Proceedings of International Conference on Distributed
Systems, Software Engineering and Database Systems

Efficient Eigenvalue Computation of Symmetric Tridiagonal Matrices on Heterogeneous Workstation Clusters*

Shen-Fu Hsiao and Kuo-Chung Chen

Institute of Computer and Information Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
sfhsiao@cie.nsysu.edu.tw

Abstract

Many computation-intensive problems require costly platforms such as supercomputers or massively parallel computers (MPC) in order to achieve reasonable execution speed. Recently, due to the popularity of workstation and computer network, workstation clusters emerge as a low-cost alternative of parallel and distributed computing for applications requiring large amount of computations. This paper utilizes heterogeneous workstation clusters composed of various types of machines to calculate the eigenvalues of symmetric tridiagonal matrices, a problem frequently encountered in many scientific and engineering applications. Among the parallel algorithms for eigenvalue problems, the split-and-merge method with Laguerre's iteration is selected due to its inherent high parallelism, low communication overhead and suitability for distributed implementation. The computation platform consists of clusters of non-dedicated workstations connected by Ethernet with PVM (Parallel Virtual Machine) as the supporting software package. Various heterogeneous configurations are tested and compared to find a best machine combination given a fixed overall normalized computing power. To improve the computation efficiency, both static and dynamic load balancing is considered especially when the multi-user environment has highly uneven and time-changing load. The experimental data shows promising results of significant speed-up compared to the sequential implementations.

Key words: *PVM, split-and-merge algorithm, parallel computing, eigenvalue problem, workstation clusters, load balancing*

1 Introduction

Many numerical computation problems in scientific and engineering applications requires huge amount of operations which usually take too much time to execute on a single-CPU machine. The major solution for these computation-intensive problems is through the use of costly supercomputers or massively parallel computers (MPC) to reduce the total computation time. There have been a lot of papers focusing on implementing various parallel algorithms using these dedicated multiprocessor computers.

Recently, through the fast advance of VLSI and computer network technology, the speed performance of each single-CPU computer and the capability of the network that connect these computers improve dramatically. These trends make it feasible to develop applications using physically distributed resources as if they were part of the same computer. Thus, distributed scientific computing using workstation clusters begins to attract attention due to the popularity and availability of high performance workstations and local area network [2][4][11][12]. Furthermore, some network supporting software package such as PVM (Parallel Virtual Machine) [1] provides utility to assist users easily writing distributed-computing programs running on workstation clusters. In these parallel computation problems, the data communication among workstations is through the general-purpose computer network such as Ethernet, FDDI or ATM, contrary to that in MPC where the communication goes through the dedicated communication channels with specific architecture [8]. Currently, the data communication speed in MPC is much faster than that in workstation cluster while the computing power of each composing processor in MPC is in general slower than that of a single workstation especially when the workstation is becoming more powerful (such as the advance from microSPARC, superSPARC to ultrSPARC in Sun workstation series).

Thanks to the difference of the communication latency, some application problems parallelizable on MPC may not have comparable performance when run on workstation clusters [5]. Another difference between MPC and workstation clusters is that every node in a *heterogeneous* workstation cluster might have different computing speed due to different machine types. Even in homogeneous environment where each composing node is of the same machine type, every node may still have different computing power since the multi-user environment allows other users to share CPU at the same time when our parallel pro-

*This work was supported by National Science Council, Republic of China, under Grant NSC85-2213-E-110-016

grams are running. The non-controllability of such environment requires load balancing techniques to exploit more efficiently the computing resources of workstations. Hence, the good parallel algorithms in MPC may not have satisfactory performance when running on workstation clusters without taking into consideration these communication and computation differences.

In this paper, workstation clusters are used to solve eigenvalue problems of large symmetric tridiagonal (ST) matrices, an important issue in many scientific and engineering problems. A symmetric matrix can be reduced to an ST matrix by applying from both sides a sequence of Householder reflection matrices to zero out all the off-diagonal entries but the first super- and sub-diagonal entries. After this preprocessing step, the eigenvalues of the ST matrix may be solved using many well-known methods including parallel QR algorithm, Jacobi method, divide-and-conquer method, and bisection method [3][7]. The QR algorithm is well suited for sequential implementation but is hard to parallelize. The Jacobi method is inherently parallel and is suitable to be implemented on parallel computers consisting of hundred of nodes. However, in normal workstation clusters the number of computing nodes is rarely larger than 50, and the Jacobi method may cause heavy communication overhead in this environment. All the above methods have been implemented on various parallel computers such as CM-2, Cray YMP or nCUBE by exploiting the special features of the target machine.

