, there is probably some overhead in the software that effects the speedup. In addition, there is certain to be communication overhead as the number of nodes increases.

In the first approach at parallelizing the second algorithm, the times seemed to have a lot of noise in them as the number of nodes increased. After examining the data it became apparent that high speedup values occurred when the data was evenly distributed across the nodes. Low speedup values occurred, when $n - \lfloor n/p \rfloor \neq 0$. A review of the code, pointed to a load balancing problem. The distribution of data to the nodes was changed so that a node received either $\lfloor n/p \rfloor$ or $\lfloor n/p \rfloor + 1$ data points. This lessened the impact that load balance had on speedup. The results of this modification can be seen in Figure 6.

If we look at efficiency, in Figures 7 & 8, it is easily seen that efficiency drops off fairly quickly when more nodes are added. However, as the sample size increases, the efficiency tends to increase overall the nodes. This means that while we do get a significant speedup using a large number of nodes, the algorithm is not performing as well as expected in the parallel environment. The algorithm was not optimized for parallelization and this may have an impact on the efficiency. Figure 9 illustrates the interaction of efficiency and speedup. The point where the efficiency begins to fall below the speedup is where the interaction of the system starts to have an effect on speedup. This effect may be a result of how the algorithm is processed by the node, the quantity of data passed, or some other system dependent factor.[4] An interesting phenonmenon occurs when comparing the crossover points of each sample size (see Figure 10). The points where speedup and efficiency lines cross for each sample size appear at exactly the same node for each sample size. This indicates that there is some system interaction with the algorithm that is not data dependent. Additional research is required to determine the cause of this system interaction.

Conclusions

The results of this study show that parallelizing the computation of the degrees of freedom significantly improves the performance of the application for large sample sizes. The computation of the fitted regression values has little if any improvement and should probably be left as a sequential process. This problem was a good example for showing that the distribution of the

