Scheduling Real-time Graphics Geometry Operations in Multicomputer Systems

Felicia Ionescu
Simultec S.A.
Platforma Magurele
P.O.B. 24, Bucharest
ROMANIA

Abstract: Proper exploitation of the potential power of parallel computers requires an efficient solution to tasks-to-processors scheduling problem. This paper discusses the scheduling problem of real-time graphics geometry operations to be executed on a multicomputer system. Assuming a run-time scheduling approach, a pre-scheduling operation will be defined as a cluster partitioning of processes with a view at minimising the parallel execution time on a virtual architecture with unbounded number of processors. Such partitioning compensates for the finding of a mapping that minimises a cost function in a large configuration space, defined as p^v configurations (where p is the number of processors and v is the number of processes), with a problem of dynamically mapping a reduced number of clusters onto processors.
A weighted program dependence graph is used for representing the program for the scheduling problem and a heuristic clustering algorithm reduces the granularity of the program, preserving the whole embedded parallelism. For load balancing of the processors, dynamic mapping of clusters onto physical processors in the network is implemented as a receiver initiated distribution of computing work.

Keywords: Real-time image synthesis, Multiple Instruction Stream, Multiple Data Stream Architecture, program dependence graph, cluster partitioning, dynamic mapping, scheduling, parallel overhead.

Dr. Felicia Ionescu received the Applied Electronics degree and Ph.D degree in Microelectronics from the Department of Electronics and Telecommunications, the Polytechnical Institute of Bucharest, Romania. She is now R&D Manager at Research Institute for Simulators, "Simultec S.A.". Romania, and is dealing with parallel processing, real -time image generation and development software for visual database generation.

1. Introduction

Synthetic images are obtained by a sequence of graphics operations executed on object models, structured and stored in a graphical database. For each object, the image synthesis system performs two sequences of operations [9], i.e.:

- *graphics geometry operations* which consist in coordinate transformation of all vertices from object coordinates into view coordinates, clipping transformation of each surface into a 6-plane bounding box describing what region is visible to the viewer, the perspective division for all vertices of clipped surfaces, that takes a perspective view of the resulting object, and compresses the 3D object into a 2D coordinate space for viewing on the screen.
- *rendering operations* which consist in a scan-conversion transformation that computes all pixels that belong to object 2D screen coordinate surfaces and their color, writing the resulting values on the image memory.

This large scale, complex problem of image synthesis can be solved by combining a number of processors and memory blocks in a network environment. The architecture class of the projected parallel computer is a MIMD distributed memory network with message passing communications between nodes. This network is structured into two functional subnetworks: a geometry subsystem which performs graphics geometry operations, and produces a set of graphics primitives, and a rendering subsystem which performs rendering operations of these graphics primitives. These subsystems are connected by a flow network, for transfer messages containing graphics primitives.

This structure allows good exploitation of different types of parallelism at each stage of processing [6].

In a rendering subsystem, operations provide the data parallelism characteristics: many data items (pixels of the image) are liable to identical processing, which allows a perfect domain decomposition into processes which require little or no communication at all with one another. This leads to a SPMD (Single Program, Multiple Data) operational mode: each processor executes the same program asynchronously, on its own subdomain of data [7].

In the geometry subsystem, graphics geometry operations can be described as a number of asynchronous processes, which are scheduled into processors of the network. Scheduling algorithms are more complex for multicomputer systems than for uniprocessor systems, since the scheduling algorithm must not only indicate an ordering of processes, but also it must determine a specific processor to be used. A number of researchers succeeded to prove that finding an optimal schedule for a set of real-time tasks in a multicomputer system is NP-complete. Therefore, research efforts have been directed to the development of suitable heuristic scheduling algorithms capable of finding a suboptimal solution in reasonable time [1], [2], [3],[10], [11].

This paper presents a hybrid scheduling strategy for real-time graphics geometry operations in a distributed memory network as a two -step approach: a static cluster partitioning of the processes described by a program dependence graph, and a dynamic mapping of clusters onto physical processors of the network, with a view at obtaining a minimum parallel execution time.

2. Strategy for Scheduling Graphics Geometry Operations

The implementation program of graphics geometry operations into an image synthesis system is a complex one, where a large number of processes which can be executed in parallel by different processors of the network, is identified. In order to benefit from this parallelism, processes should be scheduled over as many processors as possible. However, the overhead of communications among distributed processes may outweigh the increased performance reached thanks to the use of multiple processors. Therefore, scheduling strategies need determine the best trade-off between parallelism and overhead.