Recently, a parallel algorithm called split-and-merge algorithm based on Laguerre's method to compute the eigenvalues of ST matrices of size 2^n was implemented on homogeneous workstation clusters with promising results compared to those obtained in nCUBE-2 parallel machine [10]. In this paper, we extend the split-and-merge method to compute the eigenvalues of ST matrices of *arbitrary size* on several *heterogeneous* workstation clusters in order to find the optimal workstation configuration for this ST eigenvalue problem. Special care has been exerted in order to reduce the communication overhead and thus increase the overall speed-up [9]. Furthermore, both static and dynamic load balancing is considered when the composing machines have time-changing load. A load emulation program is developed to simulate more realistically the load change condition in our noncontrollable multi-user environment.

2 Split-and-Merge Algorithm

Before describing the split-and-merge algorithm, we first give two lemma:

Lemma 1 (Separation Property [3]) *Assume that an $n \times n$ ST matrix A*

$$A = [\beta_{i-1}\ \alpha_i\ \beta_i]$$

$$= \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \cdots \\ 0 & \cdots & 0 & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ 0 & & \cdots & 0 & \beta_{n-1} & \alpha_n \end{bmatrix}$$

has eigenvalues $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$. Another ST matrix $\hat{A}$ is generated from A by replacing two off-diagonal entries with zeros ($\beta_k = 0$), i.e.,

$$\hat{A} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

where the $k \times k$ ST submatrix A_1 and $(n-k) \times (n-k)$ ST submatrix A_2 are

$$A_1 = \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ 0 & & \beta_{k-1} & \alpha_k \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \alpha_{k+1} & \beta_{k+1} & & 0 \\ \beta_{k+1} & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ 0 & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

Let $\hat{\lambda}_1 \le \hat{\lambda}_2 \le \cdots \le \hat{\lambda}_n$ be the eigenvalues of $\hat{A}$. The separation property states that

$$\hat{\lambda}_1 \in (\lambda_1, \lambda_2),\ \ \hat{\lambda}_n \in (\lambda_{n-1}, \lambda_n),$$
$$\hat{\lambda}_i \in (\lambda_{i-1}, \lambda_{i+1}),\ \ 2 \le i \le n-1.$$

Assuming that the eigenvalues of the two neighboring split submatrices A_1 and A_2 have been found and sorted, then these eigenvalues $\hat{\lambda}_i$ of $\hat{A}$ can be used as the initial approximation to find the actual eigenvalues λ_i using the Laguerre iteration to be stated below.

Lemma 2 (Laguerre's Iteration [6])
Given an initial eigenvalue approximation x where $\lambda_i \le x \le \lambda_{i+1}$ and the characteristic polynomial $f(\lambda)$ of the $n \times n$ ST matrix A, the following Laguerre's iteration defines two values $L_+(x)$ and $L_-(x)$

$$L_\pm(x) = x + \frac{n}{[-\frac{f'(x)}{f(x)}] \pm \sqrt{(n-1)[(n-1)[\frac{f'(x)}{f(x)}]^2 - n\frac{f''(x)}{f(x)}]}}$$

with

$$\lambda_i \le L_-(x) \le x \le L_+(x) \le \lambda_{i+1}.$$

The recursive application of the Laguerre iteration with initial value $x \in (\lambda_i, \lambda_{i+1})$ will approach the real eigenvalues λ_i or λ_{i+1}. Let

$$x_+^{(k)} = L_+^k(x) \equiv \overbrace{L_+(L_+(\cdots L_+}^{k}(x)\cdots)),$$

$$x_-^{(k)} = L_-^k(x) \equiv \overbrace{L_-(L_-(\cdots L_-}^{k}(x)\cdots)),$$

then

$$\lambda_i \leftarrow \cdots x_-^{(2)} < x_-^{(1)} < x < x_+^{(1)} < x_+^{(2)} \ldots \rightarrow \lambda_{i+1}$$

In other words, the two sequences, $x_-^{(k)}$ and $x_+^{(k)}$, converge respectively to λ_i and λ_{i+1}.

For simplicity, we assume that the initial ST matrix A is size of $2^r \times 2^r$. The split-and-merge algorithm is illustrated in Fig. 1. During the split process, the

