Parallel Ant Systems applied to the Multiple Knapsack Problem

Marcelo Cena, Guillermo Leguizamón
Proyecto UNSL N° 338403*
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis - Argentina
e-mail: {mcena,legui}@unsl.edu.ar
Fax: +54 652 30224

Keywords: parallel programming, distributed systems, ant colony systems, combinatorial optimization, multiple knapsack problem.

Abstract

Interesting real world combinatorial problems are NP-complete and many of them are hard to solve by using traditional methods. However, several heuristic methods have been developed in order to obtain timely suboptimal solutions. Most of those heuristic methods are also naturally suitable for a parallel implementation and consequently, an additional improvement on the response time can be obtained. One way of increasing the computational power is by using multiple processors operating together on a single problem. The overall problem is split into parts, each of which is operated by a separate processor in parallel. Unfortunately problems cannot be divided perfectly into separate parts and interaction is necessary between the parts like data transfer and process synchronization. However, substantial improvement can be achieved, depending on the problem and the amount of parallelism in the problem. Our work aims to exploit the capability of a distributed computing environment by using PVM and implementing a parallel version of an Ant System for solving the Multiple Knapsack Problem (MKP). An Ant System (a distributed algorithm) is a set of agents working independently and cooperating sporadically in a common problem solving activity. Regarding the above characteristics, an Ant System can be naturally considered as a nearly embarrassingly parallel computation. The proposed parallel implementations of an Ant System are based on two different approaches, static and dynamic task assignment. The computational study involves processors of different velocities and several MKP test cases of different sizes and difficulties (tight and loose constraints). The performance on the response time is measured by two indexes, Speedup Factor and Efficiency when is compared to a serial version of an Ant System. The results obtained show the potential power of exploiting the parallelism underlying in an Ant System regarding the good quality of the results and a remarkable decreasing on the computation time.

*The research group is supported by U.N.S.L and ANPCyT (Agencia Nacional para la Promoción de la Ciencia y la Tecnología)
Parallel Ant Systems applied to the Multiple Knapsack Problem

1 Introduction

Parallel programming uses multiple computers, or computers with multiple internal processors, to solve a problem at a greater speed than using a single computer. Areas requiring great computational speed include optimization, numerical modeling and simulation of problems in science and engineering, which often need huge repetitive calculations on large amounts of data to give valid results. However, apart from having an algorithmic solution and the amount of memory required, the execution time is a key issue. One way of increasing the computational power is by using multiple processors operating together on a single problem. The overall problem is split into parts, each of which is operated upon by a separate processor in parallel. Unfortunately problems cannot be divided perfectly into separate parts and interaction is necessary between the parts like data transfer and process synchronization. However substantial improvement can be achieved, depending on the problem and the amount of parallelism in the problem. Moreover, for obtaining the potential for increased speed on an existing problem, the use of multiple computers/processors often allows a larger or more precise instance of a problem be solved in a reasonable time.

There exist several types of computer platforms suitable for implementing parallel applications. Our work aims to exploit the capability of a distributed computing environment by using PVM package (Parallel Virtual Machine), and implementing a parallel version of an Ant System for solving the Multiple Knapsack Problem (MKP). The proposed parallel implementations of an Ant System are based on two different approaches, static and dynamic task assignment. The computational study involves processors of different power and several MKP test cases of different sizes and difficulties (tight and loose constraints). The performance on the response time is measured by two indexes, Speedup and Efficiency when is compared to a serial version of an Ant System. The results obtained show the potential power of exploiting the parallelism underlying in an Ant System regarding the good quality of the results and a remarkable decreasing on the computation time.

The remainder of the paper is organized in the following way. In the next sections the classical and adapted version of an Ant System are given. Next, two approaches for task assignment in a distributed system, the experiments performed and results obtained are shown. Finally, the conclusions are exposed.
2 A Brief Description of an Ant System

An Ant System (AS)[2,3,7,8,9] is a new meta-heuristic for hard combinatorial optimization problems. This meta-heuristic is a new member in the class of meta-heuristic derived from nature[4] that includes Genetic Algorithms, Neural Networks, Simulated Annealing, Evolution Strategies, etc. AS is an approach based on the result of low-level interaction among many cooperating simple agents that are not aware of their cooperative behavior[7]. Each simple agent is called ant and the Ant System (a distributed algorithm) is a set of ants working independently and cooperating sporadically in a common problem solving activity. Regarding the above characteristics, an Ant System can be naturally considered as a nearly embarrassingly parallel computation[14]. Since earlier applications of Ant Systems[7], plenty of work has been done in this area by applying Ant Systems to solve ordering problems like Traveling Salesman Problem (TSP), Bin Packing Problem and Quadratic Assignment Problem [2,3,4,7,8,9,11]. In [5] an Ant System was adapted in order to solve non-ordering or subset problems. The adapted AS[5] shown to be efficient to solve MKP. In this section, the original AS for solving TSP (an ordering problem) and the adapted AS for solving MKP (a subset problem) are presented.