Figure 6
Load Balancing - Sample size 500
(#) - size of node imbalance

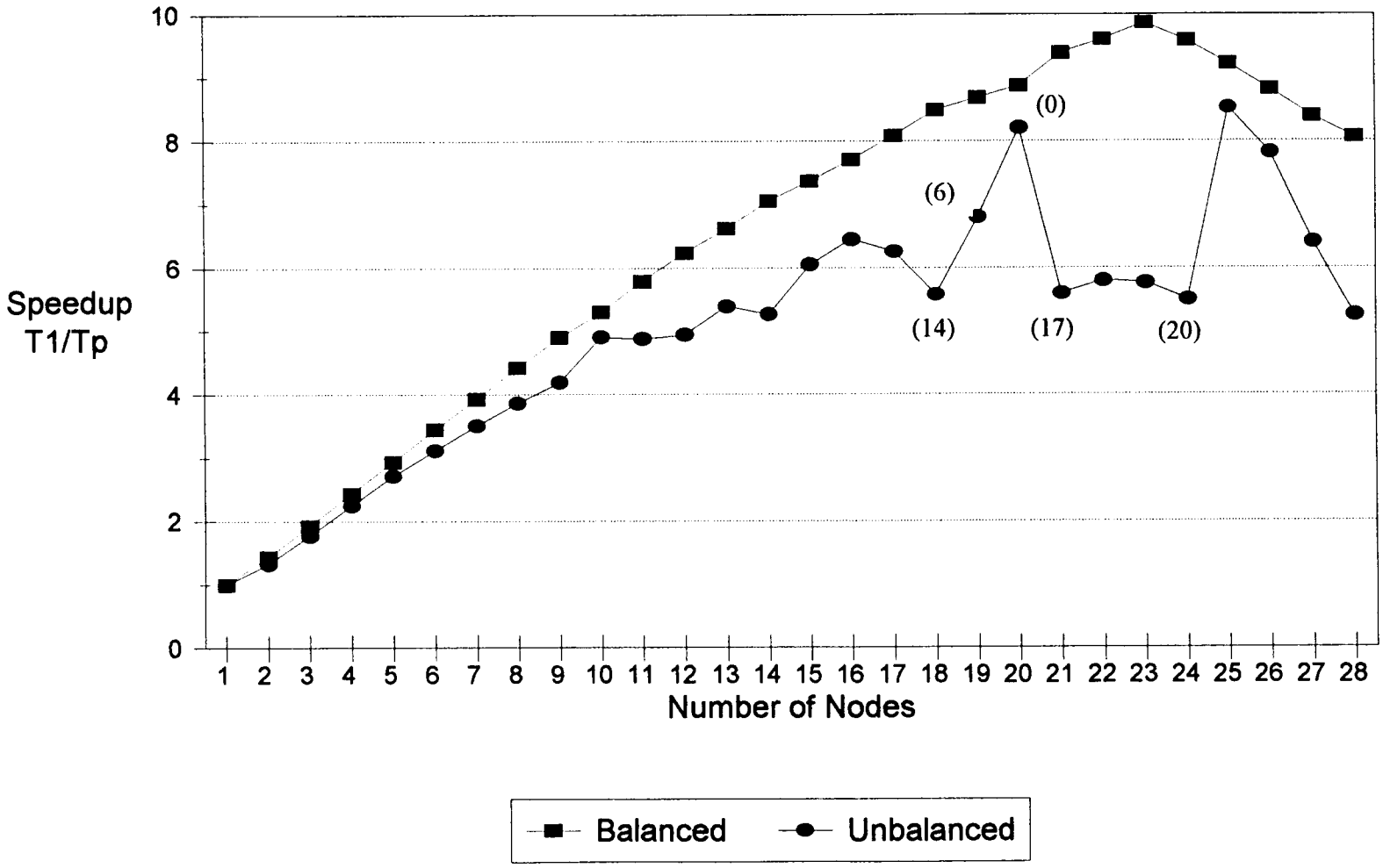


