A Processors Management System for PVM

Jose Aguilar Tania Jimenez

CEMISID, Dpto. de Computación
Fac. Ingeniería, Universidad de Los Andes
5101 Mérida, Venezuela.
e-mail : {aguilar,tania}@ing.ula.ve

Abstract. Currently, PVM constitutes a widely used software for developing parallel applications in workstation and parallel environments. In this paper we propose a processors management system for PVM which allows to assign the PVM tasks over a computers system. The Processors Management System uses two task assignment heuristics. These heuristics are based on Neural Networks and Genetic Algorithms.

1 Introduction

PVM [5] is a widely used, public-domain software system that allows an heterogeneous network of parallel/serial computers to be used as a single computational resource. PVM allows the computing power of widely available, general-purpose computer networks to be harnessed for parallel processing. Nevertheless, PVM requires more support for intelligent allocation, resource/file management, etc.

The integration of allocation systems to PVM is a well studied subject[2]: DynamicPVM, MPVM, CoCheck, UPVM, etc. In this paper, we propose a Processors Management System (PMS) for PVM. The PMS is responsible for allocating processors among all parallel programs submitted to the system using PVM. The static task allocation decision will be obtained using Task Assignment Heuristics (TAHs) based on the Random Neural Model of Gelenbe [1, 3] and the Genetic Algorithms [1, 4]. All requests related to host addition, or deletion, and queries about the state of the system should be in the domain of PVM.

2 Design Issues

Submitting a parallel program into PVM can be described as follows. Usually, a task of PVM acts as the master and creates other tasks dynamically. A PVM master task has *pvm-spawn* calls creating successors tasks which are added to the tasks list controlled by the PVM daemon. These can also create new tasks.

In our context, we use a Tasks Graph (TG) to model the creation and execution of PVM tasks, that is a program is modeled as a directed acyclic *series – parallel* TG. The decomposition of the parallel programs into several cooperating PVM tasks by means of primitives provided by PVM are using to build the TG. In this way, a parallel program is represented as a collection of tasks which correspond to nodes in a graph. The arcs of the graph represent communication between tasks and precedence relations. We denote the TG by $G = (N, A)$, where $N = \{1, .., n\}$ is the set of n tasks of the program and $A = \{a_{ij}\}$ is the adjacency matrix (precedence order between tasks).

The static task assignment is then formulated as a graph partitioning problem. The problem consists of the assignment of the n tasks to K processors in such a way that the communication times between different processors of the system must be kept to a minimum and the load at different processors must be balanced. These goals are represented in the following cost function:

$$F(G_z) = \sum_{i,j \in D} \tau_{ij} + b\frac{\sum_{z=1}^{K} (N_{G_z} - n/K)^2}{K} \tag{1}$$

where,
τ_{ij} = communication cost between task i and j
$D = \{i \in G_m \ \& \ j \in G_l \ \& \ l \neq m \ \& \ a_{ij} = 1\}$
N_{G_z}= number of tasks assigned to processor z (in partition G_z)
b = factor of load balancing. In our case, $b = 1$.

In general, this problem is well known to be NP complete, that is the reason of using heuristics to obtain an optimal solution. Our PMS is constituted by the next phases:

2.1 Task Graph Generation

It is initiated by an allocation condition (demanding to execute a new parallel program to PVM). Then, the PMS generates the TG of the program, saves the required information about the system to allocate the new tasks, and disconnects PVM master task from its pvmd.

2.2 Task Allocation

In this phase is made a distribution of the tasks over the host machines in the system based on the actual system state. The new TG is submitted to the TAHs, which assign the corresponding tasks to the processors. We have used the following TAHs for solving the static task assignment problem:

- *The Random Neural Model (RNM):* The RNM has been developed by Gelenbe [3] to represent a dynamic behavior inspired by natural neural systems. The basic descriptor of a RNM is the i-th neuron's probability of being excited $q(i)$, which satisfy the following set of non-linear equations:

$$q(i) = \frac{\sum_{j=1}^{n} q(j)r(j)P^{+}(j,i) + \Lambda(i)}{\sum_{j=1}^{n} q(j)r(j)P^{-}(j,i) + \lambda(i)} \tag{2}$$

  where:

  - $\Lambda(i)$ is the rate at which *external excitation signals* arrive to the i-th neuron,

- $\lambda(i)$ is the rate at which *external inhibition signals* arrive to the i-th neuron,
- r(i) is the rate at which neuron i fires when it is excited,
- $P^{+}(i,j)$ and $P^{-}(i,j)$, are the probabilities that neuron i (when is excited) will send an *excitation* or an *inhibition* signal to neuron j.

