EXPLOITING PARALLELISM IN IMAGE SYNTHESIS APPLICATIONS

Gerald Leitner
Department of Computer Science
Columbia University, New York, NY 10027

ABSTRACT

The generation of digital images and computer animated sequences is a very computation-intensive process which can be partitioned into different phases. Logically, many phases can be executed concurrently. Appropriate environments for concurrent execution are a single-processor system with multiprogramming, or a multi-processor system, or a local area network. This paper addresses basic mechanisms to utilize concurrency. First, a structure called "production net" is introduced to specify the process of generating an image (or a sequence of frames) and the parallelism inherent in this process. Then, the execution of a production net in a distributed system is analyzed and a form of dynamic task distribution based on "contract nets" is proposed.

KEYWORDS: image synthesis, parallel processing, distributed application, production net, contract net.

SUMMARY

The generation of high-quality digital images and computer-animated films is a very computation-intensive application. As an alternative to using a Cray computer it is well worth exploiting parallelism to increase the speed and efficency of the image synthesis process.

Parallelism can be exploited in many different hardware environments. On a single-processor system, concurrent processes speed up the execution of an application if processes perform frequent I/O operations. This is definitely the case for image generation applications, because many functions involve I/O to a graphics processor (e.g. a frame buffer system), to disk (e.g. for large files in a paging system), or to a user terminal. In a multi-processor system, shared memory and high communication bandwidth permit tightly coupled concurrent execution. The development of "supercomputers" containing thousands or hundreds of thousands of processing nodes permits massive parallelism. Local area networks consisting of powerful workstations and fast communication lines are an especially interesting environment because of their widespread existence today.

In order to exploit the above environments, the programs for the generation of digital images and animated sequences have to be partitioned and executed as concurrent communicating processes. This partitioning can occur at different levels. At a low level, massive parallelism can be achieved by partitioning functions at the level of individual pixels or groups of pixels (e.g. for ray tracing algorithms) or at the level of individual polygons in a polygon-based scene database. At a medium level, different non-intersecting groups of objects in a scene can be processed separately (after the appropriate pre-processing, see e.g.[1]). At a high level, different planes are overlayed or otherwise processed to form the final image; in animated sequences, different frames or different parts of frames can be generated in parallel (see e.g.[2]). (Computer-generated frames in the film Star Trek II: The Wrath of Khan consist of a sky as a background image and of several layers - mountain ranges, a lake, etc. - sandwiched on top of it. The generation of some frames involved the use of up to 50 different programs and took up to five hours.[3])

Our research concentrates on two fundamental problems: (1) how to specify the parallelism inherent in an application; (2) how to execute the subtasks of a program efficently in a given hardware environment, i.e. how to map individual tasks to available resources.[4]

It is highly desirable to separate the specification of the parallelism from the details of the underlying system, because hardware environments change and programs should be

portable. We introduce the formal structure of a "production net" to specify the parallelism inherent in an image synthesis application. A production net is a directed graph whose nodes are "objects" and "modules". An object is a well-defined data structure that belongs to a specific "object class" and has a unique ID. Examples of objects are data structures defining groups of polygons, data structures defining viewing parameters, pixel arrays, etc. A module is a procedure activation which implements a partial function of the image synthesis application. A module takes instances of well-defined object classes as inputs and produces instances of well-defined object classes as output. Examples of modules are "transformation modules" which transform a 3D scene and given viewing parameters into a "rendered scene", or a module that overlays two pixel arrays, etc.

The nodes in the directed graph are connected by "arrows". An arrow from an object to a module means that the object is an input to a function; thus, it has to be generated before the function can be applied. An arrow from a module to an object means that the object is a result of executing the module function. The final result are the objects which have no outgoing arrows.

Such a production net constitutes an exact specification of how to create an image or a sequence of frames. The second part of our research is concerned with the distributed execution of production nets, i.e. with the distribution of tasks (= execution of modules) among available resources and the flow of objects through the system. The execution takes into consideration both the static configuration of the underlying system and dynamic (run-time) parameters, e.g. the current system load.

The execution of a production net is based on the concepts of "contract nets" and "goal-driven" processing. In the "contract net" paradigm[5], a "manager" process is responsible for the completion of a certain task. The manager can "contract" other processes for "subtasks". The connection between the "manager" process and the "contract" processes is established dynamically at run-time. Furthermore, this process is iterative, i.e. a "contract" process can become a "manager" process for its own subtask.

Tasks are initiated in a "goal-driven" order, where the "goal" is the creation of the final image or sequence of images. Let us assume that the final image is an object x. If x is the result of applying a module M to objects y and z, then the generation of y and z are subtasks to be contracted. By applying this principle recursively, the primitive already existing objects are reached, which will then be applied to modules to generate intermediate objects, etc., until the final image(s) are complete.

The major advantages of the system described above are the provision of a formal framework to specify the parallelism inherent in a task which is independent of the underlying system, and the efficient distributed execution of the specification. Production nets provide a high-level specification language which supports modular design and provides the basis for concurrent execution; the distributed execution of production nets take into consideration the actual system environment at run-time and optimizes its utilization.

REFERENCES

1. Crow, F.C., ``A More Flexible Image Generation Environment,'' ACM SIGGRAPH '82 Conference Proceedings, July 1982, .

2. Reynolds, C.W., ``Computer Animation with Scripts and Actors,'' ACM SIGGRAPH '82 Conference Proceedings, July 1982, .

3. Christian, K., The UNIX Operating System, John Wiley & Sons, 1983.

4. Leitner, G., ``A Distributed System for Digital Image Production,'' submitted for publication, 1984, .

5. Davis,R. and R.G. Smith, ``Negotiation as a Metaphor for Distributed Problem Solving,'' Tech. report 624, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1981.