A Study on Optimal Task Decomposition of Networked Parallel Computing Using PVM

Kwanjae Seong* and Han-Gyoo Kim**

(*Received October 30, 1996*)

A numerical study is performed to investigate the effect of task decomposition on networked parallel processes using Parallel Virtual Machine (PVM). In our study, a PVM program distributed over a network of workstations is used in solving a finite difference version of a one dimensional heat equation, where natural choice of PVM programming structure would be the master-slave paradigm, with the aim of finding an optimal configuration resulting in least computing time including communication overhead among machines. Given a set of PVM tasks comprised of one master and five slave programs, it is found that there exists a pseudo-optimal number of machines, which does not necessarily coincide with the number of tasks, that yields the best performance when the network is under a light usage. Increasing the number of machines beyond this optimal one does not improve computing performance since increase in communication overhead among the excess number of machines offsets the decrease in CPU time obtained by distributing the PVM tasks among these machines. However, when the network traffic is heavy, the results exhibit a more random characteristic that is explained by the random nature of data transfer time.

Key Words: Communication Overhead, Heat Equation, Networked Parallel Computing, Network Traffic, Optimal Task Decomposition, PVM

1. Introduction

Parallel processing, the method of having many small tasks solve one large problem, has recently received much attention with increasing demand for higher performance, lower cost and sustained productivity, and has emerged as a key enabling technology in modern computing. This adoption has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing.

At present, MPPs provide the most powerful environment under which high computational power can be attained, combining a few hundred to a few thousand CPUs connected to hundreds of gigabytes of memory. (Alamasi and Gottlieb, 1994) However, MPPs suffer from two drawbacks: economy and availability.

Large MPPs typically cost more than 10 million U. S. dollars. The cost in running programs on an MPP is usually more than tenfold compared to that of running equivalent ones on a set of networked workstations. Aside from the price of MPPs, lack of effective compilers and application programming interfaces (APIs) further restrict the availability of them to a limited number of users and even then they might have to spend a considerable amount of time familiarizing themselves with MPPs before they are ready to tackle real problems. (Anderson et al., 1995)

On the other hand, distributed computing over networked workstations is gaining its edge over MPPs because of its widespread availability and economic advantage. Distributed computing provides well defined APIs easily applicable to various programs as well as effective compilers over a network of workstations, resulting in cost effec-

* Department of Mechanical Engineering, Dong-Guk University, Seoul 100-715, Korea
** Department of Computer Engineering, Hong-Ik University, Seoul 121-791, Korea

tive solutions.

PVM (Parallel Virtual Machine) is one of the most promising distributed computing systems available, and may be applied to a network of heterogeneous workstations (Sunderam et. al. 1990). PVM provides easy to program APIs through which complex and CPU intensive scientific problems, such as global climate modeling and new drug design, can be solved without relying on expensive MPPs by decomposing them into a set of simple tasks manageable on a workstation or even a PC.

PVM, however, suffers from an inherent problem of overhead due to communication between tasks, since each task in a PVM program needs to exchange data with others distributed over the network. A seemingly optimal decomposition in the traditional sense will not guarantee an optimal solution under the PVM environment, especially if such decomposition entails more communication overhead than performance gain. Therefore, it is necessary to investigate how a large problem should be decomposed into a set of PVM tasks which will produce the most performance gain taking into account the communication overhead.

Objective of this study is not in any specific engineering problem itself but to perform a parametric numerical experiment with PVM on its performance by solving a transient one dimensional heat conduction problem. In this paper, we present our experience with PVM in pursuit of finding a pseudo-optimal decomposition of PVM tasks for the specific problem of a one dimensional heat diffusion. It will only be a *pseudo* -optimal, since communication overhead is affected by nondeterministic factors such as actual data transfer rate and the network load at time of execution. We will investigate how total execution time of a finite difference program processed in parallel varies as number of homogeneous machines on a network is varied along with the effects of discretization size in both space and time.

2. PVM: Parallel Virtual Machine

The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing for high performance scientific computing. Furthermore, the message passing model appears to be gaining predominance as the paradigm choice of parallel programming. PVM is a software infrastructure that emulates a generalized distributed memory multiprocessor in heterogeneous networked environment. PVM supports a straightforward but functionally complete message passing model (McByran, 1994), and is capable of harnessing the combined resources of typically heterogeneous networked computing platforms to deliver high levels of performance and functionality.

