Tribology Transactions

ISSN: 1040-2004 (Print) 1547-397X (Online) Journal homepage: www.tandfonline.com/journals/utrb20

The Application of Nearly Embarrassingly Parallel Computation in the Optimization of Fluid-Film Lubrication©

NENZI WANG, YAU-ZEN CHANG & CHIH-MING TSAI

To cite this article: NENZI WANG, YAU-ZEN CHANG & CHIH-MING TSAI (2004) The Application of Nearly Embarrassingly Parallel Computation in the Optimization of Fluid-Film Lubrication©, Tribology Transactions, 47:1, 34-42, DOI: 10.1080/05698190490279056

To link to this article: https://doi.org/10.1080/05698190490279056

Tribology Transactions, 47: 34-42, 2004

ISSN: 0569-8197 print
DOI: 10.1080/05698190490279056

The Application of Nearly Embarrassingly Parallel Computation in the Optimization of Fluid-Film Lubrication©

NENZI WANG (Member, STLE), YAU-ZEN CHANG and CHIH-MING TSAI

Chang Gung University
Department of Mechanical Engineering
Tao-Yuan, Taiwan, R. O. China

The combination of powerful, yet inexpensive PCs and readily available open sources for parallel computation marks a new era of easy access to massive computation for the tribology community. The study demonstrates the applicability of embarrassingly parallel computation in the optimization of air-lubricated porous bearings with four design variables. To achieve high speedup without increasing the coding complexity, the master computer implements the lattice method to allocate the near-the-same computational load in the master-slave cluster. The effect of master capability on the cluster performance is also presented. The results are compared with that of an unparallelized simplex method and indicate a significant reduction in execution time due to parallelism. In a simulated analysis, a high speedup can also be obtained in dealing with a problem with many design variables. This study provides the framework for optimization of applications with complex tribological models to be solved with minimum execution time.

KEY WORDS

Optimization; Parallel Computation; Air Bearings

INTRODUCTION

The performance of inexpensive PCs is advancing at a fast pace. The trend to pack multiple processors in a system is also emerging to create parallel machines on desktop computers, as is the hyper-threading technique that allows a single processor package to appear as two logical processors to the operating system. The demand for computational power is not satisfied by the computing power of PCs, and is increasing even more due to the need for comprehensive calculation for reliable products and the computational-intensive tasks in the modeling of tribological systems in submicron or nanometer scale.

Presented at the STLE/ASME Tribology Conference in Ponte Verda Beach, Florida October 27-29, 2003
Final manuscript approved July 2, 2003
Review led by Bill Marscher

For a long while, mainframes and supercomputers have been built and operated by large companies or organizations. Lots of universities and small companies could not afford the cost of such a system. The concept of clustering was then born when one first tried to spread different tasks over several computers and then collect the result produced. As cheaper and more common hardware became available, the performance gap between a supercomputer and a cluster of multiple inexpensive PCs became small. Clusters can grow in size by adding more machines. The scalability is a merit multiprocessor computers do not possess. A cluster as a whole will be more powerful when the number of computers increases and faster individual processors and higher connection speeds are used.

In order to reduce the computational time in a lubrication analysis, planning and conducting a numerical computation in parallel is essential. In the analysis for porous air bearing optimization, Wang and Chang *(8)*, proposed a hybrid method that uses a genetic algorithm for the initial search and proceeds with the search for a refined solution with a simplex method to lower the number of function calculations in a single-processor system. As indicated in the end of the paper, a further reduction of execution time may require the employment of some sort of parallel computation. In 2002, Hanke and Talke *(2)* presented a new method for reduced computational effort in an air slider optimization problem. The proposed method subdivides the chosen design area of the slider to be optimized in a number of small subregions. In addition to the advantage of this subdividing for the slider optimization, the approach of subdividing allows the potential use of parallel computation.

Both the genetic algorithm and simulated annealing techniques are well known for the high probability of finding the global optimum of a complex optimization problem with many local extremes. The application of genetic algorithm (Kotera and Shima *(4)*; Kato and Soutome *(3)*; Wang and Chang *(8)*) and simulated annealing (O'Hara, et al. *(5)*) for optimization have been successfully implemented in many fields. As both methods usually required a large number of function evaluations in the fitness improvement process where many calculations can be conducted concurrently, this computational convenience makes them suitable for parallel computation.

There are many ways parallel computation can reduce the execution time, such as loop fusion and loop reduction conducted automatically by modern compilers. However, various parallel computational approaches vary dramatically in their complexity. In general, the arrangement of a parallel computer system can be one of the following: (1) shared memory multiprocessor system; (2) message-passing multicomputer system with physically distributed memory. From the viewpoint of a programmer, the best architecture to conduct a parallel computation is via shared-memory processing. This will ease the programming task as well as eliminate many overheads such as I/O-bound and dynamic load-balancing problems. The shortcoming of this type of approach is its expensive and non-scalable special hardware system in which the number of the processors is predefined and cannot be altered easily. A general message passing multicomputer is constructed by a cluster of computers connected by a network. The system may require special software to establish the communication between the computers. Nevertheless, the communication software is freely available in open source (Akl *(1)*; Wilkinson and Allen *(10)*).