Figure 7: Efficiency
Computed Regression Values

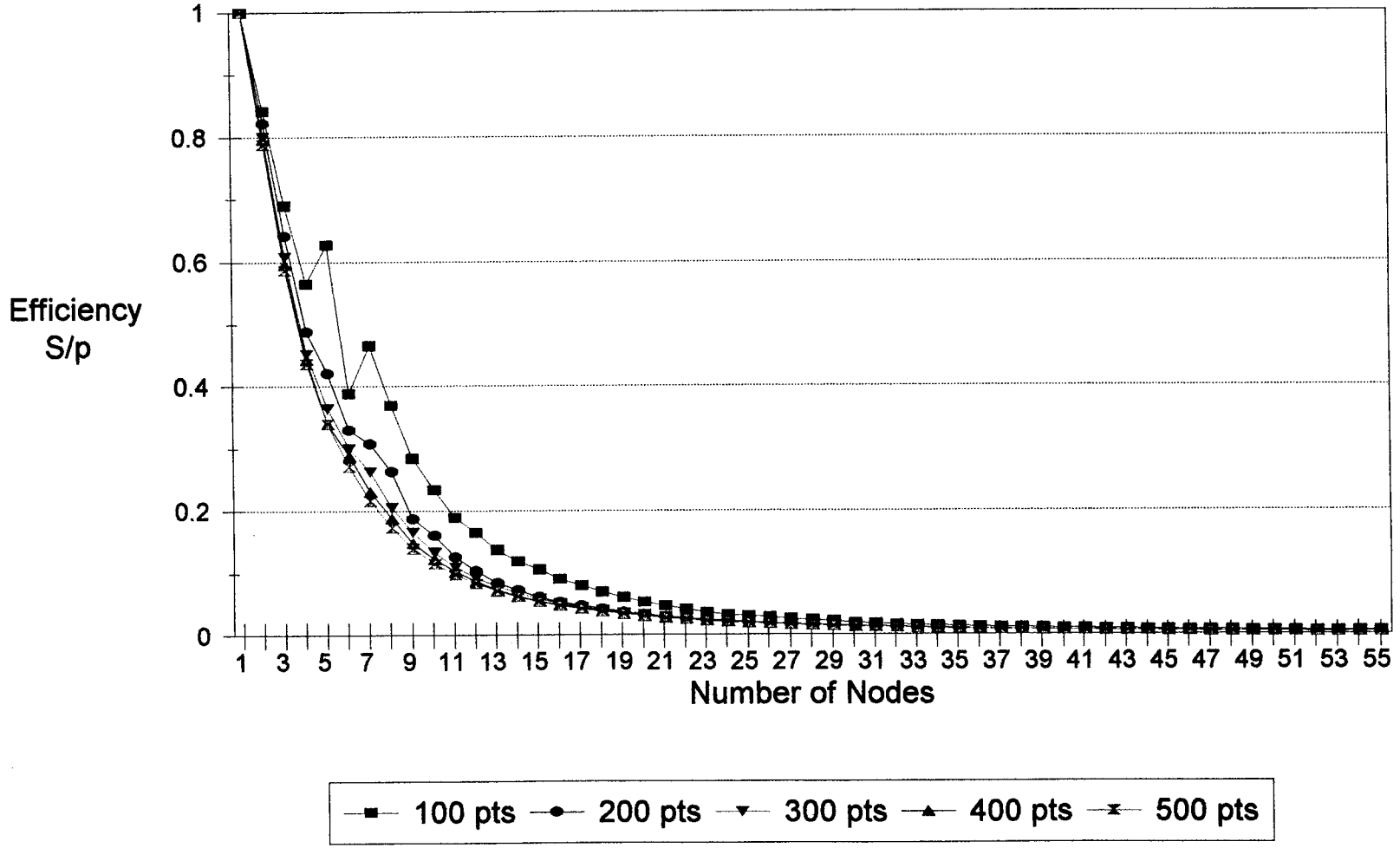