Parallel computing using PVM may be approached from three fundamentally different viewpoints based on the organization of computing tasks. (Kim et al. 1996) The first and the most common model for PVM applications can be termed crowd computing, where a collection of loosely related processes, typically executing the same code, performs computations on different portions of the workload usually involving periodic exchanges of intermediate results. The second model supported by PVM is termed a *tree* computation. In this scenario, processes are dynamically spawned in a tree-like manner. This paradigm is a natural fit to applications where the total workload is not known a priori, for example as in recursive divide-and-conquer algorithms. The third model termed *hybrid* is a combination of the tree and the crowd model.

As mentioned above, PVM provides various methods of task decomposition that fit into modeling of diverse scientific problems as well as those of general purpose. These models assume that any PVM task can send a message to any other task and that there is no limit to the size or number of such messages. It also supports multicast of a message to a user defined group of tasks. The choice of model will be application dependent, and should be selected to best match the natural structure of the parallelized program, while taking into account the communication overhead. Skeleton of the actual PVM program used in our experiment is presented below, and can be used as an example to understand PVM user interfaces, but readers should refer to PVM user's manual (e. g. Geist *et al*., 1994) for complete understanding of PVM APIs.

3. Numerical Experiment with Heat Equation

In this section we present a PVM program that calculates heat diffusion in a thin wire by solving the finite difference version of a one dimensional heat equation. To accomplish our aim, as mentioned in the introduction, it suffices that we choose to analyze a most simple problem preferably to which an analytic solution is also available.

Consider a thin wire of length L, density ρ, specific heat c and thermal conductivity k with the ends of the wire maintained at a fixed temperature of T_e and an initial temperature profile of:

$$T(z, \tau=0)=T_e+T_0 \sin\left(\pi\frac{z}{L}\right) \qquad (1)$$

With the usual non-dimensionalization, where $A=(T-T_e)/T_0$, $t=\tau/L^2\left(\frac{\rho c}{k}\right)$ and $x=z/L$, temperature in the wire is described by the following heat equation;

$$\frac{\partial^2 A}{\partial x^2}=\frac{\partial A}{\partial t} \qquad (2)$$

with the initial and boundary conditions of;

$$A(x, t=0)=\sin(\pi x) \qquad (3a)$$

$$A(x=0, t)=A(x=1, t)=0 \qquad (3b)$$

The exact solution of Eq. (2) subject to (3a) and (3b) can be found by the method of separation of variables as (Meyers, 1971);

$$A(x, t)=e^{-\pi^2 t}\sin(\pi x) \qquad (4)$$

Finite-difference solution to the above problem will be sought via parallel processing and in particular through distributed computing using the PVM program.

We will adopt an explicit scheme with forward differencing in time and central differencing in space, and hence the solution can be obtained by marching in time from a given initial temperature distribution. If we denote $A(x_i, t_j)$ as $A_{i,j}$, temperature at position x_i and at time t_{j+1} can be expressed as;

$$A_{i,j+1}=\beta(A_{i+1,j}+A_{i-1,j}) + (1-2\beta)A_{i,j} \qquad (5)$$

where $\beta=\frac{\Delta t}{(\Delta x)^2}$ and the stability criterion for the explicit scheme requires that $\beta\leq 1/2$.

For our problem, where the solution over the whole space can be obtained through the same equation as provided by Eq. (5), natural choice of programming structure would be the master-slave method of the crowd programming paradigm where the slaves, spawned by the master program, perform the actual computations. We choose to divide the wire into 5 subsections, and the solutions to each subsection will be obtained separately by each slave programs, although it will be required that the right most and the left most temperature information be exchanged with its right and left neighboring subsections. The workload of the slaves are allocated by the master through data decomposition whereby the initial temperature distribution of each subsection is sent to respective slaves.

Overall structure of the parallel processing is as follows:

The master program spawns 5 copies of the same slave program, each of which handles a subsection of the wire. After receiving initial temperature distribution, each slave computes heat diffusion in the corresponding wire subsection. At each time step, each slave program needs to communicate boundary information with its left and right neighboring slaves. When a specified final time is reached, all 5 slave programs send its final temperature profile to the master, who then terminates the spawned slaves and ends the program.

In order to study the effect of communication overhead on the total CPU time, same program was executed on 1 to 6 machines on the network. When only one machine is used, the master program along with all of the five slaves are executed on the same processor on the same machine, in which case there is no additional time spent in communication over the network and can be regarded as a serial processing. When more than one machine is utilized, the master and the slave programs are allocated to each machine such that an even distribution of workload is achieved. For example, with four machines, the first machine handles the master program along with one of the five slaves and the second machine handles two slaves with the remaining two each executing one slave program.