In a complex tribological optimization analysis, two processes are involved in the computation: (1) the process of dealing with the CPU-intensive performance calculations such as a finite-difference or finite-element type methods for a thermal non-Newtonian elastohydrodynamic lubrication model; (2) the optimization process. Depending on the numerical scheme adopted, both parts can be handled in parallel. With proper parallelization of the optimization process while conducting the performance calculations in individual processors with efficient serial codes, which have been developed for a long period of time, one can parallelize the process and solve the problem efficiently.

The main focus of this study is to illustrate the applicability of embarrassingly parallel computation in the optimization analysis of fluid-film lubrication. The performance of the cluster constructed by inexpensive network-connected PCs running openly available software is also demonstrated. The characteristics of the communication network are important for the design of parallel machines, but are not essential for the implementation of the parallel algorithm presented in this study. Hence, the overhead due to communication time is treated as a part of general overhead in the proposed scheme. As will be seen in the latter part of the article, the overall overhead is small for the computational-intensive tasks with limited data exchange.

THE NUMERICAL MODEL

A square thrust air bearing with 4-porous-pad design is used in this study to demonstrate the parallel optimization procedure and the cluster performance. In the calculation for an objective that maximizes the load-capacity and stiffness of the bearing while minimizing the airflow, a converged solution between that of Darcy's law for porous material pressure distribution and the air film pressure described by the compressible fluid Reynolds equation is required. The four design variables are the size of the porous pad, the location of the porous pad, the pressure of the supplied air, and the feeding parameter of the porous material. The detail of the porous air bearing analysis can be found in Nang and Chang *(8)*.

The main selection criterion of the optimization methods is to illustrate the key issue in this study—an algorithm that requires a large number of operations to reach a solution could be more efficient than an algorithm that requires fewer operations, if the steps of the former can be executed concurrently. Two numerical optimization methods used in this study are the lattice method and the simplex method (Rao *(6)*; Seireg and Dodgriguez *(7)*; Wang, et al. *(9)*). The lattice method is a search-all-corner method and the simplex method is basically a reject-the-poorest-point scheme. The lattice method is characterized by being easily implementable but less efficient in multifactor optimization due to all-direction searches. For an n-variable problem, $2n$ function calculations are required in each search step. Nevertheless, this inefficiency in multivariable design can be relaxed by parallel computation.

On the other hand, the simplex method basically requires only two to three calculations in a search step for any number of variables. The search direction in the simplex method, which is the direction reflected from the worst point, is usually not the optimum direction. This relatively small amount of calculations not only leads to high efficiency in solving multifactor optimization problems for a single-processor system, but also leads to difficulty in parallelization for a multicomputer cluster. If parallelism is demanded, the task-dividing can only be conducted in the calculation of objective functions, which increases the coding complexity for the numerical models. If the computational loads of subtasks are small or uneven, the efficiency also significantly suffered from the overhead of communication.

EMBARRASSINGLY PARALLEL COMPUTATION

From a parallel computing standpoint, an "ideal" computation can be divided into a number of completely independent parts, each of which can be executed by a separate processor. This computational paradigm is known as an embarrassingly parallel computation (Wilkinson and Allen *(10)*). A truly embarrassingly parallel computation suggests no communication between the separate processes and hence requires no special techniques or algorithms to obtain a working solution. In this ideal case, each process receives independent raw data and generates results from these inputs without waiting for results from other processes. The minimum execution time is achieved if all the available processors are assigned processes for the entire duration of the computation.

Nearly embarrassingly parallel computations are those architectures that require intermediate calculations and results to be distributed, collected, and combined in some way. This suggests that a single process must be operating alone initially and finally. For smooth operation of embarrassingly parallel computation, a master node is required in addition to slave computers. The master computer is used for controlling the computational sequence, such as distributing the tasks among the slave computers and waiting for a process to be completed before assigning subsequent tasks. The available slaves are the ones conducting the computationally intensive tasks.

In this study a master-slave parallelism is used. The optimization problem is decomposed and managed by a master process. The subtasks (calculations of objective functions) of nearly equal load are then assigned to slave nodes for processing, and subsequent