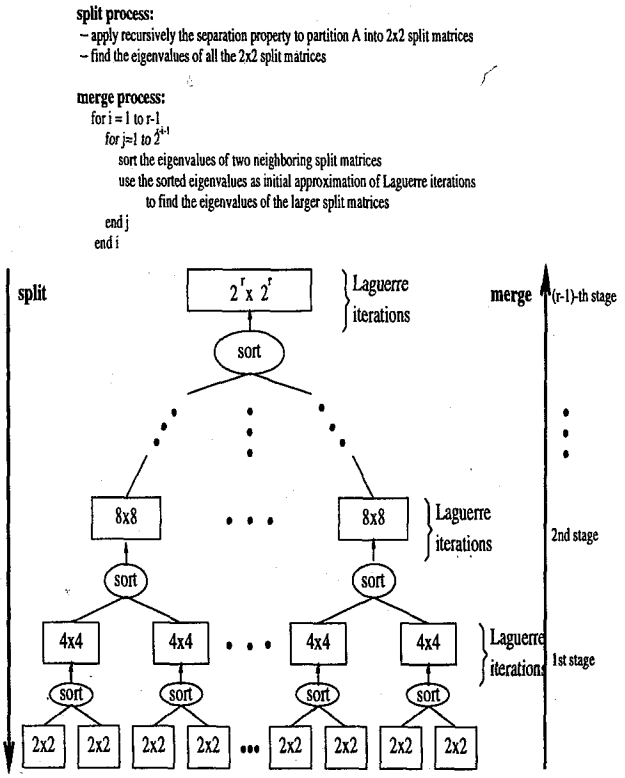


Figure 1: The split-and-merge algorithm of $2^r \times 2^r$ ST matrices.

$2^r \times 2^r$ ST matrix is partitioned into 2^{r-1} 2×2 split matrices by applying recursively the separation property of Lemma 1. After the split process, the eigenvalues of all the 2×2 split matrices are solved. During the merge process, the eigenvalues of each pair of neighboring $2^k \times 2^k (k = 1, 2, \cdots, r-1)$ split matrices are merged and sorted. The sorted 2^{k+1} eigenvalues are then used as the initial approximation in Laguerre iterations (Lemma 2) to find the eigenvalues of the larger split matrices of size $2^{k+1} \times 2^{k+1}$. Repeating the above procedure with $k = 1, 2, \cdots, r-1$, all the eigenvalue of the original ST matrix A are found after $r-1$ stages.

3 Distributed Implementations

The sequential split-and-merge algorithm is implemented on workstation clusters using PVM software which provides a unified framework to develop parallel programs using a collection of heterogeneous computer systems viewed as a single parallel virtual machine. The adopted PVM computing model is the *master-and-slave model* where the master is in charge of the initial task distribution and the final result collection while the slaves on different machines executes the main operations (Laguerre iterations, sorting and data redistribution).

Fig. 2 is a parallel version of the split-and-merge algorithm for the case of four-machine cluster. At the

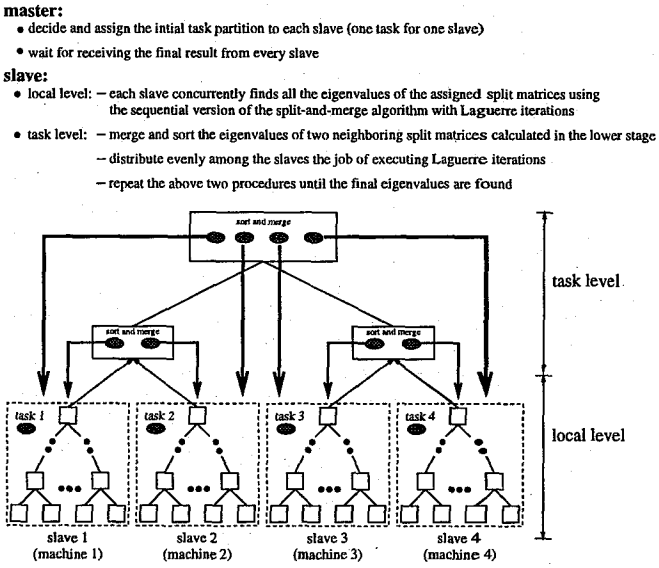


Figure 2: Distributed implementation of the split-and-merge algorithm on four machines (slaves), each assigned a task.

local level, each slave (task) calculates the eigenvalues of the assigned split matrices using the *sequential* version of the split-and-merge algorithm. At the task level which allows parallel computation, the operations of the Laguerre iterations are distributed evenly among slaves according to the number of available slaves, as shown by the bold-faced arrows.

When the number of slaves is not power of 2, special caution should be exercised in order to increase the utilization efficiency of all the slave machines. Fig. 3 shows the job partition and distribution for 6 slaves. Task1234 merged from tasks 1, 2, 3 and 4 is redistributed among 6 slaves instead of 4 to avoid the idle of the slave 5, and 6. Similarly, when the number of slaves is 5, task12 (merged from task1 and task2) and task34 (merged from task 3 and 4) is distributed among the 5 slave machines. Despite of the effort, as will seen shortly, the parallelization efficiency is still low when the number of slave machines is 5 or 9 since the communication overhead during the data redistribution is large in these cases.