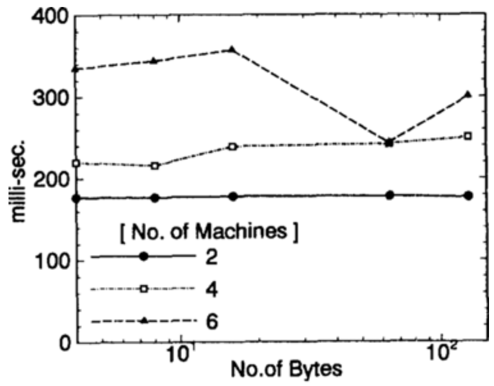
The finite difference PVM program was executed on six different configurations as mentioned above with four values of $\triangle x$ ranging from 1/50 to 1/1000 and four $\triangle t$'s for each $\triangle x$ ranging from 2.0×10^{-4} to 6.25×10^{-8} until a preset final time is reached. The final time was chosen such that for each different $\triangle x$, the largest $\triangle t$ results in 750 iterations and the smallest 6000. Hence for a given $\triangle x$, total number of iterations on Eq. (5) is inversely proportional to the size of $\triangle t$. The final temperature profile for all of the above parameter values resulted in essentially the analytic solution of Eq. (4). The total computing time elapsed in carrying out this numerical experiment on a network of Sun Sparc workstations were recorded and are analyzed in the following section.

4. Results and Discussions

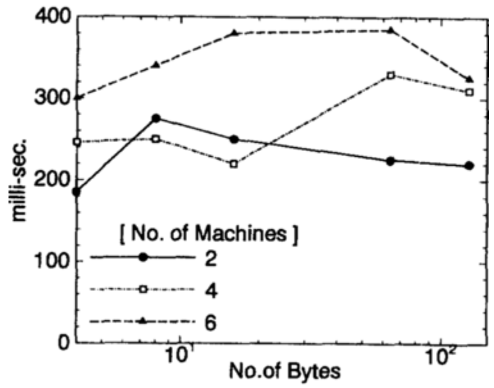
Results analyzed in this section are obtained from a group of networked Sun Sparc Classic workstations running on Sun OS 4.2. Total computing time, including data communication overhead between machines, will definitely depend on the degree of network usage at the time of execution and numerical experiments were carried out under two different conditions; 1) when the network is under a light usage during night time which will be referred to as 'light traffic' and 2) at day time when the network is under a typical usage of a university engineering department referred to as 'heavy traffic'. Results used in the analysis below are only one out of many trials executed under similar working environment.

4.1 Data communication overhead

To investigate the effect of data transfer over the network on the total computing time, time taken solely in data transfer between machines were investigated and the results are presented in Fig. 1. This includes time taken for PVM setup plus network latency and data transmission. In both cases of light and heavy traffic, it can be seen that the increase in the amount of data will not necessarily result in a larger communication overhead but will be strongly dependent on the number of machines on the network. This is especially true when the traffic is light whereas under heavy traffic conditions it exhibits a more random nature as may be expected. However the results of



(a) Light traffic



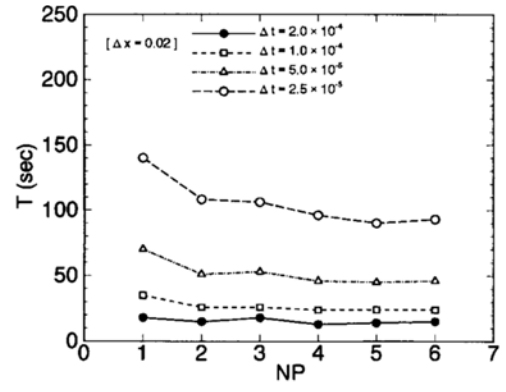
(b) Heavy traffic

Fig. 1 Communication time between machines for data transfer.

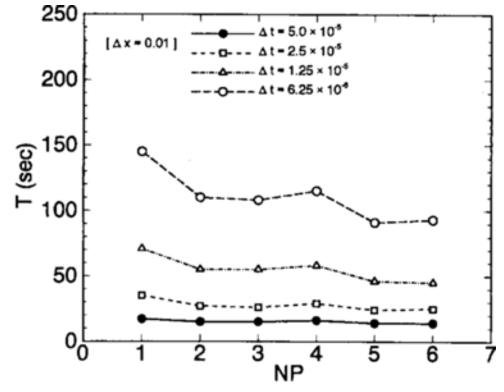
Fig. 1 clearly show that data communication time is bounded, at least with PVM and data transfer between 2 to 6 machines. One can also deduce that the PVM setup time is about 190 ms for our experimental environment and the data transfer time exhibits some randomness reflecting the characteristics of Ethernet.