2.1 The Ant System for Ordering Problems

Given a set of \( n \) cities, the Traveling Salesman Problem [12,13] is to find a closed path that visits every city exactly once (tour) with minimal total length. i.e.

\[
\text{minimize } \text{COST}(i_1, \ldots, i_n) = \sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_n}, C_{i_1})
\]

where \( d(C_x, C_y) \) is the distance between city \( x \) and city \( y \).

The Ant-cycle approach for solving TSP proposed in [7] is briefly presented here. Let \( b_i(t) (i = 1, \ldots, n) \) be the number of ants in city \( i \) at time \( t \) and let \( N\alpha = \sum_{i=1}^{n} b_i(t) \) be the total number of ants in the system. Let \( \tau_{ij}(t+n) \) be the intensity of trial on path \( \tau_{ij} \) at time \( t+n \), given by

\[
\tau_{ij}(t+n) = (\rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n)) \quad (2.1.1)
\]

where \( \rho \) is such \((1 - \rho) \) is the coefficient of evaporation \((0 < \rho < 1)\).

\[
\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{N\alpha} \Delta\tau_{ij}^k(t, t+n), \quad \text{where } \Delta\tau_{ij}^k(t, t+n) \text{ is the quantity per unit of length of trial substance (pheromone in real ants) laid on } \tau_{ij} \text{ by the } k^{th} \text{ ant between time } t \text{ and } t+n \text{ and is given by the following formula:}
\]

\[
\Delta\tau_{ij}^k(t, t+n) = \left\{ \begin{array}{ll}
\frac{Q}{L_k} & \text{if } k^{th} \text{ ant uses edge } (i, j) \text{ in its tour} \\
0 & \text{otherwise}
\end{array} \right. \quad (2.1.2)
\]

where \( Q \) is a constant and \( L_k \) is the tour length of the \( k^{th} \) ant. The intensity of trial at time 0, \( \tau_{ij}(0) \), is set to a randomly chosen value.

During the next \((t+n) \) tour the probability to visit city \( j \) when being at city \( i \) is

\[
P_{ij}(t, k) = \left\{ \begin{array}{ll}
\frac{\tau_{ij}^k(t)\eta_{ij}^k}{\sum_{h \in \text{allowed}_k} \tau_{ih}^k(t)\eta_{ih}^k} & \text{if } j \in \text{allowed}_k \\
0 & \text{otherwise}
\end{array} \right. \quad (2.1.3)
\]

where \( \text{allowed}_k \) is a set of cities not visited for that particular tour and \( \eta_{ij} \) is a local heuristic.

For TSP the parameter \( \eta_{ij} \), called visibility, is \( \frac{1}{d(C_i, C_j)} \) [7].
The parameters $\alpha$ and $\beta$ allow control on the relative importance of trail versus visibility. Hence, the transition probability is a trade-off between visibility, which says that close cities should be chosen with high probability, and trail intensity, that says that if on path $ij$ there is a lot of traffic then it is highly profitable.

A data structure, called tabu list, is associated to each ant in order to avoid that ants visit a city more than once, i.e. tabu list maintain a set of visited cities up to time $t$ by the $k^{th}$ ant. Therefore allowed$_k$ set can be defined as follows: allowed$_k = \{j| j \notin \text{tabu}_k\}$. When a tour is completed the tabu$_k$ list ($k = 1..Na$) is emptied and every ant is free again to choose an alternative tour for the next cycle.