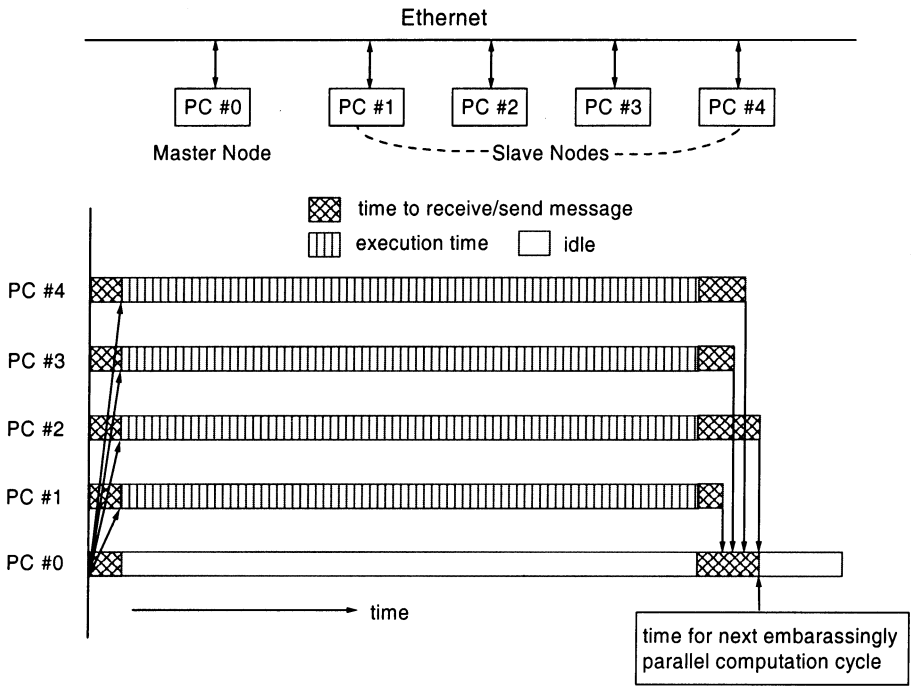

Fig. 1—The master-slave cluster configuration and message passing diagram of an embarrassingly parallel computation.

processes (search directions) are determined based on the results of slave processes. The only function for slave nodes is to process the task(s) assigned by the master node. Since the computational load for each of the slave nodes is about the same, a nearly embarrassingly parallel computation is achieved.

THE CLUSTER

The basic unit of a cluster is a single computer, so-called a node. In this study, a small-size cluster was constructed and composed of one master node and four slave nodes. The nodes were interconnected by means of a 100 Mbit/sec local area network. Figure 1 shows the configuration and message passing diagram of the cluster. The key specifications of the nodes of the cluster are listed in Table 1. To examine the effect of master capability on the cluster performance, two dramatically different master nodes were tested. In any instant, only one master node is required to manage the processes in the system.

Several software tools are freely available from the open source community for constructing a message-passing parallel computational environment. In this study, we use GNU/Linux as the kernel of the operating system. The communication software used is OpenMosix for node message passing. Since the kernel of each node is modified by the message-passing software for networked accessing only, users' programs and other resources will remain unchanged. The message passing software is also designed to prevent anyone from accessing any files on a remote node, but external access to the CPU cycles and memory is permitted. The programming language used is Fortran 90.

TABLE 1—SPECIFICATION OF THE MASTER-SLAVE CLUSTER

Node	CPU Type/MHz	RAM (MB)	Ethernet (Mbit/s)
Master*	Intel Pentium-II/233	128	
	Intel Pentium-IV/2000	512	100
Slaves	Intel Celeron/1000	256	

*Only one master node is required at a time.

CLUSTER PERFORMANCE

The primary reason for using parallel algorithms is to speed up computationally intensive applications. A simple measure of relative performance between a cluster with n slave computers and a single processor system is the speedup factor, $S(n)$, which is defined as

$$S(n) = \frac{t_s}{t_p} \quad [1]$$

where t_s is the execution time on a single processor and t_p is the execution time on the cluster. The execution time is the time elapsed when a program is executed in either serial or parallel on the unoccupied PCs. Except for some special cases (e.g., a much larger combined memory in a cluster than that in a single processor computer may facilitate some computational algorithm and reduce execution speed) the value of $S(n)$ is less than n. If a parallel algorithm did achieve speedup better than n, this normally suggests that the original sequential algorithm was not optimal. In practical applications, parallel algorithms are usually much different from the highly optimized serial algorithms. The need of a proper algorithm or procedure is crucial for efficient operation in a parallel computational environment.

The overheads appearing in a general parallel computation that may significantly reduce the speedup are (1) idle periods when some processors do not perform useful work; (2) extra computations in the parallel computing code that do not appear

TABLE 2—EXECUTION TIME (SEC.) IN THE NEARLY EMBARRASSINGLY PARALLEL COMPUTATION TESTS ON A MASTER-SLAVE CLUSTER WITH 4 SLAVE NODES

	Master Node Pentium IV/2.0 GHz 512 MB (5 Tasks for Each Slave-Node)		Master Node Pentium II/233 MHz 128 MB (5 Tasks for Each Slave-Node)	
	Tasks Executed Simultaneously	Tasks Executed One at a Time	Tasks Executed Simultaneously	Tasks Executed One at a Time
Trial no. 1	513.2	514.7	517.2	520.2
Trial no. 2	513.7	515.7	519.3	526.0
Trial no. 3	512.4	515.4	517.9	520.9
Average	513.1	515.3	518.1	522.4

in the sequential version; (3) communication time for sending and receiving messages. The first factor can be eliminated by adopting an embarrassingly parallel computation scheme. The second issue can also be addressed easily in the embarrassingly parallel computation environment due to the fact that in each calculation the coding is an optimized serial version, and hence not a less efficient parallel version in a slave node. The parallelization is carried out on the search process (evaluating the cost functions) controlled by the master node. The third concern is proved to be insignificant for the given procedure as can be seen in the latter part of this section.