4.2 Total computation time

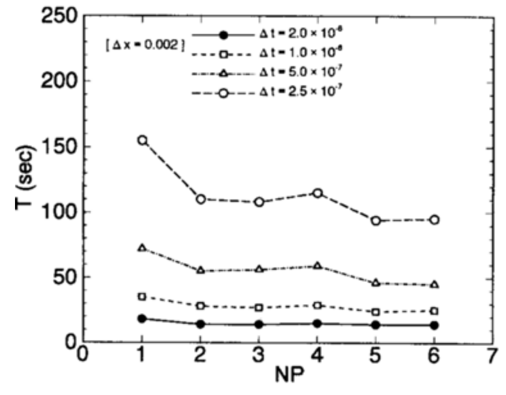
Figs. 2 and 3 describe the total computation time versus the number of machines utilized by the PVM program under light and heavy traffic conditions respectively. The most important result is that for all cases under light traffic, there seems to exist a pseudo-optimal configuration, i. e. least computation time, of NP=4, 5 or 6. And this pseudo-optimum becomes more pronounced as computation load increases (see Fig. 2(d)). This can be explained by the fact that total computation time is comprised of actual CPU time, which decreases as more machines are utilized and workload is more evenly distributed, and communication overhead, which increases with NP as more data transfers are required between more machines. Hence total computation time is the least when gains obtained from work distribution minus the increase in communication overhead is the largest. For our problem it can be concluded that this optimum configuration is obtained when 4 to 6 machines are utilized. It should be remembered that this analysis is based on the fact that the programs were executed under light traffic conditions and data transfer time is almost constant (see Fig. 1(a)). Also, we can conclude that



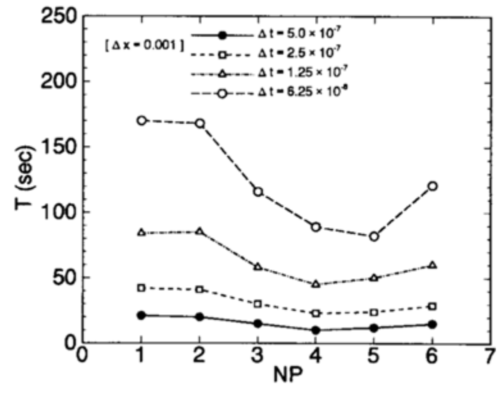
(a) $\triangle x==0.02$



(b) $\triangle x=0.01$



(c) $\triangle x=0.005$



(d) $\triangle x=0.001$

Fig. 2 Total computing time, T, under light traffic condition as a function of the number of machines, NP, utilized by the task. NP=1 corresponds to serial processing without communication overhead.