By using the above definitions, we describe the Ant-cycle algorithm:

**Initialize**

for $t=1$ to number of cycles do
  for $k=1$ to Na do
    Repeat Until $k$ has completed a tour
      - Select city $j$ to be visited next with probability $P_{ij}$ given by equation (2.1.3)
    end
    Calculate the length $L_k$ of the tour generated by ant $k$
  end
  Save the best solution so far
  Update the trail levels on all paths according to equation (2.1.1)
end
Print the best solution found

### 2.2 The Ant System for Subset Problems

The Multiple Knapsack Problem which is an example of a subset problem can be formulated [1,12,13] as follows:

$$\text{maximise } \sum_{j=1}^{n} p_j x_j$$

subject to $\sum_{j=1}^{n} r_{ij} x_j \leq c_i \quad i = 1,\ldots,m \quad (2.2.1)$

$$x_i \in \{0,1\} \quad j = 1, \ldots n$$

Each of the $m$ constrains described is called a knapsack constrain, so the MKP is also called the $m$-dimensional Knapsack Problem. Let $I = \{1,\ldots,m\}$ and $J = \{1,\ldots,n\}$, with $c_i \geq 0$ for all $i \in I$, $j \in J$. A well-stated MKP assumes that $p_j > 0$ and $r_{ij} \leq c_i \leq \sum_{j=1}^{n} r_{ij}$ for all $i \in I$, $j \in J$, since any violation of these conditions will result in some $x_j$ being fixed to zero and/or some constrains being eliminated. Note that the $(r_{ij})_{mxn}$ matrix and $(c_i)_m$ vector are both non-negative which distinguishes this problem from general 0-1 linear integer programming problem. Many practical problems can be formulated as a MKP, for example, the capital budgeting problem where project $j$ has profit $p_j$ and consumes $r_{ij}$ units of resource $i$. The goal is to find a subset of the $n$ projects such that the total profit is maximized and all resource constrains are satisfied. For solving MKP, the ants[5] look for a subset of $n$ items or projects (see MKP formulation) such that the total profit is maximized and all resource constrains are satisfied.

Let $b_i (i = 1,\ldots,n)$ be the number of ants incorporating in the solution the item $i$ at time $t = 0$ and let $Na = \sum_{i=1}^{n} b_i$ be the total number of ants in the system. Since in MKP there are not paths, the intensity of trial and local heuristic are computed in a slightly different way. Let $\tau_i(t + N_{\text{max}})$ be the intensity of trial on item $i$ at time $t + N_{\text{max}}$, given by

$$\tau_i(t + N_{\text{max}}) = \rho \tau_i(t) + \Delta \tau_i(t, t + N_{\text{max}}) \quad (2.2.2)$$

where $\rho$ is such $(1 - \rho)$ is the coefficient of evaporation and $N_{\text{max}}$ is the maximum number of items qualified to be added to some solution by some ant.
\[ \Delta \tau_i(t, t + N_{\text{max}}) = \sum_{i=1}^{N_a} \Delta \tau_i^k (t, t + N_{\text{max}}), \]

where \( \Delta \tau_i^k (t, t + N_{\text{max}}) \) is the quantity per unit of length of trial substance (pheromone in real ants) laid on item \( i \) by the \( k^{th} \) ant between time \( t \) and \( t + N_{\text{max}} \) and is given by the following formula:

\[
\Delta \tau_i^k (t, t + n) = \begin{cases} \frac{L_k}{Q} & \text{if } k^{th} \text{ ant incorporates item } i \\ 0 & \text{otherwise} \end{cases} \tag{2.2.3}
\]

where \( Q \) is a constant and \( L_k \) is the profit (objective function in [Eq. 2.2.1]) obtained by the \( k^{th} \) ant. The intensity of trial at time 0, \( \tau_i(0) \), is set to a randomly chosen value. During the next \( (t + N_{\text{max}}) \) item incorporation the probability for selecting item \( i \) by the \( k^{th} \) ant, in order to complete the solution \( k \) is:

\[
P_i(t, k) = \begin{cases} \frac{\tau_i^* (t) \eta_j^k (k)}{\sum_{j \in \text{allowed}_k} \tau_j^* (t) \eta_j^k (k)} & i \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{2.2.4}
\]

where \( \text{allowed}_k \) is a set of items still not considered by the \( k^{th} \) ant and the solution \( k \) satisfies all constraints if some of them are added. The parameter \( \eta_i(k) \), called pseudo-utility, is the local heuristic. We chose \( \eta_i(k) \) as follows:

\[
\eta_i(k) = \frac{p_i}{\delta_i(k)}; \quad \bar{\eta}_i (k) = \frac{\sum_{j=1}^{m} \delta_{ij} (k)}{m} \\
\delta_{ij} (k) = \frac{r_{ij}}{(c_j - u_{ij}(k))}; \quad u_{ij}(k) = \sum_{t \in \text{solution}_k} r_{jt}
\tag{2.2.5}
\]