The geometry subsystem is a parallel computer with distributed memory and message passing, and is represented as a graph R = (P, E), where:

$$P = \{ p_i \mid 0 \le i < p, p = |P| \};$$
$$E = \{e_{i,j} = (p_i, p_j) \mid p_i, p_j \in P \}.$$

P is the set of nodes, each node representing a processor in the network.

E is the set of communication links between processors.

The parallel program which implements graphics geometry operations is represented by means of a directed acyclic graph, the *program dependence graph* (PDG), which is a popular and general representation of *control and data dependences* in a parallel program. A PDG node represents a process in program, i.e. a basic computation block; a directed edge in a PDG represents control or data dependence. PDGs represent the ideal program parallelism, and are useful for solving a variety of problems, including optimisation, vectorisation, detection and management of parallelism. Therefore, PDG is a natural option for program representation in the scheduling problem.

If physical communication costs on the parallel computer links can be quantified then, by using them in conjunction with the PDG, a weighted PDG can be defined to represent the program as a set of processes with known (or estimated) computation time for every node, and explicit dependences expressed by weighted edges in the graph. Formally, a weighted PDG is a tuple G = (V,D,T,C), where:

$$V = \{v_i \mid 0 \le j < v, v = |V| \};$$
$$D = \{d_{i,j} = (v_i, v_j) \mid v_i, v_j \in V\};$$
$$T = \{t_i \mid 0 \le i < v \};$$
$$C = \{c_{i,j} \mid 0 \le i,j < v \}.$$

V is the set of nodes, each node representing a process in the program.

D is the set of directed communication edges between processes. A directed edge $d_{i,j} = (v_i, v_j)$ expresses the dependence between processes v_i and v_j, which can be either control dependence or data dependence. Control dependence of a process v_j towards process v_i means that the execution of the process v_j can start only after the termination of the process v_i. Data dependence of a process v_j towards process v_i means that at least one output variable of the process v_i is input variable for the process v_j (Bernstein's conditions). Two processes can be executed in parallel if there exist no control or data dependences between them, and, given this, the execution yields the same results, regardless of whether they are executed sequentially in any order or in parallel.

T is the set of node computation costs. The value $t_i \in T$ is the computation cost for node $v_i \in V$.

C is the set of edge communication costs. The value $c_{i,j} \in C$ is the communication cost incurred along the edge $d_{i,j} \in D$, which is zero if both nodes are mapped in the same processor.

With this formulation, *a schedule* is defined by a processor assignment mapping of the nodes of G onto the **p** processors of network R, and by the starting time of all nodes.

The *degree* of parallelism in a PDG, also called *width* of the PDG, is the size of the maximal set of independent processes. The *length* of a path is the summation of all node computation and edge communication costs in the path. The *critical path* is the path with the longest length in the PDG. Given a parallel program that is characterised by a weighted PDG, the scheduling of the processes to processors can be done either statically (before program execution), or dynamically, in an adaptive manner (as the parallel program is executed).

Static approach is more attractive, as the scheduling computation needs only be performed once, but this approach can only be used only if the parallel program is static, i.e. the processes are executed on a regular data structure and the nodes and edges weights can be accurately estimated a priori. This is not the case with graphics geometry operations which start with the database traversal for finding visible graphical objects. The database traversal dynamically generates an irregular search space, depending on the current viewpoint [5]. Therefore, a dynamic scheduling approach must be considered for graphics geometry operations.

For a weighted PDG a cost function can be formulated to evaluate a particular mapping of the processes onto the processors. The scheduling problem can thus be formulated as a problem of finding a mapping that minimises this mathematical cost function, and, indeed many researches were devoted to this aspect. The primary problem with most of the approaches based on explicit minimisation of a cost function is its exponentially large configuration space of possible mapping that must be selectively searched in trying to optimise the function. The flat and unstructured view of the scheduling as a mapping of each process onto a processor leads to an unmanageable large search space. If the program dependence graph has **v** nodes and the parallel computer has **p** processors, the space of possible mappings has $\mathbf{p}^{\mathrm{v}}$ configurations.