Before adapting the embarrassingly parallel computation test using the master-slave configuration to the optimization problem, various computational loads were tested to examine their effect on speedup as well as the effectiveness of the master node on the system performance. There are two task-assignment methods in the current setup. If the master node allocates several tasks simultaneously to each of the slave nodes at the beginning of the process, the main overhead will be the communication time required at the start and the end of the process plus the task-switching time within each slave. On the other hand, if only one task is sent to each node at a time, although the multitasking overhead is eliminated, communication overhead is increased.

The task is to calculate the pressure distribution of the aforementioned 4-porous-pad air bearing with grid size 100. For given boundary conditions, the size of numerical grid in the 3-dimensional porous material and 2-dimensional air film affects the solution accuracy and the computational load. To compare the cluster performance in various configurations, three trials of five tasks assigned to slave nodes from different master nodes were conducted. The five tasks were either executed sequentially or simultaneously. The results of this master-slave parallelism simulation are listed in Table 2. As can be seen in the averaged results, the difference due to master node capability as well as conducting the tasks sequentially or simultaneously are small. The CPU loads of the first trial where the tasks were submitted sequentially or simultaneously by the Pentium II/233 master node are displayed in Figs. 2(a) and 2(b), respectively.

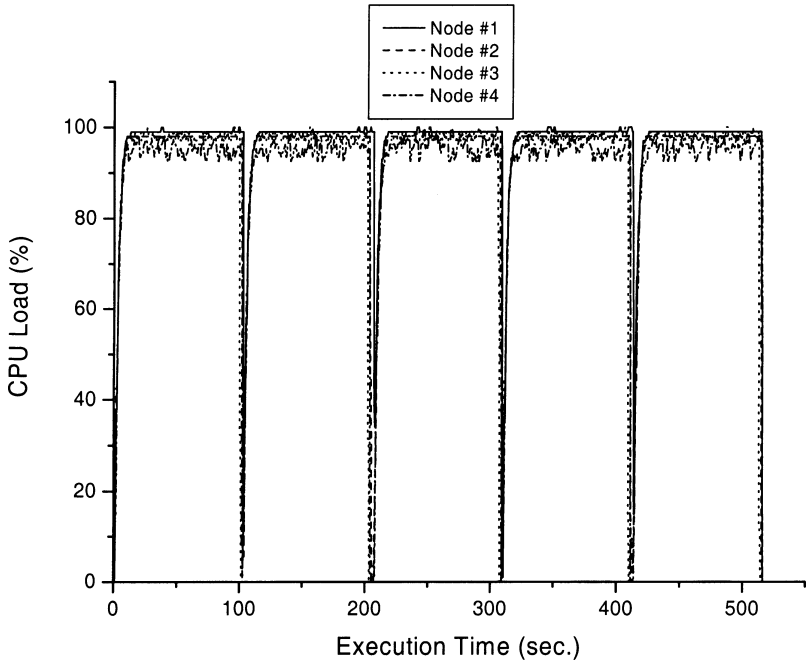

Fig. 2(a)—CPU load for sequentially assigned tasks in a master-slave cluster with four slave nodes (the configuration of master node is Pentium II/233 MHz with 128 MByte RAM).

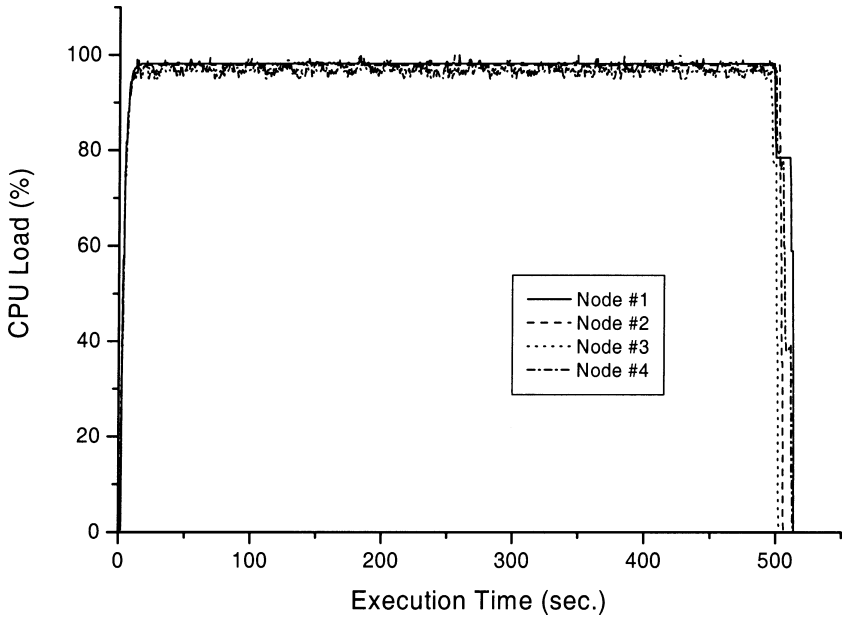

Fig. 2(b)—CPU load for simultaneously multiple-task computation in a master-slave cluster with four slave nodes (the configuration of master node is Pentium II/233 MHz with 128 MByte RAM).

