Chapter-6

AN ADAPTIVE HYBRID STRATEGY OF TASK
PARTITIONING, SCHEDULING, AND LOAD BALANCING
FOR DISTRIBUTED RAYTRACING SYSTEM

INTRODUCTION

It is well known that the raytracing rendering approach has generated some of the highest quality images. The research on Photo-Realistic rendering intends to investigate efficient task partitioning, scheduling and load balancing strategies for distributed raytracing system. High quality computer graphics images (photo-realistic) are used today in a wide variety of fields. The best-known technique for the synthesis of photo-realistic images is raytracing.

Ray tracing is one of many techniques for rendering three-dimensional images. Ray tracing is a rendering model, which is based on the physics of light and how it interacts with different materials. Ray tracing gets its name from the use of simulated rays of light in producing images. In ray tracing, visible surface determination, reflection, refraction, and shading are done using physically based approximations of the way real light behaves. Ray tracing is capable of rendering complex mathematical surfaces, multi-dimensional vector fields, and discrete polygonal meshes. Classical sequential ray tracing algorithms can be adapted for parallel execution environments, and are well suited for the production of photorealistic images. Other rendering techniques that are computationally cheaper are typically limited to the use of polygonal meshes for modeling objects, and can be much more difficult to parallelize efficiently. Ray tracing is well suited to parallel computation. By parallelizing the rendering process, execution time can be reduced by more than two orders of magnitude, given appropriate computational resources.

Rendering artificial scenes in a physically correct manner requires a computationally very expensive lighting simulation. In practice, and using today’s processing power, such simulations may take between couples of hours to several days. The goal of real-time rendering appears not yet achievable using current single processor technology. Under simple lighting conditions and for scenes that are relatively simple to render, parallel-processing solutions can now achieve interactive frame-rates. Hence, multi-processor solutions provide an opportunity to make rendering more practical. The creation of realistic three-dimensional images is one of the most powerful applications of computer vision/graphics. The ray tracing has become one of the most popular approaches to creating synthetic images [1]. Ray tracing is useful when we want to view a realistic image of something that either: We do not want to make a physical model due to cost, size, or other constraints; or we do not have direct access the real situation. Such kinds of situation may arise in advertising (product promotions and logos, for example), architecture (including virtual reality), art, Computer Aided Design, especially in the mechanical field, entertainment (scientific movies, graphics) and etc. Such types of image computing applications require very large amounts of floating point
calculations, comparable to the requirements of the largest problems in computational sciences and engineering. Moreover, often computation requirements and application data size crosses the limit of single workstation/personal computer. It has been realized that massively parallel computers can be an effective way to accelerate these computations [2, 3].

Two types of parallelism are generally used when generating ray-traced images. The first is the object base image partitioning and second is the pixel/horizontal scan line base image partitioning. The first (data parallel) approach requires a possibly large amount of communication during raytracing, because information about other subsets of objects is needed to perform the computation. In the second (task parallel) approach, each processor/worker must have the copy of complete scene description, so that it can access all the information about the rays involved in and determining the color of a pixel. Nevertheless, the second approach is an ideal candidate for achieving high speedups, but it is better than first one [4]. Therefore, it is used in this investigation.

Due to the advancements of commodity based distributed computing environment has made it possible to process such kind of images in less time on less budgets. Efficient partitioning and scheduling are necessary for heterogeneous distributed computing system for better utilization of the computational power of all processors/remote machines. In general, the purpose of partitioning operation is to split the task evenly among all processors/workers. Whereas the approach of efficient task scheduling and load balancing are to ensure that no machine stay idle until whole application is completed.

In PDP system, the static and dynamic (RTS) task distribution strategies are used to balance the loads among the WSs/PCs. Since WSs/PCs has a performance variation characteristic [5, 6], therefore, the static task distribution is not effective for PDP system [7, 8]. RTS strategy can achieve nearly perfect load balancing [9], because of the machines' performance variations and non-homogeneous nature of the application (image) is adjusted at runtime [10]. The RTS strategy performance depends upon the size of the sub-task. If sub-task size is too small then it generates a serious inter-process communication overhead. The challenge in parallel ray tracing is to find algorithms, which allow large scenes to be distributed without losing too much efficiency due to load imbalances (data parallel) or communication (demand driven). Combining data parallel and demand driven aspects into a single hybrid-scheduling algorithm is one such approach. Hybrid scheduling algorithms have both demand driven and data parallel components running on the same set of processors.

In other case, if the sub-task size is too large then it may create a longer master (client) machine waiting time due to the inappropriate sub-tasks size of slow performance machine [10]. Many researchers suggested enhancement in dynamic task scheduling strategies for homogeneous PDP system [11-13]. However, these strategies do not work well for heterogeneous PDP system without further modifications. The crucial point is that they are based on fixed parameters that are tuned for the specified hardware. In heterogeneous systems, this tuning is often not possible because both the computational power and the network
bandwidth are not known in advance, which may change unpredictably during runtime.

The strategies based on task distribution and then task migration from heavy loaded to light loaded workers are expressed [14, 5]. The task migration has two serious drawbacks [15]:

- All workers should continuously monitor the status of other workers.
- During the computations, a worker has to search its load and float the information on the network, hence, produces a large amount of communication overhead.

In contrast to all above, we proposed a hybrid strategy with several enhancements, i.e. the AHTS strategy has adaptive nature; it reduces inter-process communication time overhead; and effectively balances the load among the workers.

6.1. PREVIOUS WORK

For the last ten years, task partitioning, scheduling, and load balancing problems for homogeneous parallel-distributed systems have been thoroughly studied. However, these topics are relatively new for heterogeneous PDP platforms and have been investigated less frequently. The eight different task partition strategies for images homogeneous PDP system were presented in [16] and its improved work is presented in [17]. In both papers, the authors suggested a best partitioning and scheduling strategy for distributed ray-tracing system:

The authors suggested to partition the whole image into two parts as upper and lower half of the image. The upper half is evenly distributed among all worker processes. The lower half will be dynamically distributed to the workers, as worker becomes idle.

The authors' image partitioning is unadaptive and he has not clarified that how much part of the image is fixed for upper and lower partition. If upper partition is more, in that case, the serious load imbalance will occur at the end of the application due to bigger sub-task size of low performance machine. If lower partition is larger, then the number of requests to be performed by the master will increase, that will cause the overhead of inter-process communication time. They also have not expressed the load balancing mechanism in their proposed strategies, which we are presenting in this chapter.

This work is extended work of [18] in which authors studied two adaptive task partitioning & scheduling strategies and reached to the same conclusion as [16, 17]. The main features of AHST are: it's adaptive task sizes according to the machines' performance; it avoids the master machine waiting time at the end of the application and it reduces the inter-process communication time.

This chapter is organized as follows. In section 6.2, we described the theory of our investigated strategies. The strategies performance evaluation terms and
experimental setups explained in section 6.3. Section 6.4 contains the obtained results and discussions. The conclusion of this chapter and future research directions are given in section 6.5 and in section 6.6 we presented summary of this chapter.

6.2. TASK PARTITIONING AND SCHEDULING

The efficient execution of parallel program on PDP system depends on the effective partitioning of the program into modules (or sub-partitions) and scheduling of these modules. The goal of the program partitioning is to minimize the overhead caused by inter-task communication time, while preserving the highest possible degree of parallelism. Here we describe the investigated strategies.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Total number of tasks (640 horizontal scan-lines of the image).</td>
</tr>
<tr>
<td>$i$</td>
<td>Worker number.</td>
</tr>
<tr>
<td>$t_i$</td>
<td>Worker's assigned tasks or computed sub-task size.</td>
</tr>
<tr>
<td>$T_h$</td>
<td>Total number of unprocessed tasks ($T_h = T - t_i$)</td>
</tr>
<tr>
<td>$T_d$</td>
<td>Total number of tasks distributed.</td>
</tr>
<tr>
<td>$t_p$</td>
<td>Total number of tasks in partitioned sub portion.</td>
</tr>
<tr>
<td>$W$</td>
<td>Worker’s estimated current performance.</td>
</tr>
<tr>
<td>$W_{pi}$</td>
<td>Worker's performance at unloaded condition.</td>
</tr>
<tr>
<td>$W_{qul}$</td>
<td>Worker’s job queue length at unloaded condition.</td>
</tr>
<tr>
<td>$W_{qi}$</td>
<td>Worker’s current job queue length.</td>
</tr>
<tr>
<td>$W_i$</td>
<td>Total number of workers in PDP configuration.</td>
</tr>
<tr>
<td>$W_{up}$</td>
<td>Sum of all workers' estimated performance.</td>
</tr>
<tr>
<td>$W^{dpi}$</td>
<td>Worker’s normalized performance.</td>
</tr>
</tbody>
</table>

Table 1. Symbols used in strategies and their definitions

6.2.1. Static task partitioning and scheduling strategy

The static tasks distribution strategies' chief advantage is their simplicity; there is no need to collect current system state information. Their disadvantage is that they cannot respond to changes in system states, so the performance improvement they provide is limited [15].

Since the heterogeneous ODP system is composed of WSS/PCS of different performance machines, therefore, equal task partitioning is not feasible for such type of system. The optimum task distribution (ODT) rule is used to divide the task optimally at compile time using the pre-knowledge of workers' performance. The machines' performances are measured by well-known Unix byte
benchmark [http://g.biokemi.su.se/~arne/benchmarks]. The ODT rule is defined as below:

\[ W_{pi} = \frac{1}{P_i} \]  

(1)

where \( P_i \) is the processing time taken by the worker \( i \) to process a specified program (benchmark program).

Sum of all workers estimated performance:

\[ W_{pdp} = \sum_{i=1}^{n} W_{pi} \]  

(2)

The worker's normalize performance is computed:

\[ W_{npi} = \frac{W_{pi}}{W_{pdp}} \]  

(3)

Optimum distribution rule (ODT) is used to divide the task among all workers in a PDP system:

\[ t_i = T \cdot W_{npi} \]  

(4)

### 6.2.2. Runtime task scheduling strategy

A unit of task (fix one horizontal scan-line of the image) is distributed at runtime. As the worker completes the previous assigned sub-task, new task is assigned for processing. The task mapping mechanism of static and RTS can easily be differentiated by viewing Fig. 1.

An example image

![An example image](image1.png)

a) Image partitioning by ODT rule in static strategy among four workers  

b) Image partitioning in RTS strategy

Fig. 1. Tasks mapping among four workers: a) Static strategy, b) RTS strategy.
6.2.3 Adaptive hybrid task partitioning, scheduling, and load balancing (AHTS) strategy

It is the mixture of static and RTS strategies. Researchers use to monitor the machines’ performances in a heterogeneous PDP system using the benchmark process. Since due to the non-homogeneous nature of application (image), the worker’s performance estimated from worker’s runtime responses never gives the accurate worker’s performance [19] (for an example see Fig. 8, the processing complexity of image A). The authors [20, 21] clearly indicated that the number of CPU job queues indices length approach for load balancing gives better result.

<table>
<thead>
<tr>
<th>WS/PC No.</th>
<th>Machine (vendor) / CPU / Clock (MHz)</th>
<th>RAM (MB) / Cache (KB)</th>
<th>Operating System</th>
<th>Measured average performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC 1</td>
<td>GP6-450 (Gateway) / Pentium-II / 450 MHz</td>
<td>64 MB / 512KB</td>
<td>FreeBSD 3.0</td>
<td>48.2</td>
</tr>
<tr>
<td>PC 2</td>
<td>ViP6400/BX-ATX (VP) / Pentium-II / 400 MHz</td>
<td>64 MB / 512KB</td>
<td>FreeBSD 3.1</td>
<td>34.7</td>
</tr>
<tr>
<td>PC 3</td>
<td>ViP6400/BX-ATX (VP) / Pentium-II / 333 MHz</td>
<td>64 MB / 512KB</td>
<td>Redhat 5.0</td>
<td>33.7</td>
</tr>
<tr>
<td>PC 4</td>
<td>GP6-333 (Gateway) / Pentium-II / 333 MHz</td>
<td>128 MB / 512 KB</td>
<td>FreeBSD 2.6</td>
<td>33.0</td>
</tr>
<tr>
<td>PC 5</td>
<td>Scratch-K6-2/350 (Wave) / K6-2 / 350 MHz</td>
<td>64 MB / 512 KB</td>
<td>FreeBSD 2.28</td>
<td>28.0</td>
</tr>
<tr>
<td>PC 6</td>
<td>Pro-UWS (Frontier) / Pentium Pro / 200 MHz</td>
<td>136 MB / 256 KB</td>
<td>FreeBSD 3.0</td>
<td>27.3</td>
</tr>
<tr>
<td>PC 7</td>
<td>Scratch-K6-2/300 (Wave) / K6-2 / 300 MHz</td>
<td>64 MB / 512 KB</td>
<td>Linux 2.0.32</td>
<td>25.7</td>
</tr>
<tr>
<td>PC 8</td>
<td>Millennia XRU (Micron) / Pentium-II / 266 MHz</td>
<td>96 MB / 512 KB</td>
<td>Solaris 2.6</td>
<td>25.2</td>
</tr>
<tr>
<td>PC 9</td>
<td>WIS-K6-3D/300 (NetBank) / K6-2/300 MHz</td>
<td>64 MB / 1024 KB</td>
<td>FreeBSD 3.0</td>
<td>25.2</td>
</tr>
<tr>
<td>PC 10</td>
<td>Scratch-K6-2/400 (Wave) / K6 / 266 MHz</td>
<td>64 MB / 512 KB</td>
<td>FreeBSD 2.27</td>
<td>20.5</td>
</tr>
<tr>
<td>PC 11</td>
<td>JM5516DT-ATX+ (Proside) / K6 / 200 MHz</td>
<td>96 MB / 512 KB</td>
<td>FreeBSD 2.28</td>
<td>17.1</td>
</tr>
<tr>
<td>WS 12</td>
<td>AS4085/HS21 (Toshiba) / HyperSparc / 125 MHz</td>
<td>64 MB / 256 KB</td>
<td>Solaris 2.5</td>
<td>12.9</td>
</tr>
<tr>
<td>WS 13</td>
<td>AS4075CS+ (Toshiba) / TurboSparc / 170 MHz</td>
<td>64 MB / 512 KB</td>
<td>SunOS 4.1.4</td>
<td>12.6</td>
</tr>
<tr>
<td>PC 14</td>
<td>JP4 (JCC) / PowerPC604 / 100 MHz</td>
<td>64 MB / 256 KB</td>
<td>4.4BSD</td>
<td>9.7</td>
</tr>
<tr>
<td>WS 15</td>
<td>JS5/70 (JCC) / microSparcII / 70 MHz</td>
<td>72 MB / 256 KB</td>
<td>Solaris 2.5</td>
<td>7.9</td>
</tr>
<tr>
<td>PC 16</td>
<td>J3100PV2/466+ (Toshiba) / Pentium ODP / 83 MHz</td>
<td>48 MB / 256 KB</td>
<td>Linux 2.0.33</td>
<td>6.8</td>
</tr>
<tr>
<td>PC 17</td>
<td>J3100PV2/466+ (Toshiba) / Pentium ODP / 83 MHz</td>
<td>32 MB / 256 KB</td>
<td>FreeBSD 2.2.0</td>
<td>6.0</td>
</tr>
<tr>
<td>WS 18</td>
<td>S-4JX (Fujitsu) / Sparc / 40 MHz</td>
<td>52 MB / 64 KB</td>
<td>LinuxSPARC</td>
<td>5.2</td>
</tr>
<tr>
<td>WS 19</td>
<td>AS4015 (Toshiba) / microSparc / 50 MHz</td>
<td>48 MB / 0 KB</td>
<td>SunOS 4.1.3</td>
<td>4.0</td>
</tr>
<tr>
<td>WS 20</td>
<td>AS4015 (Toshiba) / microSparc / 50 MHz</td>
<td>24 MB / 0 KB</td>
<td>SunOS 4.1.3</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 2. PDP system connected WSs/PCs specification and their measured performance
In our proposed strategy, we use the CPU jobs queue length information for estimating the worker’s performance that is used for task partitioning and load balancing.

The pictorial of task partitioning under AHTS strategy is illustrated in Fig. 2.

![Image](an_example_image)

Fig. 2. Image partitioning in AHTS strategy

In this strategy, the whole task is partitioned into ten equal portions. The manager has pre-knowledge of each worker’s performance and its job queue lengths’ information at unloaded condition (no one user was logged in the network). According to pre-measured worker’s performance, the first partitioned sub-portion of the task (image) is again partitioned using the OTD rule and scheduled among the workers of PDP system. Whenever the worker returns the result it also returns the worker’s current job queue length information. Manager uses this information for estimating the worker’s performance (using Eq. 5) with reference to its noted performance and queue length at unloaded condition. Note that, each worker’s relative performance and sub-task size (tasks assigned to the worker) is modified if the worker’s performance ratio = \( \frac{\text{Current}}{\text{Previous}} \) or vice versa becomes 0.25, because the rapidly fluctuating task sizing will never give the better results for PDP system [22].

\[
W_{pi} = \frac{W_{pi} \times W_{qi}}{W_{pri}} \quad (5)
\]

The estimated worker’s performance is also used to compute each worker’s normalized performance (using Eq. 3) and the portion of task to be distributed among the workers according to OTD rule (\( t_i = t_r \times W_{nqi} \)). Remaining seven portions are partitioned and scheduled according to the same principle. To achieve a better load balancing and avoid master machine waiting time at the end of the application, we include following components in this strategy for better load balancing:
• Every worker is allowed to assign a unit (tiny) of sub-task up to four times, if all workers still did not completes their previous assigned task. If in case, still all workers did not complete their previous task, the next sub-task size \( (\ell_i) \) of the idle worker is modified from its relative performance not by ODT rule. This logic reduces the number of request made by faster worker and effectively reduces the worker’s idle time cost overhead.

• After 70% of the total task (T) completion, manager checks those workers, which still did not complete the assigned task from the first partitioned portion. The manager marks that worker “as not available” in the current PDP configuration and adds that assigned task \( (\ell_i) \) to the unprocessed task balance \( (T_b) \). It eliminates serious manager waiting time that occurs at the end of the application.

• The manager reduces the sub-task size \( (\ell^r) \) of each worker by 30% on each worker’s request when 70% of the total task \( (\ell_i) \) is processed. This reduces the work load-imbalance and manager waiting time that occurs at the end of the application.

• When all tasks \( (T_d = T) \) are distributed and some faster workers may completes the assigned last task quickly but other worker could not complete the task. In that case, the manager searches the least performance worker in the current PDP configuration and duplicates its assigned task to the next idle worker. This logic creates a competition between two workers, the worker who completes the task first his result will be considered and other will be ignored. Each least performance worker task in the current PDP configuration is duplicated only once. Therefore, all computing resources will utilize at the end of the application. This also eliminates serious manager waiting time that may occur at the end of the application.

Fig. 3 illustrates the expected worker’s sub-task sizing behavior in AHTS strategy. The tiny task is used to adjust the difference of performances of the workers. This behavior is much close to the proposed theoretical approach, which we discussed in this section.
Fig. 3. Sub-task size modification of a worker at unloaded condition.

![Graph showing sub-task size modification](image)

**Manager/Master**

![Communication channel](image)

Fig. 4. Network computing model

### 6.2.4. Short view of AHTP strategy

The computing model of our strategy consists of a manager process and a collection of worker processes as depicted in Fig. 4. The manager is responsible for partitioning a given image into a set of several sub-images (sub-portions) and further, each sub-portion of an image partitioned according to worker’s performance (OTD rule). However, for few last sub-portions each request of worker’s task size reduces. So all the workers try to finish their assigned task with a minimum time difference.

Below are the working sequences of manager process:

- Using OTD rule and workers’ pre-measured performances distribute the first sub-portion of the image among the worker.
• Collect the results (pixel of image) and current jobs queue information from the workers.

• Based on runtime job queue information of worker and worker's job queue information at unloaded condition of the worker's current performance is estimated.

• A tiny (unit of task) task is assigned to the faster worker up to four times to compensate the sub-task completion time difference among the workers.

• Partition and parallelizes the other sub-portions of the image among the workers according to OTD rule.

• Omit the slow-performance worker (who still did not complete a single task) after 70% of the total tasks completion.

• Not even a single worker is kept idle until full application (image) is processed (the current slow worker's task is duplicity assigned to the idle worker).

6.3 IMPLEMENTATION

The performance of static, RTS and AHTS strategies are mainly evaluated in terms of speedup and workers idle time cost [23]. Here we briefly define these terms.

6.3.1. Speedup

The speedup is used to quantify the performance gain from a parallel computation \( T_{polp} \) of an application over its computation \( T_{pf} \) on a single fastest machine in a heterogeneous PDP system. It is defined as follows,

\[
\text{Speedup of PDP system (SP)} = \frac{T_{pf}}{T_{polp}} \quad (6)
\]

6.3.2 Workers idle time cost

The activities of worker's process in a PDP image-processing system is mainly composed of three factors that are:

• Worker setup time to load pattern data file and initialize all programming parameters.
• Worker ray tracing computation time
• Worker time taken to report result (data) and getting new task from master.
The value of the third factor is directly affected when number of requests made by the worker increases. Practically we compute this overhead time as:

\[ O_i = (\text{Worker's previous task completion time}) - (\text{worker's new task starting time}) \quad (7) \]

Where \( O_i \) is the overhead time, which includes all communications data access delays for starting the processing of new task on idle worker.

The workers idle time cost can be expressed as:

\[ \text{Workers idle time cost} = \sum_{i=1}^{W} O_i \quad (8) \]

Remote Procedure Call (RPC) allows a client (master) to execute procedures on other network computers (workers). RPC forms the foundation for most of the distributed system utilities used today like network file server and network information server. It supports the heterogeneity of processors and provides a transparent communication interface between the client and server.

In RPC client/server setup, the client sends out a request over the network. The default UNIX service daemons of each machine continuously monitor service request from other remote machines. When a request is received, it invokes the service. The services requested machine (master machine) is inactive between the times of the request and when it receives a reply. Blocking the client/master machine process is not acceptable in case when multiple servers/workers are to be involved for computation. One way to avoid this blocking and to control the multiple workers at the master end is to use the UNIX built-in facilities of parent/child process creation (using the fork () system call) that we used in our PDP system.

For our distributed raytracing application, we used UNIX parent/child processes parallelism technique for controlling the multiple workers. In this technique, the client (master) creates a number of child processes, which is equal to the number of workers used in PDP configuration. The master machine uses child processes to request jobs on remote workers. Each child process of the associated workers waits patiently for their reply, till it gets the signal from specified worker. The client machine collects the data (pixel) from the specified worker then assigns a new task. This process continues till all the tasks are processed.

6.3.3. Parallel Distributed Processing System

The heterogeneous PDP system used in our investigation is composed of seven Sun workstations (WSs) loaded with SunOS/Solaris, and thirteen PCs (Intel based machines), loaded with Linux/FreeBSD operating system; all of these machines are connected via Ethernet. The WSs/PCs specification and their measured performance listing are shown in Table 2. The configuration in which the PDP system's WSs/PCs are connected is shown in Fig. 5.
The network communication is handled by *Open Consortium Remote Procedure Call (ONC-RPC)* library and XDR filters. UNIX heavy weight processing technique is used to control many workers at the manager's end. Following are the sequence of steps in distributed image computing application:

- Establish master/workers communication.
- Database and scene parameter distribution for each pixel in display
- Master assigns scan-line of the image for computation.
- Worker performs intersection shadow, and shading computations.
- Worker returns pixel value to master.

<table>
<thead>
<tr>
<th>Image scene A</th>
<th>Image scene B</th>
<th>Image scene C</th>
<th>Image scene D</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 sec</td>
<td>122 sec</td>
<td>152 sec</td>
<td>244 sec</td>
</tr>
</tbody>
</table>

Table 3. The single fastest machine in the network image scenes processing time

![Diagram of network connection configuration](image_url)

Fig. 5. PDP system WSs/PCs network connection configuration.

The investigation is carried out on a ray tracing image because it can be parallelized without complex inter-processors communication. To analyze the performance of the proposed AHTS strategy, we used four parallel-distributed ray
tracing images that are known as scene A, B, C, & D and each image is composed of 840 x 640 pixels. Fig. 6 gives the impressions of the images. The five PDP configurations are set, i.e., the PDP system is composed of number of worker (NW) = 20, 16, 12, 8 & 4. These PDP configurations are configured in such a way that (starting from the maximum workers, i.e., NW=20) each next PDP configuration is arranged by omitting the four slowest workers from the current configuration.

Fig. 6. Raytracing images: a) scene A, b) scene B, c) scene C, and d) scene D.

6.4. RESULTS AND DISCUSSIONS

The proposed strategy (AHTS) performance is compared in terms of speedup, number of request made by the master and workers idle time cost and finally its performance is compared with static and RTS strategies. The investigated PDP configurations have high heterogeneity in machine’s performance; some machines have a speed of 400Mhz and other has 40Mhz (see Table 2). The processing time taken to process the image scenes A, B, C & D by single fastest machine in the network are shown in Table 3. From Table 3, it is clear that image scene A is lightly dense and takes less processing time and scene D is the heaviest among all scenes. The aim to include the static and RTS strategies only to clearly evaluate the comparative improvement in AHTS strategy. All measurements are carried out at condition when no one user is logged in the network.

6.4.1. Static strategy

The measured obtained speedups, listed in Table 5, are the worst as compared to RTS and AHTS strategies. This is due to two main reasons:

The tasks are distributed without knowing the non-homogeneous nature of the application (image) and whole task (image) is completely divided among the working workers at compile time. Because of that some workers complete the assigned tasks quickly, where other may take longer time due to heavy processing requirements of the assigned task (the non-homogeneous nature of an example image A can be realized from Fig. 7). If dense processing required task assigned to the slow performance worker, it may create a long master waiting time at the end of the application. The master waiting time definition can easily be understood from Fig. 8.
The machine's performance has dynamic variation characteristic that cannot be accurately estimated at compile time [6].

Due to the above-described reasons, in most of the cases it is observed that the slowest machine causes a long master waiting time at the end of the application. The measured master waiting time for images A and D are shown in Fig. 9. From Fig. 9, it can be realized that the dense application has more master waiting time as compared to less dense application. The 25% and 46% average (approximate) degradations are found in measured speedups in all PDP configurations with respect to RTS and AHTS strategies, respectively.

There are two idle features in static strategy:

- Since the tasks is completely distributed among the workers at compile time so there is no workers idle time cost exists in this strategy.
- The master machine makes less number of requests.

![Fig. 7. Image scene A scan-line processing time histogram.](image)

![Fig. 8. Master machine waiting time mechanism](image)
6.4.2. Runtime Task Scheduling (RTS) strategy

The RTS strategy has a potential to absorb the machine’s performance variation characteristics and non-homogeneous nature of the application at runtime tasks assignment. The number of tasks processed by the worker is proportional to its runtime performance. However, RTS strategy has two serious drawbacks that as explained in section 6.2. We evaluated the performance of RTS strategy using the unit of task (one horizontal scan-line of the image). Due to the small task size, all workers were utilized at the end of the application and very less master waiting time is generated in all PDP configurations say for example less than 1 second. The measured speedups of this strategy are listed in Table 5. The speedup results show that the RTS has speedups improvements of 25% average (approximate) in all PDP configurations with respect to static strategy. Due to the large number of master’s requests occurred in RTS strategy (see Table 4), the high overhead of workers idle time cost is generated (see Fig. 10). The average measured workers idle time cost increases as the number of workers increases in PDP configurations, which is due to the following reasons:

As the number of workers increases the waiting child processes increases at the master, which increases the auxiliary workload at master machine, therefore, it may decrease its task parallelization capability and effectively increase the worker’s waiting time.

As the number of workers increases in master and workers PDP system model, the low bandwidth network usage increases, because every worker have a responsibility to report the results to the master. Therefore, this creates extra load on the network, which may be the cause of long workers idle time cost.

For all PDP configurations, 45% average (approximate) workers idle time cost increases as compared to AHST strategy. It is our aim to minimize this workers idle time cost in AHST strategy.

6.4.3. Adaptive Hybrid (AHTS) strategy

As explained in section 6.2, the whole image is partitioned into number of sub-portions. Each sub-portion is parallelized to number of workers according to worker’s current performance and each worker’s sub-task size is reduced in last partitioned portions of the image. In this way all computing resources are utilized at the end of the application with very less master waiting time.

Further more, the strategy has many good load-balancing features (see section 6.2), due to that, its performance dramatically improves. From Table 5, for all PDP configurations, the measured speedup of this strategy has performance improvement on RTS and as well as static. Their speedups improvements are 46% and 28% average (approximate) in all PDP configurations with respect to static and RTS strategies, respectively. Due to adaptively workers sub-task sizing the total number of requests made by the master under this strategy is also reduce in all PDP configurations (see Table. 3) as compared to RTS strategy. Due to the same above
reason, the measured workers’ idle time cost is reduced to 45% average approximate in all PDP configurations as compared to RTS strategy (see Fig. 10).

![Bar chart: Master waiting time in image A vs. Master waiting time in image D.](image)

Fig. 9. In static task scheduling strategy the obtained master machine waiting time in images A and D.

<table>
<thead>
<tr>
<th>PDP system configuration (NW)</th>
<th>Static strategy</th>
<th>RTS strategy</th>
<th>AHTS strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>640</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>640</td>
<td>92</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>640</td>
<td>118</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>640</td>
<td>159</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>640</td>
<td>202</td>
</tr>
</tbody>
</table>

Table 4. Comparison of static, RTS and AHTS strategies in term of number of requests made by master (client) machine.

![Bar chart: RTS strategy vs. AHTS strategy.](image)

Fig. 10. Measured average workers’ idle time cost (sec) in five PPD configurations obtained in RTS and AHTS strategies.
6.5. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this chapter, we studied the performance of static, RTS and hybrid task partitioning and scheduling strategies for heterogeneous PDP system. Investigation is carried out on a ray tracing images because its task-parallelization constitutes a parallel nature, similar kind of nature is found in parallel medical imaging, remote sensing, large matrices computation, and etc.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.2 1.7 1.8 1.8</td>
<td>2.2 2.5 2.5 2.8</td>
<td>3.1 3.5 3.6 3.8</td>
</tr>
<tr>
<td>8</td>
<td>1.4 2.1 2.2 2.6</td>
<td>3.0 3.2 3.4 4.1</td>
<td>3.9 4.5 4.9 5.9</td>
</tr>
<tr>
<td>12</td>
<td>1.5 2.3 2.5 3.2</td>
<td>3.5 4.0 4.0 4.7</td>
<td>4.8 5.5 5.6 6.7</td>
</tr>
<tr>
<td>16</td>
<td>1.6 2.4 2.9 3.6</td>
<td>3.7 4.3 4.5 4.9</td>
<td>5.2 6.0 6.1 7.1</td>
</tr>
<tr>
<td>20</td>
<td>1.8 2.8 3.4 4.0</td>
<td>4.2 4.8 5.0 5.4</td>
<td>5.7 6.5 6.9 7.6</td>
</tr>
</tbody>
</table>

Table 5. Measured speedup for five PDP configurations in static, RTS, and AHTS strategies

The above results show that the static and RTS are inefficient for heterogeneous PDP image computing system. The adaptive hybrid (AHTS) approach of tasks partitioning, scheduling, and load balancing remedy the defects of static and RTS strategies.

The AHTS strategy is simple and its management overheads are not significant as compared to its performance. The AHTS strategy has 46% and 28% approximate speedup improvement over static and RTS strategies respectively. The workers idle time cost is also reduced to average 45% (approximate) in all PDP configurations as compared to RTS strategy.

For evaluating the performance of AHTS by partitioning image into 10 sub-portions, we got good results for this ray tracing application. However, for other applications some modification may be needed.

The AHTS strategy has a centralized task scheduler and strategy is tested on maximum 20 workstations/personal computers. If the PDP system becomes very large (hundreds of WSs/PCs), in that case, our proposed strategy may be less effective. We are thinking to design an efficient distributed task scheduling strategy to reduce the inter-process communication cost, to satisfy the requirements of large PDP system and give high speedup.
In this chapter we studied the construction and performance of a distributed application that renders synthetic images using the ray-tracing paradigm. Remote procedure calling, RPC, is applied in multiplicity to the computationally demanding task of ray tracing. Heterogeneous parallel distributed processing systems, which exploit the aggregate power of a network of workstations (WSs) and Personal Computers (PCs) are an inexpensive alternative to the dedicated parallel supercomputing systems. As these systems are widely available in academic and industrial environments, it is becoming popular to use these computing resources to solve time-consuming applications. The main problem with such type of cluster computing environment is the continuous change in the performance of individual WSs/PCs which requires an efficient task partitioning, scheduling, and load balancing to get better performance. In this chapter we investigated the problem of static and runtime tasks allocation for parallel-distributed ray tracing system. The Parallel Distributed Processing systems have heterogeneity in processors and resources. For such system, we propose an Adaptive Hybrid Task partitioning, Scheduling (AHTS), and load balancing strategy. The investigation is examined on a manager/master and workers model of Parallel Distributed Processing systems. The measured results show that, the AHTS strategy dramatically improves the performance of Parallel Distributed Processing systems raytracing computing system and remedy the defects of static and Runtime Task Scheduling (RTS) strategies.
REFERENCES