3.1 Homogeneous Environment

The homogeneous environment consists of 12 SPARC2 workstations connected by Ethernet. Table 1 lists each portion of the total eigenvalue com-

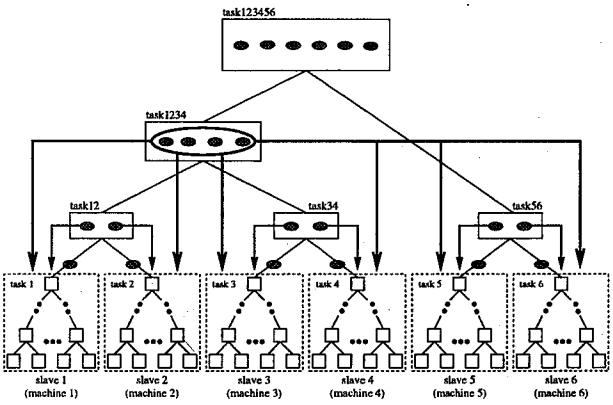


Figure 3: Distributed implementation of the split-and-merge algorithm on six machines with schemes to avoid machine idling.

putation time for ST matrix of size 2000 × 2000 on five workstations. T_{slave} is the total execution of each slave; T_{idle} is the idle time of each slave due to data waiting; T_{sort} is the time spent on sorting required before applying Laguerre iterations; T_{pvm} is the overhead to execute the PVM codes compared to the sequential implementation. The total execution time $T_{total} = T_{slave}^{max} + T_{recv}$ where T_{slave}^{max} is the largest execution time of all the slaves and T_{recv} is the data collection time of the master receiving the final results from all the slaves.

In balanced load situation T_{slave}^{max} is about the same as the average of T_{slave}. However, under unbalanced load, the difference between the largest and smallest T_{slave} could be significant and the load balancing method to be discussed in Sec. 4 will improve the execution time. We observe that all the overhead time portions (T_{idle} and T_{pvm}), including the communication overhead, are negligible compared to T_{slave}.

$T_{sequential} = 617.89s$, $T_{recv} = 3.13s$

slaves	T_{slave}	T_{idle}	T_{sort}	T_{pvm}
0	146.86	2.68	0.6	0.04
1	148.68	2.34	0	0.06
2	149.20	2.00	0.07	0.05
3	149.73	3.73	0	0.05
4	149.81	1.06	0	0.06

$T_{total} = T_{slave}^{max} + T_{recv} = 152.94s$ unit: seconds

Table 1: Each portion of the slave execution time on five workstations for 2000 × 2000 ST matrices.

Fig. 4 is the speed-up curves for ST matrix size up to 8000 × 8000 on 12 workstations where the speed-up is defined as the ratio of $T_{sequential}/T_{total}$. We have the following interesting observations:

- the speed-up basically increases proportionally to the number of workstations.
- the speed-up for larger matrices is in general better than for smaller ones because the communication overhead of larger matrices occupies a smaller percentage of the total execution time.
- the parallel efficiency defined as the speed-up divided by the number of workstations decreases as the number of workstations increases due to the increased communication overhead.
- the better speed performance when number of workstations is 2, 4 or 8 shows a fact that the split-and-merge algorithm is best suited for workstation number of power of 2 due to the inherent binary tree data structure.

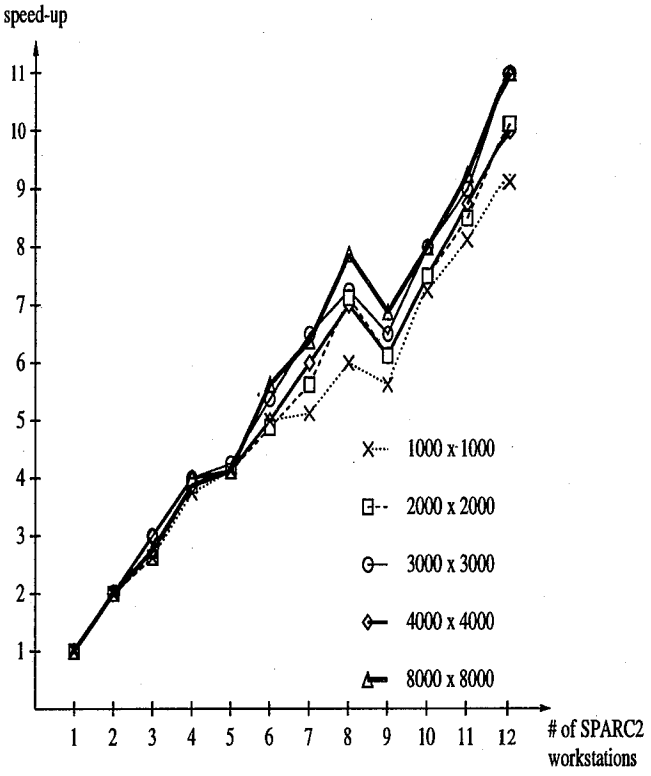


Figure 4: Speed-up vs. number of SPARC2 workstations for various matrix size.

3.2 Heterogeneous Environment

The heterogeneous platform consists of various types of workstations including Sun SPARC2, SPARC10, SPARC20/51, SPARC20/61 and HP715/33. In order to measure more accurately the relative computing power of the above machines, we run the sequential split-and-merge programs for various ST matrix size on different workstations. In general, the relative computing power may depend on the problem size and thus we need to create a table of relative computing speed of various types of machines for different matrix size under experiment [5]. However, in the eigenvalue computation of ST matrices, we observe an interesting fact that the ratio of the total computation time and the square of the matrix size is almost a constant

independent of the matrix size, as shown in Fig. 5. The relative speed ratio (normalized with the com-

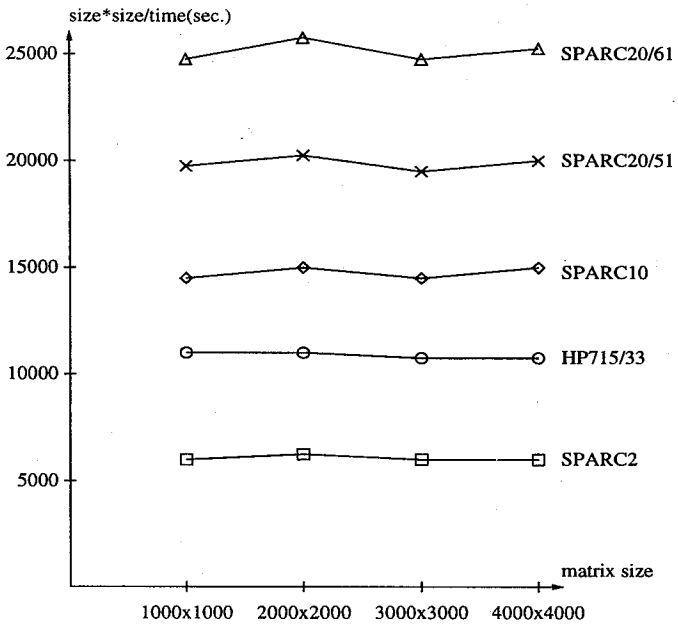


Figure 5: The ratio of the square of matrix size and the the total sequential execution time (in unit of seconds) vs. the matrix size.

puting power of SPARC2 as 1) is listed in Table 2. The relative computing power is used to determine

machine types	relative speed
SPARC2	1.00
HP715/33	1.70
SPARC10	2.20
SPARC20/51	3.08
SPARC20/61	3.91

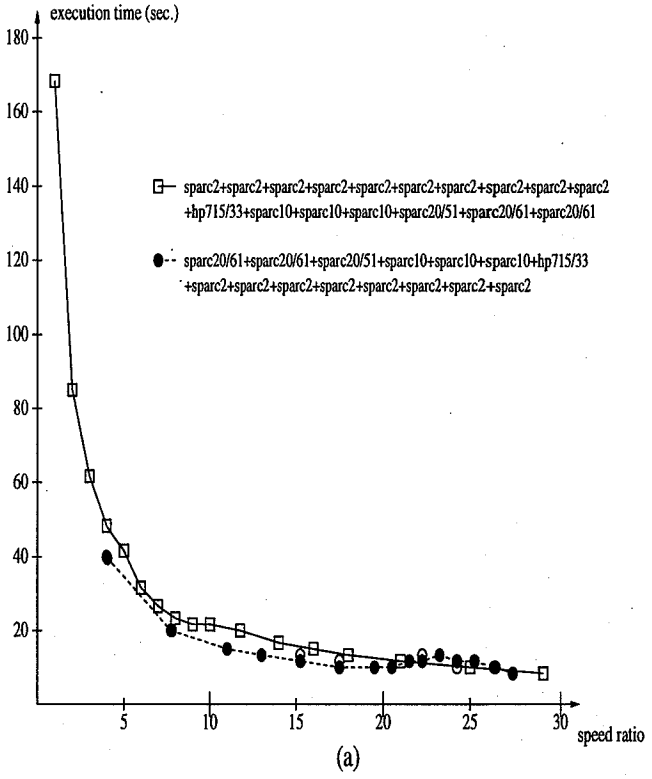
Table 2: Normalized computing power of various types of workstations.

the job partition size on different types of machines in order to fully exploit the computing power of each node in the heterogeneous environment.

In the following, we will consider the speed performance of several different configurations by gradually adding more machines into the whole heterogeneous system. The first (second) configuration series is formed by adding machines of increasing (decreasing) computing power. Fig. 6 and 7 show respectively the execution time for matrix size of 1000×1000 and 4000×4000 on heterogeneous configuration series I and II where the horizontal axis denotes the accumulated normalized computing power of the heterogeneous configuration series with up to 17 machines of various types.

We have the following observation:

- For the same normalized system computing speed power, the speed performance with fewer ma-



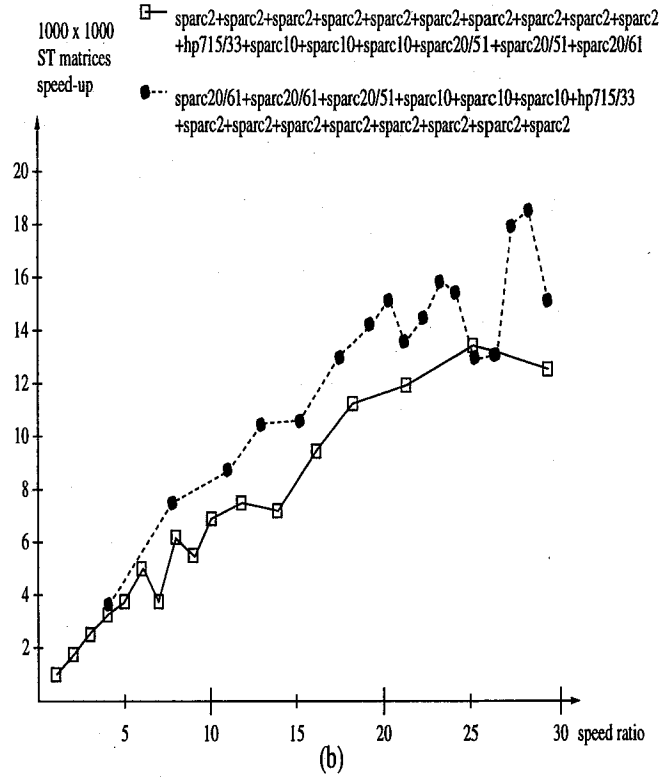
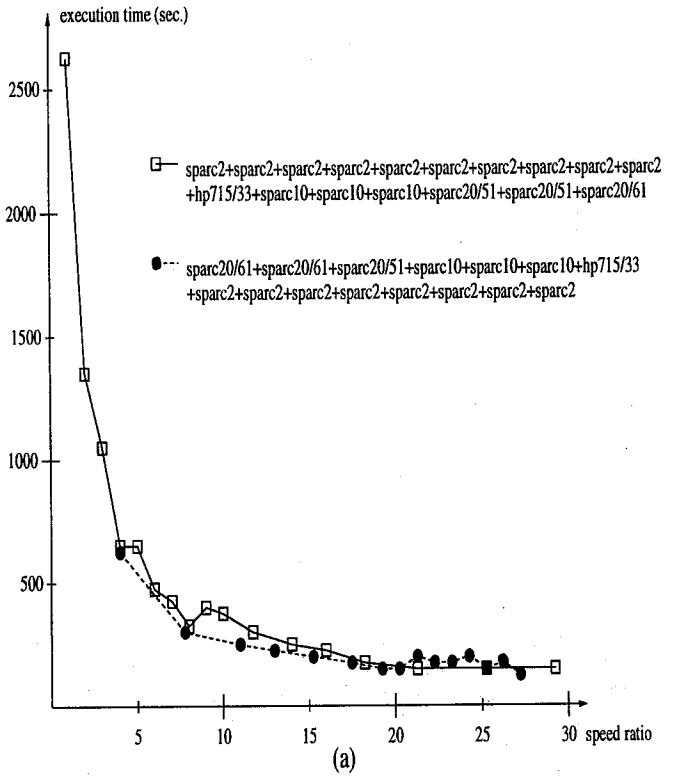


Figure 6: Execution time (a) and spee-up (b) vs. total speed ratio for 1000×1000 matrix on two different heterogeneous configuration series.



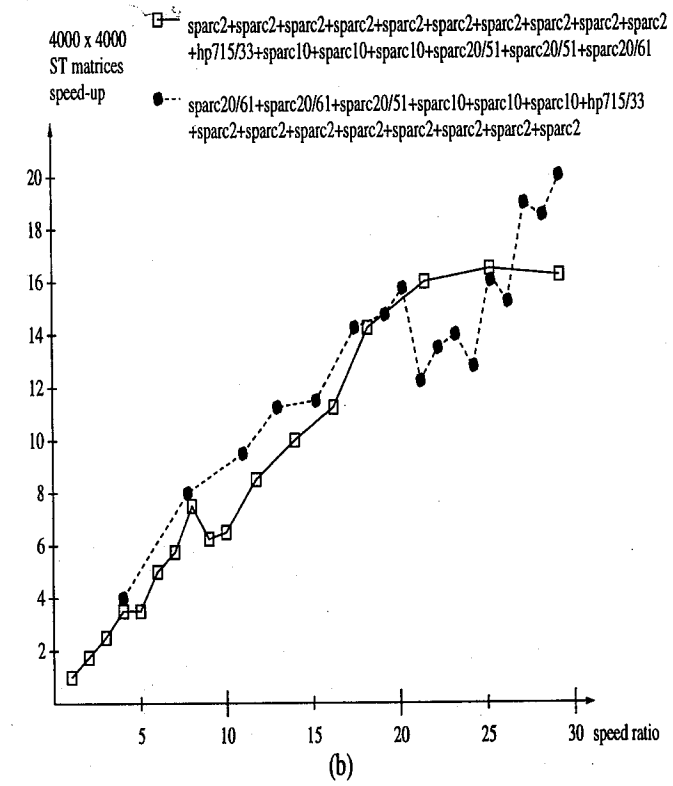


Figure 7: Execution time (a) and speed-up (b) vs. total speed ratio for 4000×4000 matrix on two different heterogeneous configuration series.

chines is better because the communication overhead is roughly proportional to the number of workstations. Hence, the first configuration (adding machines with decreasing computing power) has better performance than the second one (adding machines with increasing power). This observation tells us that we should configure the heterogeneous system by adding faster machines first.

- As the overall system computing speed achieves about 30 times the power of SPARC2, the system performance can no longer have effective speed-up increase. The reason for such speed-up saturation phenomenon is that when the total execution time drops to a quantity comparable with the communication time, the further increase of system computing power (and thus further decrease of the calculation time) is not significant compared to the communication overhead, i.e., the decrease of the calculation time is offset by the increase of the communication time. We expect that the saturation tendency of the speed-up curve will lag for larger matrices. In other words, larger matrices will have better speed-up curve. One way of improving the speed-up efficiency is to use LAN network of higher speed (such as ATM network) to reduce the communication overhead.

4 Dynamic Load Balancing

Since network workstations support multi-user multi-process environment, the machine load may change dynamically in accordance with the number of users and processes increases. In case of severe uneven load situation, the difference between the maximum and minimum T_{slave} will be large and thus the overall speed performance will degrade significantly. The difference of relative computing power in the heterogeneous environment has been taken into consideration in Sec. 3 to implement more efficient the split-and-merge algorithm. The task partition based on the initial computing power in Table 2 is called *static load balancing* which tries to distribute the computation job evenly among the various types of workstations.

However, the initial relative power does not necessarily reflect the *real* computing power due to time-changing load discussed above. In this section, we propose a *dynamic load balancing* method which determines the job partition by estimating the actual load situation in each slave machine. The whole computation is divided into several sequential stages depending on the matrix size. The job partition in each stage is based on the estimation of the relative computing speed of all the machines in the previous stages instead of the original CPU computing power. Fig. 8 is the job partition diagram where the master is numbered 0 and the slaves are numbered from 1 to 8. The horizontal width of each slave at the task level represents the size of the job partition at that stage. Wider slave means higher relative computing power (i.e., lighter load or more powerful CPU) and thus can be assigned larger job partition.

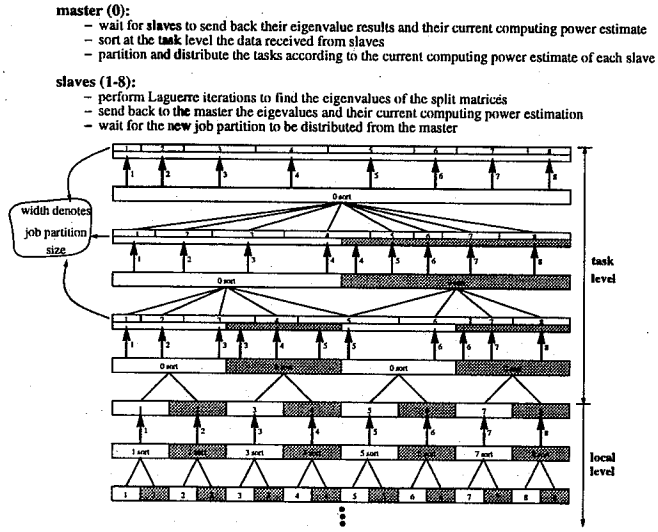


Figure 8: Dynamic load balancing method with master (0) estimating the current computing power of slaves (1-8) and determining the corresponding job partition.

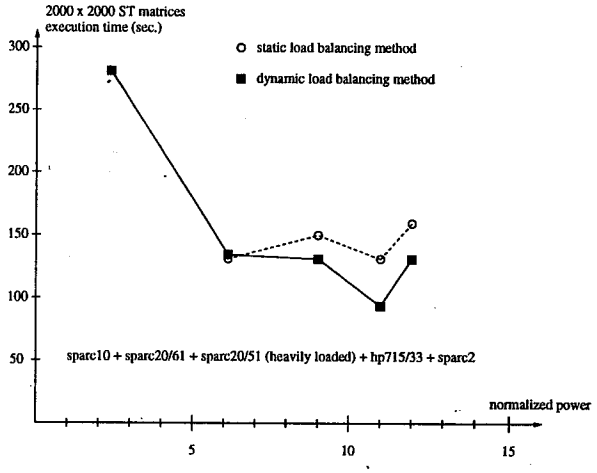


Figure 9: Execution time vs. normalized computing power on a heterogeneous configuration series composed of SPARC10, SPARC20/61, SPARC20/51, HP715/33 and SPARC2 where the load of the SPARC20/51 is aggravated by adding a never-stop pseudo code.

Fig. 9 shows the execution time under normal load situation and uneven load situation for 2000×2000 matrix running on a heterogeneous system composed of a SPARC10, a SPARC20/61, a SPARC20/51, a HP715/33 and a SPARC2. The load of the SPARC20/51 is aggravated by adding on purpose a never-stop pseudo-code into the machine to change its initial computing power. Under such unbalanced load situation, the implementation with the dynamic load balancing method always have better speed performance after the SPARC20/51 is added into the heterogeneous configuration series.

In order to simulate more practically the time-changing load, we write a PVM load-emulation program which randomly generates and distributes pseudo codes among the composing machines. These pseudo codes will stop after a prescribed time limit and the emulation program always keep the total number of the pseudo codes a constant which depends on the number of machines in the configuration. Fig. 10 shows the the performance of the dynamic load balancing method under such dynamic load situation. We observe that employing the load balancing method always gives better speed performance.

5 Conclusion

The eigenvalue problem of symmetric tridiagonal matrices is solved using the split-and-merge algorithm with Laguerre iterations, and is implemented on both homogeneous and heterogeneous workstation clusters for distributed computing. Some interesting observations are made from the experimental data. We tested several heterogeneous configuration series by adding one by one different types of machines and running the PVM programs for each configuration. With the same overall normalized computing power, the configuration with fewer machines has better speed per-

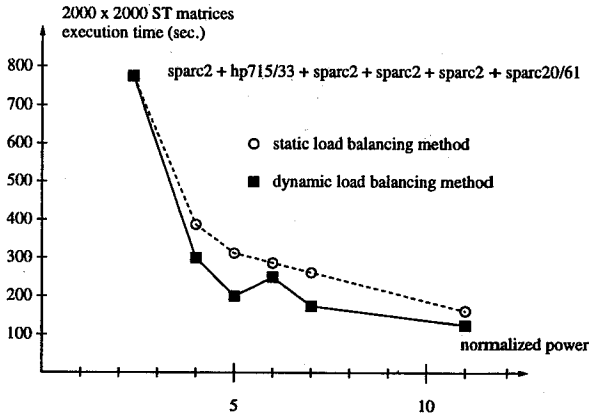


Figure 10: Execution time vs. normalized computing power on a heterogeneous configuration series composed of SPARC10, HP715/33, SPARC2, and SPARC20/61 where the load of each composing machine changes with time by a load-emulation program.

formance due to the lower communication overhead. A dynamic load balancing method is also adopted in order to obtain better speed performance under the non-controllable network-connected workstations where the load of each machine may change rapidly with time due to the multi-user and multi-process environment.

References

[1] A. Geist, *et. al.*, "*PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*", The MIT Press, May 1994.

[2] C. Giertsen and J. Petersen, "*Parallel Volume Rendering on a Network of Workstations*", IEEE Computer Graphics and Applications, pp. 16-23, Nov. 1993.

[3] G. H. Golub and C.F. van Loan, "*Matrix Computations*", 2nd ed., The Johns Hopkins Univ. Press, 1989.

[4] C.-K. Lee and M. Hamdi, "*Parallel Image Processing Applications on a Network of Workstations*", Parallel Computing, pp. 137-160, 1995.

[5] C. N. Lee, T.-Y. Lee, S.-F. Hsiao and T.-C. Lu, "*Performance Evaluation for Parallel Computing on Network Environments*", Proc. 1996 Workshop on Distributed System Technologies and Applications, pp. 237-246, May 1996

[6] T. Y. Li and Z. Zeng, "*The Laguerre Iteration in Solving the Symmetric Tridiagonal Eigenproblem, Revisited*", SIAM J. Sci. Comput., pp. 1145-1173, Sept. 1994.

[7] B. N. Parlett, "*The Symmetric Eigenvalue Problem*", Prentice-Hall Pub., 1980.

[8] J.-T. Pfenning and C. Moll, *Optimized Communication Patterns on Workstation Clusters*, Parallel Computing, pp. 373-388, 1995.

[9] V. Strumpen and T. L. Casavant, "*Exploiting Communication Latency Hiding for Parallel Network Computing: Model and Analysis*", Proc. Intl. Conf. Parallel and Distributed Systems, Dec. 1994.

[10] C. Trefftz, *et. al.*, "*A Scalable Eigenvalue Solver for Symmetric Tridiagonal Matrices*", Parallel Computing, pp. 1213-1240, 1995.

[11] S. White, A. Alund and V. S. Sunderam, "*Performance of the NAS Parallel Benchmarks on PVM-Based Networks*", Journal of Parallel and Distributed Computing, pp. 61-71, 1995.

[12] K. Zielinski, M. Gajecki, and G. Czajkowski, *Parallel Programming Systems for LAN Distributed Computing*, Proc. Intl. Conf. Distributed Computing Systems, pp. 600-607, June 1994.