Where \( (c_j - u_{j}(k)) \) is the remaining amount to reach the boundary of constraint \( j \), \( r_{ji} \leq (c_j - u_{j}(k)) \) and \( \delta_{ij} (k) \in [0, 1] \), is the tightness of item \( i \) on constraint \( j \) when item \( i \) is added to solution \( k \). Consequently the pseudo-utility \( \eta_i(k) \) turns larger as \( \bar{\eta}_i (k) \) (tightness average) turns smaller. The parameters \( \alpha \) and \( \beta \), as for TSP, allow control on the relative importance of trail versus the local heuristic (pseudo-utility for MKP). Hence, the transition probability is a trade-off between pseudo-utility, which says that more profitable items that uses less resources should be chosen with high probability, and trail intensity, that says that if item \( i \) is part of a lot of solutions, then it is highly desirable.

A data structure, called tabu list, is also associated to each ant in order to avoid that ants choose an item more than once, i.e. tabu \( k \) list maintain the set of added items up to time \( t \) by the \( k^{th} \) ant. This list also maintains \( u_{ij}(k) \) \( (j = 1..m) \) in order to reduce the required computational time. The \( \text{allowed}_k \) set can be defined as follows:

\[ \text{allowed}_k = \{ j / j \notin \text{tabu}_k \text{ and solution}_k \text{ with item } j \text{ added satisfies all constraints} \} \]

When all ants add to the solutions as many items as they can, tabu \( k \) list \( (k = 1..N_a) \) is emptied and every ant is free again to choose an alternative subset of items for the next cycle.
The outline of the adapted Ant-cycle algorithm for subset problems follows:

\[\text{Initialize}\]
\[\text{for } t=1 \text{ to number of cycles do}\]
\[\text{for } k=1 \text{ to } N_a \text{ do}\]
  \[\text{Repeat Until allowed}_k \text{ is empty}\]
  \[\text{- Select item } i \text{ to be incorporated with probability } P_i \text{ given by equation (2.2.4)}\]
  \[\text{end}\]
\[\text{Calculate } L_k, \text{ the profit obtained by ant } k\]
\[\text{Save the best solution so far}\]
\[\text{end}\]
\[\text{Update the trail levels } \tau_i \text{ on all items according to equation (2.2.2)}\]
\[\text{end}\]
\[\text{Print the best solution found}\]

3 Parallel Ant System for solving MKP

An Ant System is a distributed algorithm where multiple independent agents cooperate with each other for solving a common problem. The algorithm runs for a fixed number of cycles, after each cycle the agents interchange some information (cooperate) in order to \textit{learn} which is the most promising area of the search space. During each cycle, all agents can execute independently since no interaction between them is needed at all. Based on the above features, two approaches, \textit{static and dynamic task assignment}\cite{14} were considered in order to accomplish a Parallel Ant System (a nearly embarrassingly parallel program). In the \textit{static task assignment} approach, the problem is divided into a fixed number of processes to be executed in parallel. In addition, the processes are simply distributed among the available processors without any discussion on the effects of the types of processors and their speeds. However, it may be that some of processors will complete their task before others and became idle because the work is unevenly divided or some processors operate faster than others (or both situations). On the other hand, the \textit{dynamic task assignment} approach intend to spread the tasks evenly across the processors in order to maximize the efficiency. Under both of the implemented approaches, there exist a \textbf{Master} and \(p\) \textbf{Slaves} processes, all of them distributed on available processors (\textbf{Figure 1}) in the system. The master process is in charge of updating the trail \((\tau)\) according the solutions found by the slave processes after each cycle of the algorithm. Every slave process is capable to separately access the instance of MKP to be tested.

![Figure 1: Layout for Static and Dynamic Task Assignment Approaches](image-url)
3.1 Static Task Assignment

Assuming that the Ant System involves Na ants and there exist p processors available in the system, Na/p ants are assigned to each processor. The master and slave processes and a brief explanation concerning the purpose of each message are outlined as follows:

**Messages**

*INIT_PROCESS*: - The Ant system parameters  
  - Request for the first solutions (Na/p)
*NEW*: - Modified trail (cooperation stage)  
  - Request for (Na/p) solutions
*END_PROCESS*: - Finish the slave

**Master Process**

Initialize p slaves
Send to every slave: *INIT_PROCESS*

do
{  
  Receive one solution from every slave    
  Update Δτᵢ regarding p received solutions    
  Choose the Best Solution from the p best solutions    
  Save the best solution so far    
  if (Ec < MaxCycles) then /* Ec stands for Elapsed cycles */    
    Update τᵢ(t, t + Nmax) (i : 1..Na) [Eq. 2.2.2]    
    Send to every slave: *NEW*    
    Ec++    
  else    
    Send to every slave: *END_PROCESS*    
    Print out the best solution found    
    Exit    
  endif
} while (*FOREVER*)