We have proposed a hybrid approach to scheduling strategy that combines a static partitioning of the program dependence graph with a dynamic mapping. This results in a two-step approach of the scheduling problem:

(1) The first step is a *static partitioning of the PDG into clusters*: pairs of processes which require much communication are grouped together and mapped on the same processor in order to reduce the communication costs. Thus the process-to-processor mapping problem may be viewed instead as a problem of formatting clusters of processes with high intra-cluster communication and low inter-cluster communication, and of allocating these clusters to processors in a way that results in low inter-processor communication costs. The view of mapping in terms of formatting clusters of closely coupled processes helps reduce significantly the space of mapping that is selectively searched for a satisfactory solution. The clustering problem can be formulated as a problem of finding the minimum of a cost function.

(2) The second step is a *dynamic mapping of clusters onto physical processors* which allows workload balancing and results in low parallel overhead.

Cluster partitioning of a PDG is going to determine a mapping of nodes of G onto m clusters $\{K_0, K_1, \ldots K_{m-1}\}$ aiming at minimising a cost function, which, for real time image synthesis, is the parallel execution time, on an unbounded number of processors. This is known as min-max criterion for clustering,

$$\min_{\Pi} \{ \max_{0 \leq j \leq s} T(v_j) \}$$

where Π is the set of all possible cluster partitioning on an unbounded number of processors and $T(v_j)$ is the computation time of node j. The clustering that scores the minimum parallel execution time is called *optimum clustering*.

A clustering is called *nonlinear* if at least one cluster contains two independent processes, otherwise it is called *linear*. Linear clustering fully exploits the natural parallelism of a given PDG, while nonlinear clustering sequentializes independent processes to reduce parallelism.

The property of linear clustering as to preserve the whole parallelism of a given PDG, makes it a suitable approach for the first step of the proposed scheduling strategy. The PDG is statically partitioned into linear clusters and, if the execution of the PGD using linear clustering reaches the optimal time, then the granularity of the PDG is appropriate for the given architecture; otherwise it is to fine and a scheduling algorithm needs execute independent processes together in the same processor, using a nonlinear clustering strategy.

Dynamic mapping of clusters onto physical processors for workload balancing represents a nonlinear clustering and the goal of these two steps, i. e. static linear clustering and dynamic nonlinear clustering, is the minimisation of parallel execution time.

3. Program Dependence Graph of Graphics Geometry Operations

At this step the database traversal operation is considered as a single process, and all other graphics geometry operations are analysed for determining the program dependence graph.

Database Traversal process (DT) has a number s of successors - processes which extract visible objects for the given viewpoint, (EO_1, EO_2, ...EO_s). The dependence between DT process and EO_1, EO_2, ...EO_s processes can be control dependence or data dependence according to the modality of storing the database in the network. If the database is stored in a single node of the network, then EO_1, EO_2, ...EO_s processes must wait for a set of data which describes the object to be generated. This will result in large communication needs and serious limitation of the performances by increased communication costs. Therefore a replicated database solution has been adopted: in every node of the network there is stored the whole database and, in this case the dependence between DT process and extracting processes is a control dependence with a lower level of communication.

Every extracting process (EO_1, EO_2, ...EO_s) is followed by a sequence of processes which allows image generation of the object. These processes are:

TV process - which applies a coordinate transformation for all vertices of an object from object coordinates to view coordinates, using a 4x4 matrix stack.

LV process - process to light vertices to improve the viewing of surface contours and shapes. The color is applied to each vertex as a function of the vertex position, the surface normal direction, the lighting model, the lights and the characteristics of the surface.

CL process - is a clipping process to a 6-plane bounding box describing what region is visible to the viewer. The vertices of the clipped surfaces go through a perspective division that provides a perspective view of the resulting object, and compresses the 3D object into a 2D coordinate space for viewing on the screen.

SD process - a surface decomposition process in the screen coordinate space which splits every surface with four or more vertices into multiple independent triangular pieces. The triangles - which are graphics primitives- are then handled

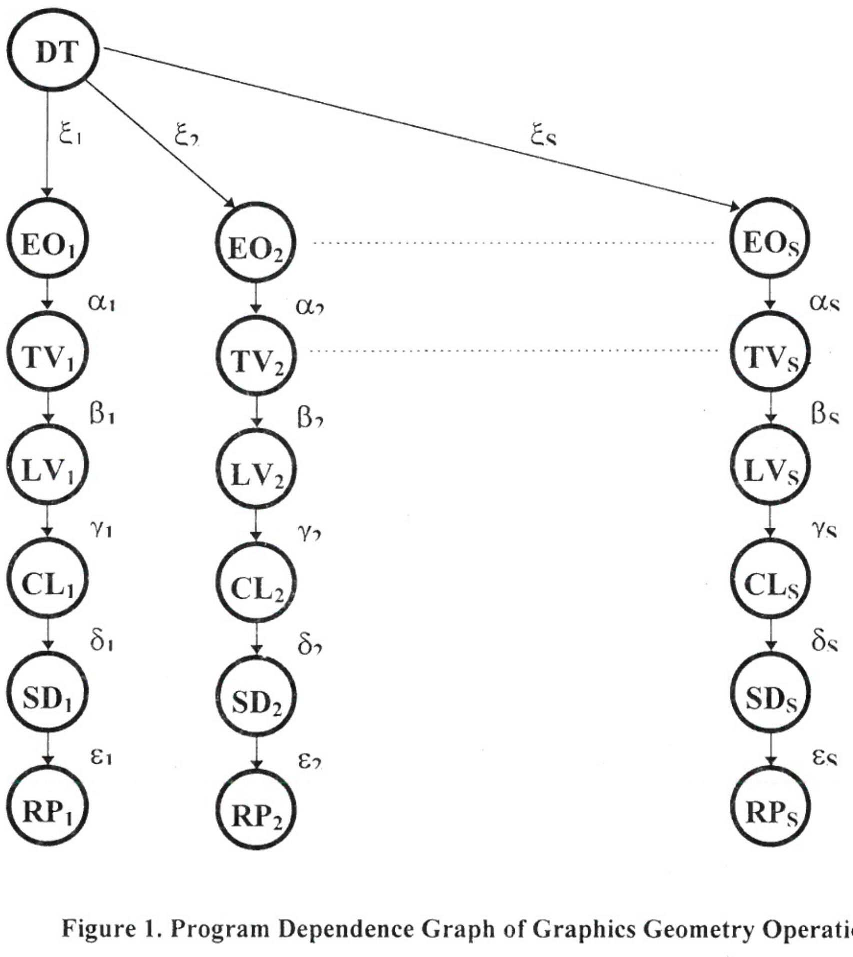

Figure 1. Program Dependence Graph of Graphics Geometry Operations (G)

identically through the remaining graphics operations.

RP process - is the final stage of calculation in the geometry subsystem and determines a number of rendering parameters of the triangles. These parameters are used by the rendering subsystem for scan-conversion graphics primitives. The output of RP processes is a set of data describing graphics primitives and this set is sent as a message to the rendering network.

For graphics geometry operations the program dependence graph G , presented in Figure 1, consists in a number of (6*s + 1) nodes, connected by communication directed edges, where **s** is the number of visible objects for the given viewpoint.

For weighting this PDG, nodes and edges weights are to be computed, based on the computational capacity of nodes and the communication capacity of edges and on the following assumptions:

(1) The computation costs of the nodes depend on the node process.

(2) The physical network contains identical processors, so the computation costs of PDG nodes do not change when a process is mapped in different nodes of the network.

(3) Edge communication costs increase linearly with the length of the message transferred between nodes.

In Figure 1 there are presented communication costs for every edge in PDG.

4. A Linear Clustering Algorithm

The first step of the proposed scheduling strategy of graphics geometry operations in a multicomputer system is a static linear cluster partitioning. The clustering algorithm consists in a number of steps, and each step tries to refine the previous clustering. Linear clustering of graph G is presented in Figure 2.

The initial step assumes that each process in PDG is a separate cluster. Each following step tries to merge two clusters by zeroing the edge that connects them. Zeroing an edge means that two end nodes of this edge are mapped into the

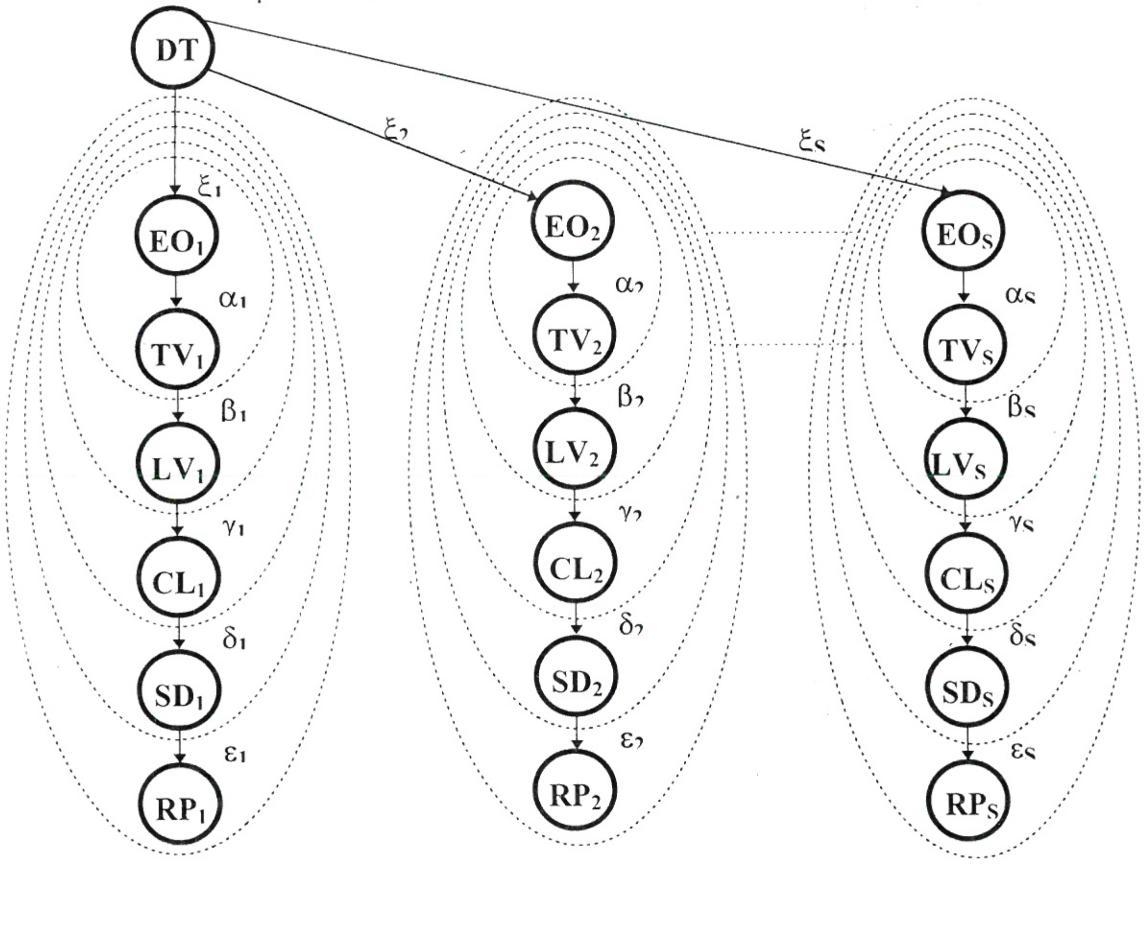

Figure 2. Linear Clustering in Graph G

same cluster and thus the communication costs become zero. Let T_{P_i} be the parallel execution time after the completion of step i , estimated according to the assumption that all processes in a cluster are executed on the same processor. T_{P0} is the length of the critical path including communication costs and computation costs in the initial graph where no edge has been zeroed.

For the clustering algorithm is used Gerasoulis's heuristics [4], " if the parallel execution time at step i , T_{P_i} , does not increase by zeroing the highest edge cost, then zero this edge". For this algorithm, the edges are first sorted by their costs, in a decreasing order. The algorithm performs maximum **d** steps ($d = |D|$), and, at each step, it tries to zero the selected edge, if the parallel execution time does not increase and the resulted cluster is linear.

For the PDG in Figure 1, at the first step each process is considered as a separate cluster, and the edges are sorted in a decreasing order by their communication costs. Let :

$\alpha_1 \geq \alpha_2 \geq \ldots \alpha_s \geq \beta_1 \geq \beta_2 \geq \ldots\beta_s, \ldots \geq \varepsilon_1 \geq \varepsilon_2 \geq \ldots \varepsilon_s \geq \xi_1 \geq \xi_2 \ldots \geq \xi_s$ be this order.

In the following s clustering steps, edges $\alpha_1, \alpha_2, \ldots \alpha_s$ are zeroed and, for every step, the parallel execution time is reduced. Clustering continues by zeroing edges $\beta_1, \beta_2 \ldots \beta_s, \ldots \varepsilon_1, \varepsilon_2, \ldots \varepsilon_s$ and in this way a linear clusters graph $G_k = \{K, H\}$ is obtained, as presented in Figure 3, where:

$K = \{K_0, K_1, \ldots K_s\}$, is the set of linear clusters;

$H = \{h_1, h_2, \ldots h_s\}$, is the set of dependence edges between linear clusters.

The cluster K_0 contains a database traversal process (DT), and each of the remaining **s** clusters, ($K_1, \ldots K_s$) contains all the processes needed for generating graphics primitives for a visible object. These clusters can be executed in a virtual architecture with an unbounded number of processors.

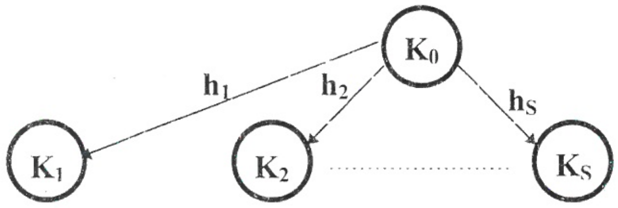

Figure 3. Linear Clusters Graph (G_k)

5. Dynamic Mapping of Clusters Onto Physical Processors

Given a linear clusters graph G_k , mapped onto a virtual network with an unbounded number of processors, the second step of scheduling can be defined as a mapping of every linear cluster onto a physical processor of the real network, with a view at workload balancing .

The network architecture was defined as a graph $R = (P, E)$. Mapping linear clusters onto $p = |P|$ processors is a function: $M:K \rightarrow P$, where $M(K_i)$ defines a processor **q** in which the cluster K_i is mapped. For mapping a number **s** of clusters onto **p** processors (and, most probably, $s > p$), a set of clusters CS_q, must be defined for each processor **q** ($0 \leq q < p$), where :

$$CS_q = \{ j \mid M(j) = q \} , q=0,1,2 \ldots p-1.$$

Each set CS_q is the set of linear clusters mapped onto processor **q** and represents a nonlinear cluster in PDG.

To determine these sets and, also the starting time of each linear cluster mapped onto a processor, a dynamic mapping of clusters to processors is used, every cluster representing a computing activity which is assigned to a processor in such a manner that every processor stays idle as little as possible.

If the number **s** of linear clusters, which represents **s** computing activities, is smaller than the number **p** of processors, then (p-s) processors will remain idle, which results in an inefficiency of the network. In this situation, the network parallelism is inefficiently exploited by the program parallelism, because the granularity of the program is coarser than the granularity of the network. The granularity of the linear clusters graph which allows a load balancing of the processors must provide a value $s > p$. For this, if the network granularity is defined (given a number **p** of processors), then the granularity of the program must be revised by changing the (increasing) number and the (decreasing) dimensions of objects in the database.

Database traversal process (DT) for extracting visible objects, assumed as a single process at the first step (linear clustering), can be assigned as a cluster in a single processor of the network, which results in a centralised database traversal, or in a decomposition of the number of processes and in the distribution of these processes to all processors of the network, that is a distributed database traversal.

A centralised database traversal can be viewed as a better approach, since it maintains global information about the state of the network and uses it for load balancing in dynamic mapping of computing activity. But this gathering of information in a single node can be prohibitive for large networks, due to the occurrence of multiple access requests, which causes communication delays. A distributed database traversal does away with this bottleneck, but adds overhead due to the communication between processors which perform this operation.

For load balancing of the network, a *receiver initiated distribution of computing work* [8] is implemented. Every processor can be in one of the two states:

- *idle*, when it has no work;
- *active*, when it performs some work.

The program starts with a single processor, say processor P_0, executing the database traversal, all the remaining processors being idle.

In a centralised database traversal, only processor P_0 performs this operation. Every idle processor sends a message to the processor P_0, as a work request, and waits for an answer from P_0 , as a message containing a handle to an object in the database. Then, the receiver processor becomes active, and executes a linear cluster of processes, for extracting the corresponding object from its replicated database, and for generating graphics primitives of the object.

This process continues until all visible objects in the database are distributed by processor P_0 to (p-1) processors of the network, and it represents a *dynamic mapping of linear clusters onto physical processors* which defines the scheduling of the graphics geometry operations. The starting time of execution of each linear cluster in a processor is dynamically determined by the moment in which a receiver processor receives an object handle from processor P_0 .

The distributed database traversal begins with assigning the whole database search space to a single processor, P_0; all the other processors have their search space null. As in the centralised database traversal, each processor can be idle, when it has no work, or active, when it does some work.

An idle processor selects a *donor* processor and sends a work request to it . If the idle processor receives some work (part of the database search space) from the donor processor, it becomes active. If it receives a reject message (because the donor has no work), it selects another processor and sends a work request to that donor.

An active processor, which has a part of the database search space, extracts all visible objects defined in this.space and initiates, for each object, the execution of a linear cluster of processes for generating graphics primitives.

In the distributed database traversal, the scheduling strategy provides a dynamic mapping of linear clusters onto physical processors and also a dynamic distribution of the database search space, which results in a global load balancing of the processors.

Dynamic mapping of clusters onto physical processors contributes an overhead function, due to communication for work requests and work distribution. This overhead function, computed for different network topologies, is smaller for a centralised database traversal, if the computational costs of the database traversal process (T_T) are lower than the amount of work performed by every other processor, i.e. $T_T < W / p$, where W is the total amount of work, and p is the number of processors.

This condition defines a maximum number p_c of processors in a geometry network for which the centralised database traversal can be efficiently used. For numbers of processors greater than p_c no higher parallel speed-up can be obtained by increasing the number of processors, because the parallel execution time is upper bounded by the computational costs of the database traversal process (T_T), and, in this case, only a distributed database traversal can ensure the scalability of the network.

6. Conclusions

We have presented the problem of scheduling real-time graphics geometry operations in an image synthesis system implemented as a multicomputer with distributed memory and message passing between nodes. We have presented a network model and a program representation as a weighted program dependence graph (PDG), for facilitating the scheduling work. Our approach to the scheduling problem consists in two steps: the first step is a static cluster partitioning of the PDG, with a heuristic clustering algorithm; the second step is a dynamic mapping of clusters onto processors, for obtaining a computing workload balancing. The dynamic mapping always distributes linear clusters which generate the graphics primitives of an object. The search space of the database traversal operation can also be dynamically distributed among multiple processors of the network, which results in a scalable image synthesis system.

REFERENCES

1. BACCELLI, F. and LIU, Z. , **On the Execution of Parallel Programs On Multiprocessors Systems - A Queueing Theory Approach**, JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, Vol.37, No. 2, April 1990.
2. BACCELLI, F., LIU, Z. and TOWSLEY, D., **Scheduling of Parallel Processing With and Without Real-Time Constraints**, JOURNAL OF THE ASSOCIATION FOR COMPUTING MACHINERY, Vol.40, No. 5, November 1993.
3. BURCHARD, J., LIEBENHERR, Y.O.H. and SON, S.H., **New Strategies for Assigning Real-Time Tasks To Multiprocessor System**. IEEE TRANSACTIONS ON COMPUTERS, Vol. 44, No. 12, December 1995.
4. GERASOULIS, A. and YANG, T., **On the Granularity and Clustering of Directed Acyclic Task Graphs**, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Vol.4, No. 6, June 1993.
5. IONESCU, F. and ALEXA, S.**, PC-Based Software System for Visual Data Base Creation**, International Training Equipment Conference and Exhibition Proceedings, the Hague, 1996.
6. IONESCU, F., **Global Optimisation for Parallelism and Locality in An Image Synthesis Parallel System**, International Semiconductor Conference - CAS - Sinaia, 1996.
7. IONESCU, F., **Mapping Image Rendering Operations Onto Parallel Processors,** International Semiconductor. Conference - CAS - Sinaia, 1996.
8. KUMAR, V., GRAMA, A., GUPTA, A.and KARYPS, G., **Introduction To Parallel Computing**, The BENJAMIN/CUMMINGS PUBLISHING COMPANY, Inc, Redwood City, 1994.
9. MAGNENAT- THALMAN, N. and THALMAN, D.**, Image Synthesis. Theory and Practice**, SPRINGER-VERLAG , Tokyo 1987.
10. SADAYAPPAN, P., ERCAL, F. and RAMANUJAM, J., **Cluster Partitioning Approaches To Mapping Parallel Programs Onto A Hypercube**, PARALLEL COMPUTING , No.13, 1990.
11. SONEOKA, T. and IBARAKI, T., **Logically Instantaneous Message Passing,** in Asynchronous Distributed Systems, IEEE TRANSACTIONS ON COMPUTERS, Vol. 43, No. 5, May 1994.