Using this approach [1], we have constructed a RNM composed of $nK + K$ neurons. For each pair (i, u) (task, processor) we will have a neuron $\mu(i, u)$ (there are nK neurons of this type). We will denote by $q(\mu(i, u))$ the probability that neuron $\mu(i, u)$ is excited, if this probability is close to 1 then task i should be assigned to processor u. For each processor u there will be a neuron $\pi(u)$. Hence, there are K neurons of this type whose role is to indicate whether processor u is heavily loaded.

- *The Genetic Algorithms (GA):* This heuristic is based in the concept of reproduction of individuals and their evolution. The GA allows an "intelligent evolution" of the individuals using evolution operators such as mutation, inversion, selection and crossover [4]. We describe next the GA heuristic applied to the task assignment problem [1]: Let define a space of research of n vectors where each one represents an individual, and each individual represents a possible solution. Each vector has n elements, each element takes one value in the set $\{1...K\}$, depending on the partition of the TG to which it belongs. In this vector the *ith* component corresponds to task i, and if its final value is u, that means that task i will be executed in processor u. Furthermore, we use the cost function $F(G_z)$ to determine the cost of each solution. The algorithm starts with an initial population of individuals randomly defined and the individuals yielding the minimal cost are chosen to generate new individuals, using the genetic operators. Since the population is constant, we substitute the worst individuals of initial solution by the best individuals generated. The procedure stops if a given number of generations is exceeded without finding a better solution.

2.3 Task Restart

This phase restarts the master tasks and saves the allocation decision in a vector. Then, for every new execution condition (demanding to spawn new PVM tasks) it allocates these tasks of the program according to the allocation decision.

3 Evaluation of Performance of the PMS

In this section we summarize the results we have obtained for our PMS, which we compare with PVM without RM and MPVM[2]. Comparisons are carried out for two parallel programs: the first one is the standard matrix $(n * n)$ multiplication parallel algorithm (table 1), the second one calculates the Fast Fourier Transform (n coefficients)(table 2). The simulations were made in a network of 6 SUN workstations (Sparc V). In these tables, for PMS, the first value represents the execution time of the parallel program, and the second one of the TAHs. It is also quite clear that when $n > 20$ our PMS versions are very time consuming in program execution time (it is due to our TAHs). On the other hand, our heuristics give good results if we do not include the execution time of our TAHs. Interestingly enough, the GA generally provides results which are substantially better than the RNM.

– Table 1. Execution Time for Matrix Multiplication Algorithm

n	PMS with GA	PMS with RNM	MPVM	PVM
20	4.6/0.8	4.4/0.8	3.7	3.3
50	15.4/8	19.1/11.2	7.5	6.8
100	29.6/19.2	34.4/23.3	10.5	12.4
500	39.2/25	50.1/34	14.3	18

– Table 2. Execution Time for FTT algorithm

n	PMS with GA	PMS with RNM	MPVM	PVM
10	5.9/1.3	6/1.5	4.3	4.4
20	9.3/2	11.7/3.5	7.8	7.8
50	22.4/13.1	24.3/14.2	9.8	10.2

4 Conclusions

Our PMS generally gives good execution time for the parallel programs, but with a substantially larger execution time to solve the static task allocation problem. This is because the computations of our TAHs are time consuming. The GA and the RNM based heuristics could be easy to implement on a parallel machine, and this can considerably improve the speed with which our PMS obtain the task allocation.

References

[1] Aguilar, J. : L'Allocation de tâches, l'équilibrage de charge et l'optimisation combinatoire. PhD thesis, René Descartes University 1995.

[2] Casas, J. et al. : MPVM: a migration transparent version of PVM. Technical Report. Dept. of Computer Science and Engineering, Oregon Graduate Institute of Science and technology.

[3] Gelenbe, E. : Random neural networks with positive and negative signals and product form solution. Neural Computation **1** (1989) 502-511.

[4] Goldberg D. : Genetic algorithms in search, optimization and machine learning. Addison-Wesley, 1989.

[5] Sunderam, V. : PVM: a framework for parallel distributed computing. Concurrency: Practice and Experience, **2** (1990) 315-339.