**Slave Process**

do {Receive order from Master
  switch (order) {
    case *INIT_PROCESS*:
      - Recover the Ant system parameters    
      - read_MKP_instance();    
      - Generate Na/p solutions    
      - Send back the best out of Na/p solutions
    case *NEW*:
      - Recover new trail    
      - Generate Na/p solutions    
      - Send back the best out of Na/p solutions
    case *END_PROCESS*:
      - Finish the slave
  }
} while (*FOREVER*)
3.2 Dynamic Task Assignment (Work-Pool)

By this approach, the master process maintains a Work Pool of agents to be processed (Centralized Dynamic Load Balancing)[14]. Thus, the Work Pool represents the inactive agents waiting for an idle processor. The master and slave processes and a brief explanation concerning the purpose of each message are outlined as follows:

Messages

INIT_PROCESS: - The Ant system parameters
   - Request for the first solution
PROCESS_SOL: - Solution request
NEW_τ: - Modified trail (cooperation stage)
END_PROCESS: - Finish the slave

Master Process
Initialize Work Pool with Na ants (tasks)
Initialize p slaves /* Each slave processes one ant */
Send to every slave: INIT_PROCESS

\[
\text{do} \{ \\
\text{Working Slaves} = \{1, \ldots, p\} /* Set of slaves still processing */ \\
\text{while (Working Slaves is not empty)} \\
\{ \\
\text{Receive a solution (from some slave, say } k) \\
\text{Update } \Delta \tau \text{ regarding the Received Solution} \\
\text{Choose the best solution so far} \\
\text{if (Work Pool is empty)} \\
\text{Working Slaves} = \text{Working Slaves} - \{k\} \\
\text{else} \\
\text{Send to slave } k: \text{PROCESS_SOL /*activate agent } i \text{ from Work Pool */} \\
\text{Delete agent } i \text{ from Work Pool} \\
\text{endif} \\
\} \\
\text{if (Ec < MaxCycles) then /* Ec stands for Elapsed cycles */} \\
\text{Update } \tau_i(t, t + N_{max}) (i:1..Na) \text{ [Eq. 2.2.2]} \\
\text{Initialize Work Pool with } Na - p \text{ ants (tasks)} \\
\text{Send to every slave: NEW_τ} \\
\text{Ec++} \\
\text{else} \\
\text{Send to every slave: END_PROCESS} \\
\text{Print Out the Best Solution so far} \\
\text{Exit} \\
\text{endif} \\
\} \text{ while(FOREVER)}
\]

Slave Process

\[
\text{do} \{ \\
\text{Receive order from Master} \\
\text{switch (order) } \{ \\
\text{case INIT_PROCESS:} \\
\text{ - Recover the Ant system parameters}
\}
\]
4 Computational Study

Six instances of MKP taken from [1] were considered in our experiments. The Ant Parallel Systems were tested, at the begining, on two processors $P_1$ and $P_2$ (Sun Sparc workstations having a similar capacity) by using the optimal parameter setting found in [5]. Table I shows the results of our experiments expressed in terms of the average of Speedup Factor (regarding a Serial Ant System running on processor $P_1$), Efficiency (Eqs. 4.1)[14] and Percentage of Hits out of 10 runs for each instance.

\[
\text{Speedup}(n) = \frac{T_s}{T_n}
\]
\[
T_s = \text{Execution time using a single processor system}
\]
\[
T_n = \text{Execution time using a system with } n \text{ processors}
\]

\[
\text{Efficiency}(n) = \frac{\text{Speedup}(n)}{n}
\]
\[
\text{i.e the fraction of time the processors are being used on the computation}
\]

Columns PAS-S and PAS-D stand for Static and Dynamic Task Assignment in a Parallel Ant System respectively.