Figure 8: Efficiency
Degrees of Freedom

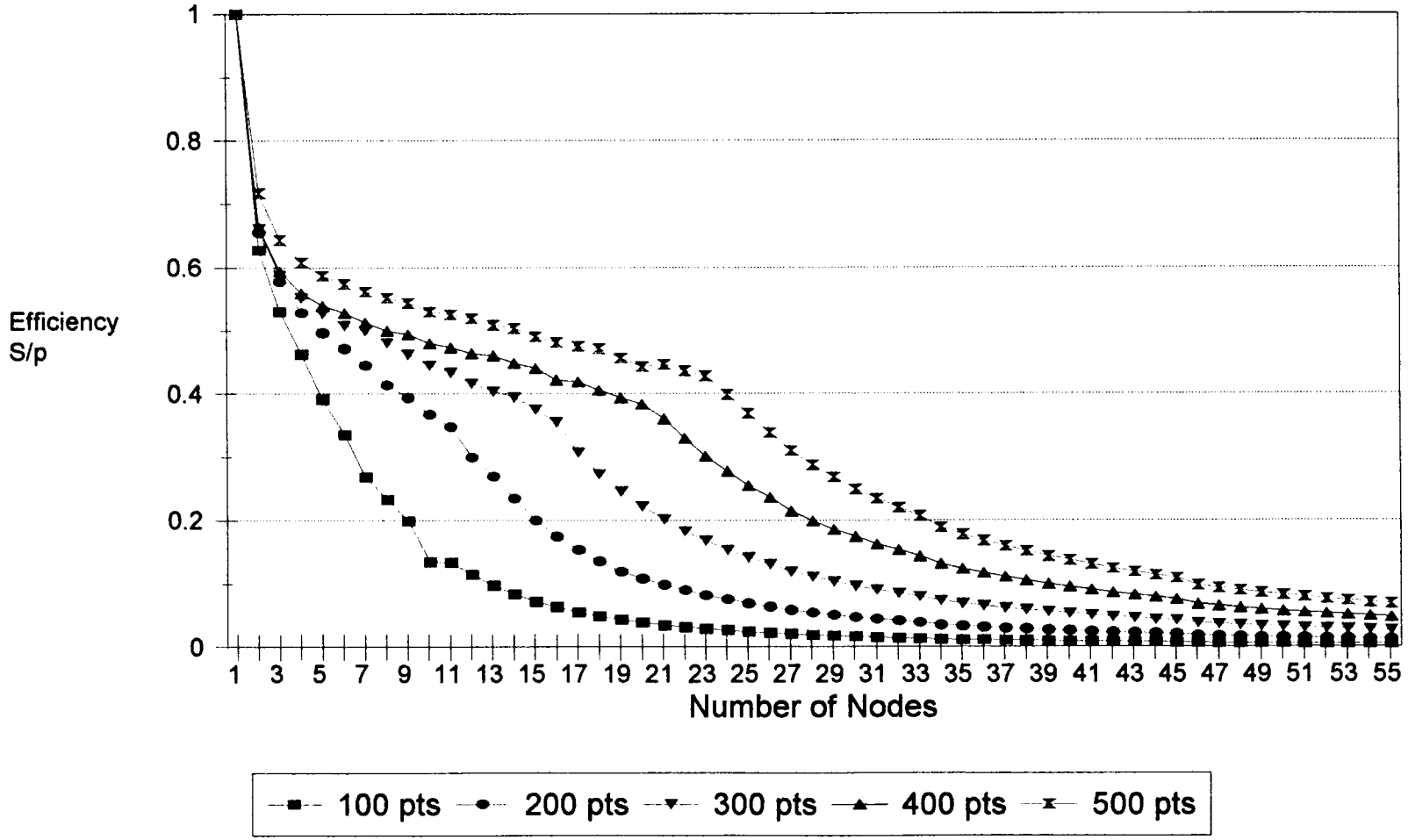