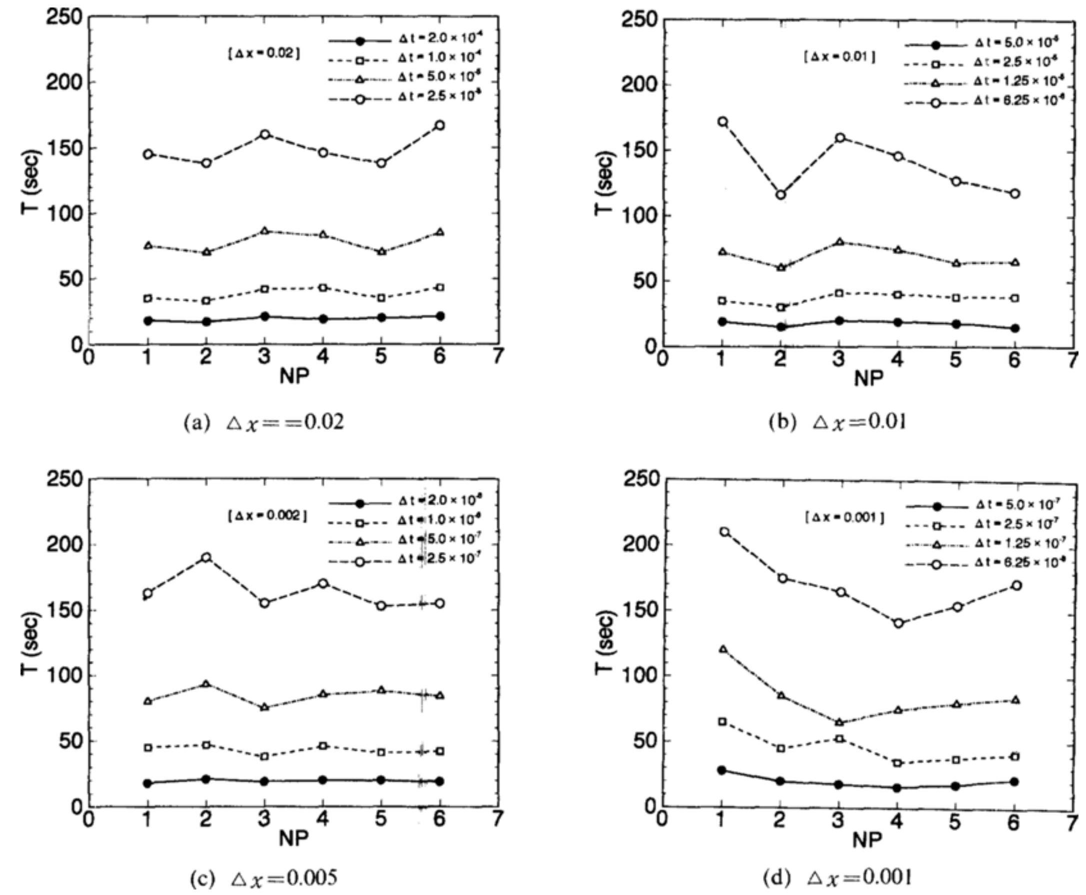


Fig. 3 Total computing time, T, under heavy trafic condition as a function of the number of machines, NP, utilized by the task. NP=I corresponds to serial processing without communication overhead.

our problem is such that communication overhead is of comparable order to actual CPU time, especially when computation burden is not low (e. g. for small $\triangle x$) and hence different types of configuration do play an important role regarding total computation time.

When the traffic is heavy and communication overhead is random as depicted in Fig. 1 (b), total computation time presented in Fig. 3 is also random. When the computation load is light total computation time shows almost no visible variations with respect to the number of machines utilized. However, even under heavy traffic performance, it is seen to improve when workload is distributed compared to a single machine configuration and especially so when the computation burden is high.

4.3 Conclusions

In this study we performed a numerical experiment over a workstation cluster connected by Ethernet using PVM to solve a simple one dimensional heat equation with the purpose of investigating how communication overhead affects the total computation time and obtaining, if any, a pseudo-optimal configuration and task decomposition. We found that the total computation time including communication overhead shows qualitatively different characteristics depending on the network usage at the time of program execution. When the traffic is light and data transfer time is relatively constant, there seems to exit an optimal task decomposition set by decrease in CPU time and increase in communication overhead as more machines are utilized. However, when the traffic is heavy with data transfer time being more random, total computation time is also random without any systematic improvement as workload is distributed among more machines. But it is clear that in most of the cases studied, parallel computation is no worse than serial computation in the worst case and shows some improvement over serial computation in many cases.

Although the purpose of our study is in obtaining the optimal task decomposition of PVM tasks under Ethernet which is widely available and not in finding the best network environment for PVM programming, it is worth mentioning that the results of our experiment will be strongly dependent on the types of network subsystems such as the ring or the ATM-like switch based network which will display different characteristics of the program behavior. From our experience with PVM under Ethernet environment, we find that when communication overhead is of comparable order as CPU time, which may by true for many types of problems, performance gains from an optimal task decomposition compared to a more heuristic one is such that it may not warrant an exhaustive investigation unless the program is to be executed under the network condition where a light traffic is expected.

References

Alamasi, G. and Gottlieb, A., 1994, *Highly Parallel Computing*, Benjamin Cummings Pub. Co., Redwood City, CA.

Anderson, T., Culler, D. and Patterson, D., 1995, "A Case for NOW (Networks of Workstations)," *IEEE Micro*, Vol. 15, No. 10, pp. 54~64.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V., 1994, PVM: *Parallel Virtual Machine-A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA.

Kim, H. G., Seong, K. J. and Kim, S. H., 1996, "A Numerical Experiment on Network Parallel Computing Using PVM," *Proc. of KISS*, Vol. 23, No. 2, pp. 1031~1034

McByran, O., 1994, "An Overview of Message Passing Environments," *Parallel Computing*, Vol. 20, pp. 417~444.

Meyers, G. E., 1971, *Analytical Methods in Conduction Heat Transfer*, McGraw-Hill, New York, pp. 271~274.

Sunderam, V. S., 1990, "PVM: A Framework for Parallel Distributed Computing," *J. Concurrency*, Vol. 2, No. 4, pp. 315~339.