<table>
<thead>
<tr>
<th>Instance</th>
<th>PAS-S</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Speedup</td>
<td>Efficiency</td>
<td>%hits</td>
<td>Speedup</td>
<td>Efficiency</td>
<td>%hits</td>
</tr>
<tr>
<td>1</td>
<td>1.81</td>
<td>0.905</td>
<td>80</td>
<td>1.70</td>
<td>0.85</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>1.63</td>
<td>0.815</td>
<td>70</td>
<td>1.67</td>
<td>0.835</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>1.77</td>
<td>0.885</td>
<td>100</td>
<td>1.65</td>
<td>0.825</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>1.84</td>
<td>0.92</td>
<td>70</td>
<td>1.6</td>
<td>0.8</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>1.68</td>
<td>0.84</td>
<td>90</td>
<td>1.72</td>
<td>0.86</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>1.89</td>
<td>0.945</td>
<td>100</td>
<td>1.8</td>
<td>0.9</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table I.** PASs running on processors $P_1$ and $P_2$

As we observe in **Table I**, both approaches showed similar behaviour. The parallel systems performed very well regarding the Speedup, Efficiency and additionally, the quality of the results (%hits) compared to those results found in [5,6,10]. However, an additional study was necessary in order to establish some difference between PAS-S and PAS-D approaches. In a second experiment, a third processor $P_3$ (slower than $P_1$ and $P_2$) was incorporated to the Parallel Virtual
Machine. Table II shows the Speedup obtained by the two approaches. It is important remarking that the Serial Ant System was run on processor \( P_1 \) (one of the faster processors). Columns \( PVM_i = \{P_j\} \) stand for the set of processors conforming the Parallel Virtual Machine.

The values in Table II indicate that PAS-D performed much better than PAS-S running on both environments, \( PVM_1 \) and \( PVM_2 \) respectively. Although the the work is evenly divided, the relative velocities of each processor are no considered by PAS-S approach. For example, PAS-S obtained for each instance considered a Speedup less than 1 running on environment \( PVM_1 \) since processor \( P_3 \) is the "bottle neck" of the system and turning \( P_1 \) idle most part of the time. On the other hand, PAS-D took advantage of the Work Pool of tasks when some processor turns idle. A similar situation is observed in \( PVM_2 \) where the inclusion of processor \( P_2 \) (the other faster processor) produced only a little improvement on the Speedup obtained by PAS-S.

<table>
<thead>
<tr>
<th>Instance</th>
<th>( PVM_1 = {P_1, P_3} )</th>
<th>( PVM_2 = {P_1, P_3, P_2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PAS-S</td>
<td>PAS-D</td>
</tr>
<tr>
<td>1</td>
<td>0.66</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td>1.37</td>
</tr>
<tr>
<td>3</td>
<td>0.7</td>
<td>1.56</td>
</tr>
<tr>
<td>4</td>
<td>0.72</td>
<td>1.33</td>
</tr>
<tr>
<td>5</td>
<td>0.76</td>
<td>1.44</td>
</tr>
<tr>
<td>6</td>
<td>0.7</td>
<td>1.41</td>
</tr>
</tbody>
</table>

Table II. Values for environments \( PVM_1 \) and \( PVM_2 \)

It is remarkable that for this kind of highly distributed algorithm (an Ant System) the profit achieved by using PAS-D approach, is evident. However, PAS-S also achieve a good performance running on an environment of processors having a similar power (see Table I).

Although the results obtained show clearly the difference between PAS-S and PAS-D running on different environments, it is not evident to carry out a straightforward analysis of the performance of the parallel ant systems, either PAS-S or PAS-D, due to they are stochastic algorithms where their computation time strongly depend on the seed given as input to generate random numbers. For example, in image processing, a task can be divided in a small number of tasks and each one processes a fixed number of pixels (some image partition). On the other hand, an Ant System is a set of small tasks (independent agents) conforming a distributed algorithm and the purpose is to distribute those small tasks on available processors in a particular parallel platform. However, when any parallel approach is applied to some instance of MKP, the Speedup achieved by augmenting the number of processors varied slightly (varying the seed) due to the variation on the number of items incorporated by each ant per cycle of the algorithm which performs accordingly the initial seed. It is also possible that for some instances of MKP, the parallel systems running on an heterogeneous platform are able to obtain a Speedup Factor very close to the optimal one.

5 Conclusions

An Ant System is a class of distributed algorithm which can be naturally considered as a nearly embarrassingly parallel computation. The two proposed approaches showed that the explicit parallelism involved in an Ant System can be easily exploited by using networked workstations. However, there exist some considerations to take in account when a particular approach will be used: Work division and processors power conforming the parallel environment. Also, it is worth remarking that the applications developed in PVM are portable enough to run on different parallel platform without major changes.
References

[1] Beasley, J. - "OR-Library: Distributing Test Problems by Electronic Mail" - e-mail: o.rlibrary@ic.ac.uk