Figure 9: Speedup vs Efficiency

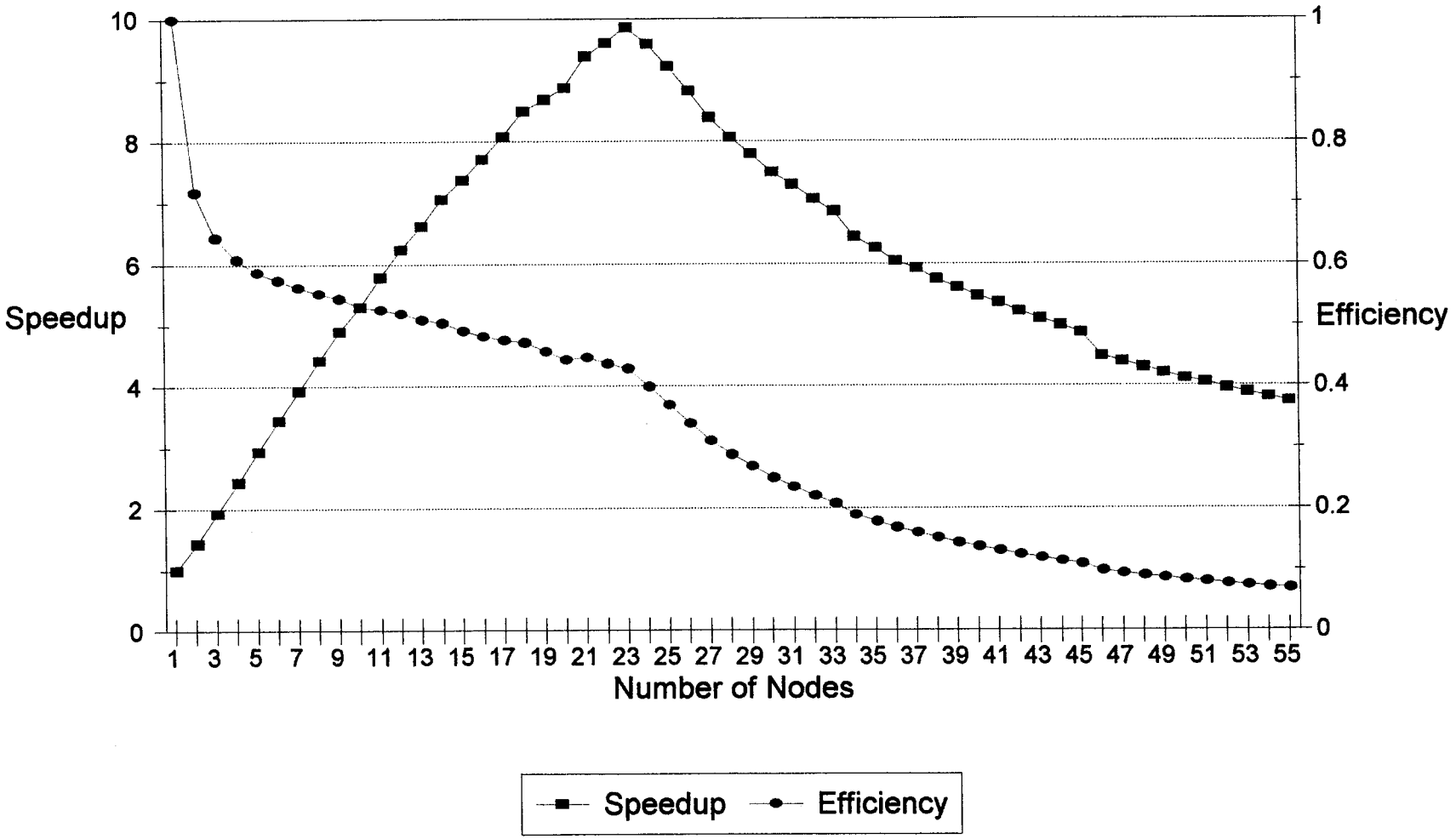
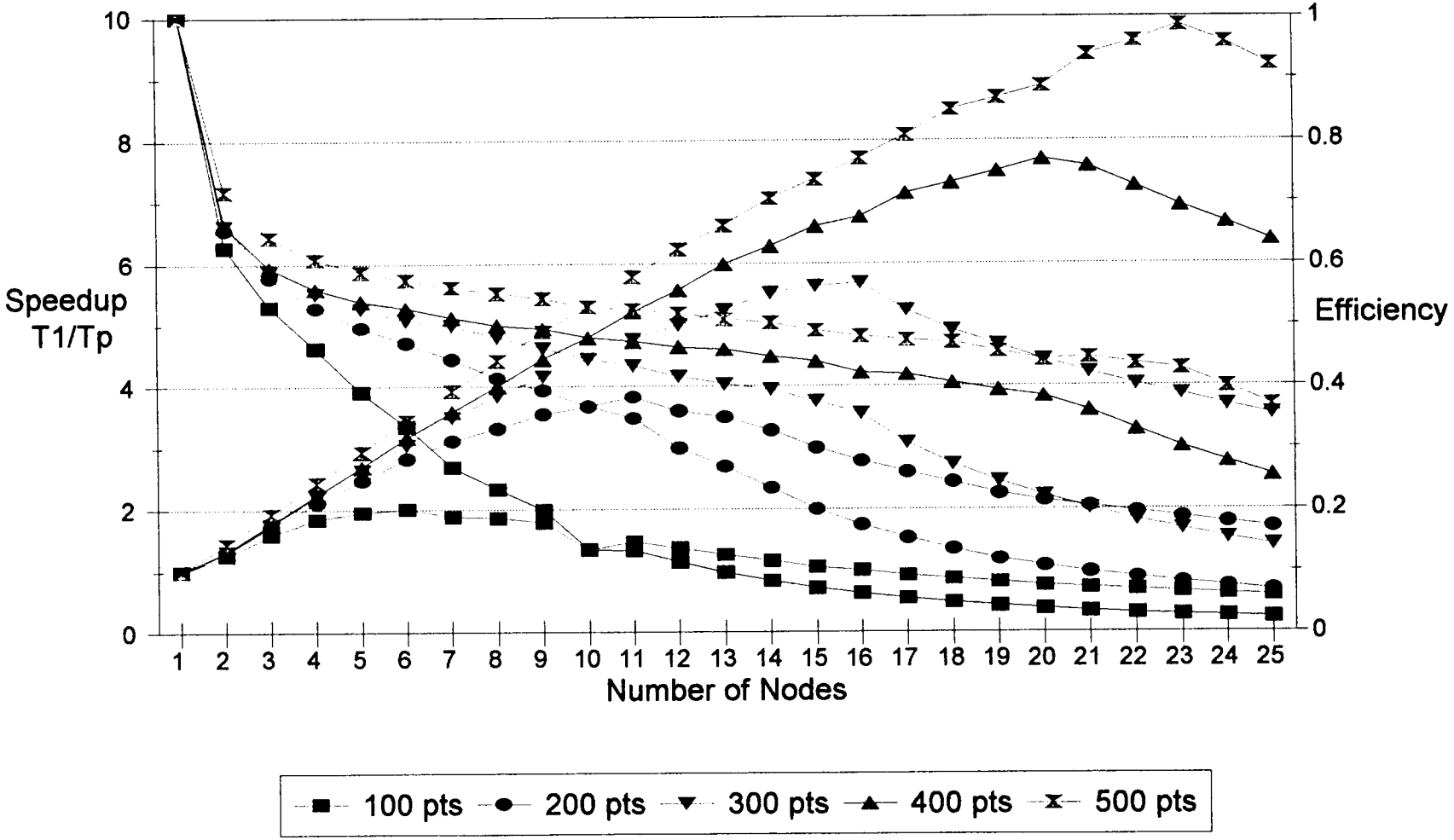