To further examine the cluster performance, the effect of various loads on the cluster speedup in the nearly embarrassingly parallel computation environment was studied. In these tests, various computational loads were obtained by changing the grid size in the porous air bearing model. Regardless of the master node configuration, the speedups reach an asymptotic value when the execution time in each node exceeds 60 sec.; see Figs. 3(a) and 3(b). All the subsequent computations were thus conducted by using the Pentium II/233 node as the master node.

A cluster is usually built to accomplish heavy computational tasks. To simulate a situation of a practical cluster of large size, up to 60 tasks were assigned to a slave node to study the effect of the communication overhead on the system. In the simulation the percentage of the overhead is small when a large amount of small tasks were conducted simultaneously (Fig. 4). The efficiency for a node is similar to the speedup for a cluster, which is calculated by dividing the CPU time required by the execution time. The overhead comes mainly from the delays due to task-switching. A

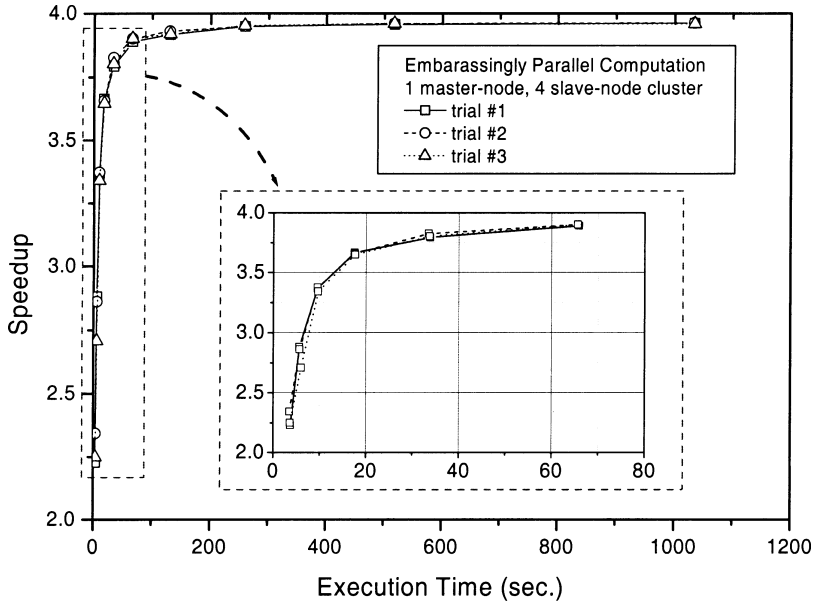

Fig. 3(a)—The speedups for various execution time in the simulated embarrassingly parallel computation (the configuration of master node is Pentium II/233 MHz with 128 MByte RAM).

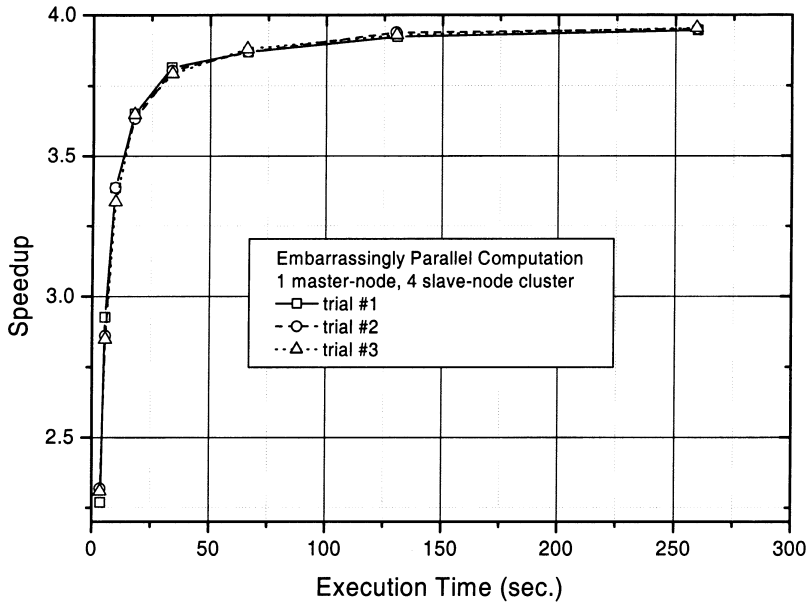

Fig. 3(b)—The speedups for various execution times in the simulated embarrassingly parallel computations (the configuration of master node is Pentium IV/2 GHz with 512 MByte RAM).

high speedup can thus be obtained in dealing with a problem with many design variables. If the task size is increased, the percentage of communication overhead will be reduced. The effect of task size on the node efficiency is shown in Fig. 5 for 20 tasks per node. As can be expected, the efficiency approaches unity as the load increased.

NUMERICAL PARALLEL OPTIMIZATION

In the lattice search method, a search step involves calculations of the corner points in a lattice. All the calculations are conducted using the same algorithms but for different design variables. The program for calculating the air bearing performance is precompiled and with a file size of 63.1 Kbyte. The input data for the program are the four design variables and the output is the value of the objective function. In the architecture of a typical optimization process, the master node allocates the executable program together with the design variables to the slave nodes. When all the slave nodes finish their calculations, the master node then collects the results from the slave nodes and determines the new set of design variables for the next step. Note that the execution time may vary slightly due to the different convergence rate with different inputs. However, each calculation will be independent of

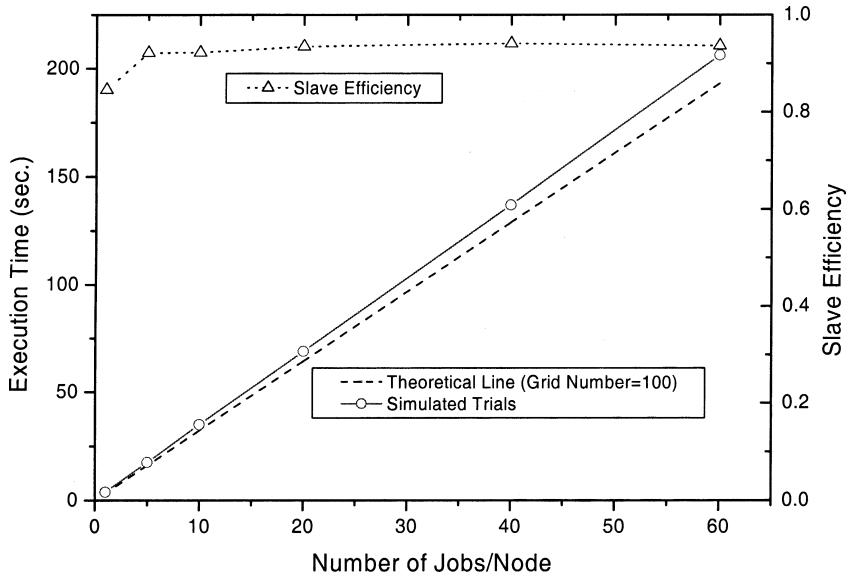

Fig. 4—The execution time and computational efficiency for various loads. The theoretical line is obtained by multiplying the execution time required in one task by the number of tasks assigned.

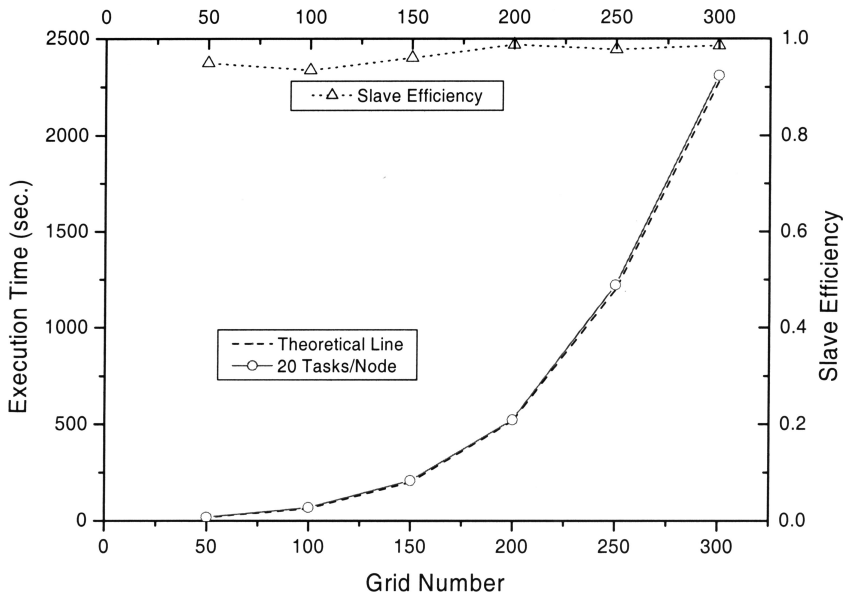

Fig. 5—The execution time and computational efficiency for various grid spacing in the simulated porous air bearing model. The theoretical line is obtained by multiplying the execution time required in one task to the number of tasks assigned.

the others and hence the architecture is in a good condition for nearly embarrassingly parallel computation.

For a converged solution the air bearing model was executed with a grid size of 200 in both the air film and the porous material in the following optimization case. In the simulated example the model has four design variables. The lattice method requires 8 corner-point calculations in a search step. For the 4-slave-node cluster, two function calculations were conducted concurrently in each of the slaves. The simplex method may require 2 to 3 calculations (reflection, expansion, or contraction) before constructing a new simplex for the next search.

Figure 6(a) shows the improved cost function by the simplex and lattice searches. Since the lattice search method is straightforward compared with the simplex method, the overall

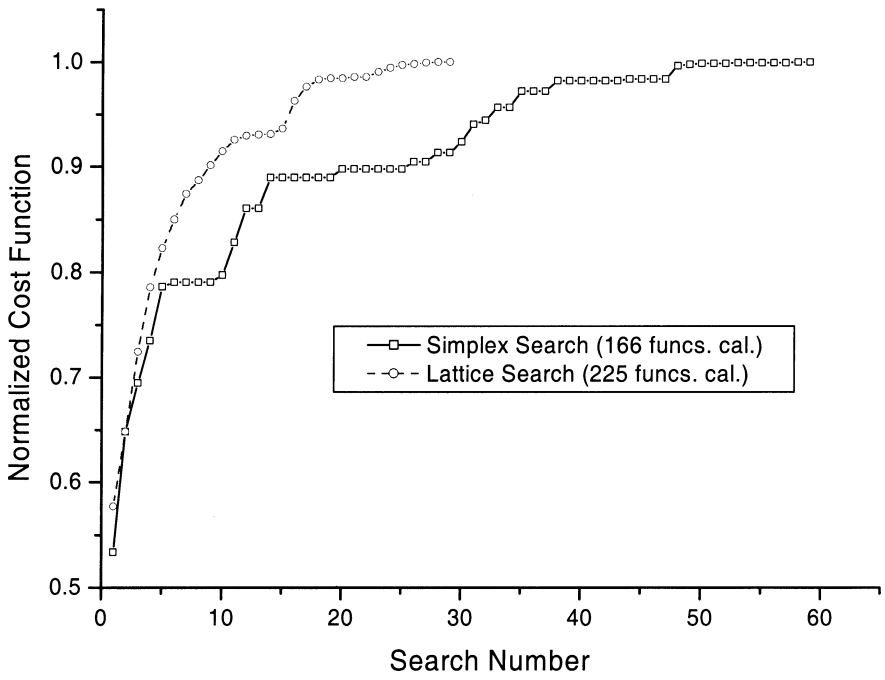

Fig. 6(a)—Search paths of the simplex and lattice methods. The CPU time is 1,377 sec. on the 4-slave cluster (5,443 sec. on a single-processor system) and 4,267 sec. for unparallelized simplex method.

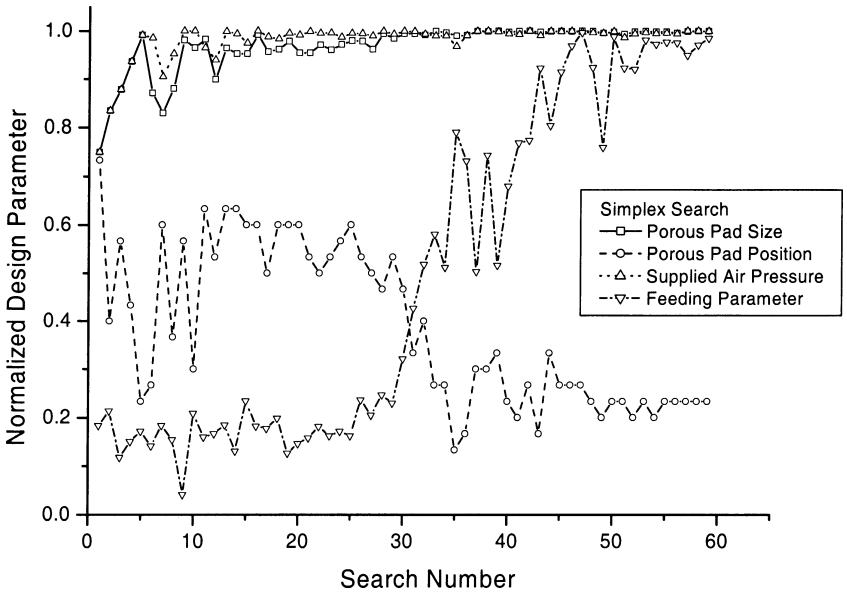

Fig. 6(b)—The search paths of design parameters of simplex solution in Fig. 6(a). The design parameters are normalized within their upper and lower bounds, respectively.

search process is usually with less search iterations but with a larger number of objective function evaluations. Note that the faster convergence rate of air film pressure in the lattice method is associated with better initial values estimated from the previous iteration. Comparing with the unparallelized simplex method, the CPU time can be significantly reduced from 4,267 to 1,377 sec. when the lattice method is used. The execute time can further be reduced to about a half if an 8-slave-node cluster is available instead of the 4-slave-node cluster. However, in this example, a cluster with more than 8 slave-nodes will not be able to decrease the execution time if a cost function evaluation cannot be further divided into smaller tasks. The task-dividing is essential for an efficient implementation of parallel computation.

Figures 6(b) and 6(c) show the search paths of design variables when the simplex and lattice methods are adopted. The drop-the-poorest-point mechanism of simplex search is characterized by the zigzag search pattern, while the lattice method demonstrates the search-in-all-corner capability, which demands computational power in a single search step. The pattern of the search path of the design variables indicates the steady improvement of the objective.

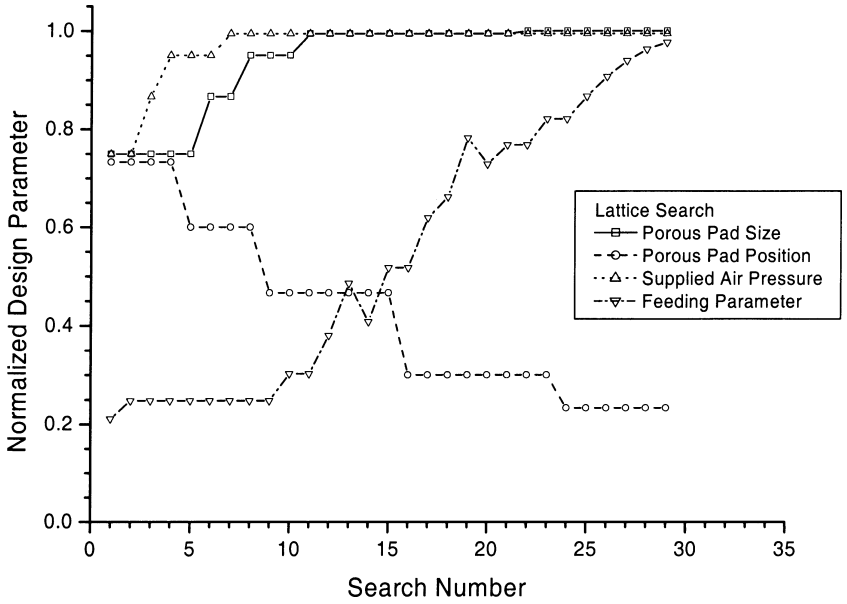

Fig. 6(c)—The search paths of design parameters of lattice solution in Fig. 6(a). The design parameters are normalized within their upper and lower bounds, respectively.

It can be seen that the lattice search is a suitable, but not necessarily the most efficient, search method in the parallel-computation environment.

CONCLUSIONS AND RECOMMENDATIONS

The transportation speed of electrons in the current semiconductor computer provides an upper bound on the speed at which serial computations can be conducted, and parallel designs appear as the only way to overcome this barrier. Due to the availability of inexpensive hardware and mature open-source software, parallel computation has become a choice in computationally intensive applications. This may fundamentally change the numerical approaches and computation techniques in the fluid film lubrication analysis. Algorithms performed in an embarrassingly parallel fashion are practical in a wide range of applications, especially in the area of optimum design. In this study, although a small-scale cluster is used for the illustrated example, the multitask tests demonstrate the usability of larger clusters with reasonable speedups. This article demonstrates an optimization approach that does not require a special multiprocessor system. Therefore, researchers can concentrate on parallel programs that can be executed on networked desktop PCs using freely available software tools.

On the other hand, one should try to optimize the serial-processing process first and then implement it in a parallel environment. For example, in the porous air bearing analysis (Wang and Chang *(8)*), the three relaxation factors used in the analysis should be chosen carefully. This will benefit the computation in the parallel computation if each cost function calculation is conducted in a node. The overall efficiency can thus be maximized in the cluster regardless of optimization method used.

To achieve high speedup without increasing the coding complexity, a master computer conducting load balance in a cluster is suggested. As illustrated in this article, the effect of various master configurations on the cluster performance is small. The results obtained by the parallelized lattice method are compared with that of an unparallelized simplex method and indicate a significant reduction in execution time due to parallelism. A small cluster with more than 4 slave nodes can significantly reduce the execution time for a computationally intensive optimization problem with speedup close to the number of available slaves. The flexibility of dividing the computational work into proper-sized tasks in order to effectively utilize the parallel architecture is also important for parallel computation. This study provides a framework for optimization applications with complex tribological models to be solved with minimum execution time.

REFERENCES

(1) Akl, S. G. (1997), *Parallel Computation: Models and Methods*, Prentice-Hall, New Jersey.

(2) Hanke, A. and Talke, F. (2002), "A New Universal Approach for Slider Optimization," in *Proc. of ASME/STLE Trib. Conf.*, ASME Paper No. **2002-TRIB-263**.

(3) Kato, T. and Soutome, H. (2000), "Friction Material Design for Brake Pads Using Database," *Trib. Trans.*, **44**, pp 137-141.

(4) Kotera, H. and Shima, S. (2000), "Shape Optimization to Perform Prescribed Air Lubrication Using Genetic Algorithm," *Trib. Trans.*, **43**, pp 837-841.

(5) O'Hara, M. A., Hu, Y. and Bogy, D. B. (2000), "Optimization of Proximity Recording Air Bearing Sliders in Magnetic Hard Disk Drives," *ASME Trans., Jour. of Trib.*, **122**, pp 257-259.

(6) Rao, S. S. (1996), *Engineering Optimization—Theory and Practice*, 3rd Ed., John Wiley & Sons, New York.

(7) Seireg, A. A. and Dodriguez, J. (1997), *Optimizing the Shape of Mechanical Elements and Structures*, Marcel Dekker, Inc.

(8) Wang, N. and Chang, Y.-Z. (2002), "A Hybrid Search Algorithm for Porous Air Bearings Optimization," *Trib. Trans.*, **45**, pp 471-477.

(9) Wang, N., Ho, C. and Cha, K. (2000), "Engineering Optimum Design of Fluid-Film Lubricated Bearings," *Trib. Trans.*, **43**, pp 377-386.

(10) Wilkinson, B. and Allen, M. (1999), *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice-Hall, New Jersey.