Figure 10: Efficiency vs Speedup



computations across the nodes can effectively reduce the amount of computation time. In addition, it demonstrated the impact of load balancing on the speedup and efficiency of an application. A maximum speedup value occurs at a particular number of nodes used. Futhermore, an increase in the number of nodes may not always improve the performance of the application. While the samples sizes used in the study were relatively small, the increase in speedup over the selected sample sizes seems to indicate that for sample sizes of 1000 or 10000 there will still be a significant increase in the speedup of the computations.

References

[1] Fox, G. , et. al. , *Solving Problems on Concurrent Processors, Vol 1. General Techniques and Regular Problems*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[2] Johnson, Richard A. and Wichern, Dean W.,*Applied Multivariate Statistical Analysis,* Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] Scott, David W., *Multivariate Density Estimation: Theory, Practice, and Visualization*, John Wiley & Sons, Inc., New York, 1992.

[4] Private Communications with Eric Harder, May 1994.

[5] Hastie, T.J. and R.J. Tibshirani, *Generalized Additive Models*, Chapman and Hall, New York, 1990.

[C] Cleveland, William S., Eric Grosse, and William M. Shyu, "Local Regression Models", *Statistical Models in S*, Chambers, John M. and Trevor J. Hastie eds.Wadsworth & Brooks/Cole, Pacific Grove, California, 1992, pp 309-376.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October, 1994	Technical

4. TITLE AND SUBTITLE

Parallelizing Locally-Weighted Regression

5. FUNDING NUMBERS

DAAL03-91-G-0039

DAAH04-94-G-0267

6. AUTHOR(S)

Julia Corbin Fauntleroy and Edward J. Wegman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Center for Computational Statistics
MS 4A7
George Mason University
Fairfax, VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER

TR No. 102

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper focuses on a nonparametric regression technique known as locally-weighted regression or LOESS. LOESS is a computationally intensive technique which makes it naturally amenable to exploitinghigh performance computers. In this paper, we explore domain decomposition techniquesfor LOESS and study the performance of our algorithm on an Intel Paragon XP/S A4 machine. We study both speedup and efficiency as a function of the number of nodes. Certain segmants of the LOESS computation are shown to be fruitfully parallelized while others are essentially sequential and cannot be parallelized effectively.

14. SUBJECT TERMS

LOESS, nonparametric regression, speedup, efficiency, load balancing.

15. NUMBER OF PAGES

18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October, 1994	Technical

4. TITLE AND SUBTITLE

Parallelizing Locally-Weighted Regression

5. FUNDING NUMBERS

N00014-92-J-1303

N00014-93-I-0527

6. AUTHOR(S)

Julia Corbin Fauntleroy and Edward J. Wegman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Center for Computational Statistics
MS 4A7
George Mason University
Fairfax, VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER

TR No. 102

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy
Office of the Chief of Naval Research
Mathematical Sciences Division
800 N. Quincy Street Code 1111SP
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Navy position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper focuses on a nonparametric regression technique known as locally-weighted regression or LOESS. LOESS is a computationally intensive technique which makes it naturally amenable to exploiting high performance computers. In this paper, we explore domain decomposition techniques for LOESS and study the performance of our algorithm on an Intel Paragon XP/S A4 machine. We study both speedup and efficiency as a function of thenumber of nodes. Certain segments of the LOESS computation are shown to be fruitfully parallelized while others are essentially sequential and cannot be parallelized effectively.

14. SUBJECT TERMS

LOESS, nonparametric regression, speedup, efficiency, load balancing

15. NUMBER OF PAGES

18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL