University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers

Computer Science and Engineering, Department
of

11-1994

DeViouS: A Distributed Environment for Vision Tasks

Phillip R. Romig III

Colorado School of Mines, promig3@mines.edu

Ashok K. Samal

University of Nebraska-Lincoln, asamal1@unl.edu

Follow this and additional works at: <https://digitalcommons.unl.edu/cseconfwork>

Part of the [Computer Sciences Commons](#)

Romig, Phillip R. III and Samal, Ashok K., "DeViouS: A Distributed Environment for Vision Tasks" (1994).
CSE Conference and Workshop Papers. 35.

<https://digitalcommons.unl.edu/cseconfwork/35>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

DEVIOUS : A DISTRIBUTED ENVIRONMENT FOR VISION TASKS

Phillip R. Romig III and Ashok Samal

Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0015

ABSTRACT

We present a system for the integration of computer vision tasks in a distributed environment. This system, called DeVious, is based on the client/server model and runs in a heterogeneous environment of Unix workstations. It takes advantage of the free cycles in modern workstation environments to distribute and speed up the execution of vision tasks.

Two primary goals of DeVious are to provide a practical distributed system and a research environment for vision computing. DeVious is based on a modular design that allows experimentation in various aspects of algorithm design, scheduling and network programming. It can make use of any existing computer vision package with very minor changes to DeVious. DeVious has been tested in an environment of SUN and Digital workstations and has shown substantial improvements in speed over sequential computing with negligible overhead.

1. INTRODUCTION

To address the difficulty and complexity of automatic object recognition, the computer vision and image processing community has developed many algorithms, each of which does a single operation very well. Only recently, as the complexity of the problems we can attempt has increased, has the importance of the ordering and interaction between tasks become apparent[1, 2]. The processes of combining individual vision tasks to perform some complex recognition job is called *task integration*.

A system for computer vision in a distributed environment with intelligent task integration is presented here. This system, called DeVious (Distributed Vision System), is based on the client/server model and runs in a heterogeneous Unix environment[3, 4]. DeVious takes advantage of the large number of workstations in modern computing environments to distribute the execution of vision tasks.

The primary goals of DeVious are to provide a distributed system tuned for the integration of computer vision tasks and to provide a research environment for future distributed vision systems. DeVious has a modular design that allows experimentation in various aspects of algorithm design, scheduling and network programming. It can make use of any existing computer vision package with very minor changes. It has been tested in a heterogeneous environment of workstations and has shown substantial performance improvements over sequential computing with negligible overhead.

1.1. Terminology

In order to avoid ambiguity some common terms used in this paper are defined. A *task* is a single vision operation, e.g., edge detection, segmentation, hough transform, etc. A *job* is the work performed by the system in response to a user's request. A user may specify a request in the form of a high-level *goal*, which can be translated into a detailed *plan* of action (typically a acyclic directed dependency graph of tasks). For example, a goal may be to identify all the objects in an outdoor scene. The plan for this goal would be the ordered set of tasks required to find the objects. The job is an instance of the plan used by DeVious to achieve the goal.

2. MOTIVATION

Over the last five years there has been a substantive change in the way computing is done at universities and other large organizations. This change has been induced by the introduction of the personal workstation, with a low priced yet powerful CPU. This has resulted in the migration away from very large multi-user environments towards systems consisting of personal workstations interconnected through a high-speed network. Distributed computing makes use of the idle CPU cycles in such an environment by treating the workstations as a single computational unit, allowing

users with large jobs to transparently make use of several CPUs at the same time.

A fundamental characteristic of computer vision is that it is task-oriented. That is, a single goal can be broken down into a number of individual tasks[5, 6]. Because of this task-oriented nature computer vision is ideally suited for use in distributed and/or parallel environments. The design and implementation of parallel algorithms for vision has been an active area of research for some time[5, 7, 8, 9]. Distributed vision is a newer field of research and as such still has a great number of unanswered questions.

2.1. Advantages of Distributed Vision

There is a wide range of styles in computer vision today. Yet almost every type of vision researcher (or user) can expect to see some advantages to working in a distributed system. These users fall into eight broad categories:

1. One Simple Task One Image : Many users just use one task to compute the result for a single image or dataset, e.g., apply a new edge detector task on a test image during the debugging process.
2. One Compound Task One Image : A compound task consists of several related tasks that are put together using a graph to accomplish a higher level goal.
3. Repeated Simple Task Many Images : In many instances, a user attempts to use the same task on many images, perhaps to accumulate and compare the results.
4. Repeated Compound Task Many Images : This is same as the previous case, except the task may be a compound task.
5. Many Simple Tasks Same Image : Sometimes, a user wants to compare several similar operators, e.g. edge detector, on the same image. Text recognition of a page may require the results of an edge detection, a vertical profile and a horizontal profile of the page.
6. Many Compound Tasks Same Image : This is same as the previous case, except a task may be a compound task.
7. Many Simple Tasks Many Images : This is not a very common mode of computation, but in some instances the user may want to work on two independent "jobs" on different images.
8. Many Compound Tasks Many Images : This is same as the previous case, except a task may be a compound task.

DeViouS provides two types advantages to the vision researchers and application users; improvement in speed and increased productivity.

2.1.1. Speed

Most users performing vision computing will get direct and substantial benefit in speed using distributed computing. Of the eight major computing modes, the last six can be mapped easily to independent pieces of computation that can be transparently distributed in DeViouS.

Even a single compound task typically has a moderately high degree of independence between the tasks. Distributed computing systems like DeViouS can take advantage of this parallelism to speed up the total run-time.

Only a small number of vision users who run single tasks that are not data parallel and on single images are unlikely see substantial speed benefits. Even those users will benefit from load distribution that is inherent in all distributed systems including DeViouS.

2.1.2. Productivity

Speed is usually the only benefit mentioned when the advantages of distributed computing are discussed, and clearly faster computation will result in increased productivity. However, DeViouS provides other potential means of improving productivity.

One way to increase productivity is based on the idea of *speculative computing*[9]. It is often the case that several different combinations of vision tasks might be used to achieve some goal. There are, for example, a large number of edge detection kernels that may be used. On a single processor system the user (or an on-line expert system) will make a judgment as to which kernel will work best for a given image. If this judgment is wrong, the user will have to run the job again with a different kernel or live with sub-optimal results.

On a distributed system it is possible to run each of the possible task combinations in parallel without increasing the amount of time the user waits for the results. The "best" result can then be selected and the others simply thrown away. Since the CPUs used to compute these discarded jobs would have been idle in any event, there is no effective penalty.

Figure 1: An overview of the modules in DeVious.

3. DESIGN OF DEVIOUS

The steps performed when a vision problem is to be solved in a distributed system are: (a) the development of a *plan* by dividing the goal into interdependent vision tasks, (b) selecting which processor should run which task and at what time and (c) running the individual vision tasks and collecting their results. The design of DeVious mirrors these three operations with three modules. A *strategy planning module* is responsible for setting up a plan to achieve the goal specified by the user. The *scheduling module* determines the running order for the tasks and their locations. The *logical function modules* are units that actually perform the computations. In addition, there is a *user interface module* for interaction with application users. A general layout of the system is shown in Figure 1. DeVious is based on a design given in [9, 10] for parallel vision systems.

3.1. Strategy Planning Module

The first step in performing a vision job is the selection of the tasks necessary for completion of the goal and the integration of these tasks into a plan. A plan consists of an ordered (or partially ordered) set of tasks along with the necessary inputs and dependencies needed to complete the job. The part of the system responsible for generating a plan from a job specification is called the *strategy planning module* (SPM). The SPM has three primary duties: (a) task selection, (b) task ordering and (c) task dependency determination. In addition, it is also responsible for runtime monitoring and modification of the plan.

An advantage of the SPM is that by separating the functionality for creating the plan from the rest of the system, the SPM has added flexibility and may be tailored for specific goals. In the current implementation

the SPM is part of the user interface, the user must supply the plan by hand.

3.2. Scheduling Module

Once a plan has been constructed, the tasks must be assigned to a processor from the pool of available machines. This is the responsibility of the *scheduling module* (SM). The SM selects the best task to be done next and the best machine for that task from the pool of available tasks and processors. Optimal scheduling of multiple tasks on multiple heterogeneous processors with different expected runtimes is known to be NP-Hard[11, 12, 13, 14].

DeVious currently uses a scheduling algorithm called modified level scheduling with frontier lookahead. Each task in the plan is assigned a priority, also called the level of the task, based on its distance from a terminal task (terminal tasks are tasks on which no other tasks depend). The highest priority task is thus the one farthest from a terminal task. Computing priorities in this fashion gives a fair approximation of the relative importance of the various tasks (an important task is one on which many other tasks depend).

Once the tasks have been prioritized they are scheduled onto the "best" processor. The best processor is determined to be the one on which the task will finish first. The best processor may or may not be the fastest available processor. It may be the case that a very fast processor is currently busy with some other task, and thus not available, while a very slow processor is idle. It may actually be faster to hold the task until the fast processor finishes its current work, then assign the task to that processor, than it would be to run the task on the slower processor. This process of determining the best processor by computing the expected run time of the task on each processor (not just available processors) is called frontier lookahead.

While the modified level scheduling with frontier lookahead does not produce an optimal schedule, it does produce a good approximation and can be run in $O(n)$. It is also important to note that, like the strategy planning module, the scheduling module can be easily replaced for experimentation and evaluation of different algorithms.

3.3. Task Execution Modules

The most unique feature of DeVious is the task execution mechanism. Once the scheduler selects the CPU for a given task, control is passed over to a task execution module called a *logical function module* (LFM). The LFM is an intelligent unit capable of sensing the environment in which it is running and selecting the

best possible technique for accomplishing its assignment in that environment. For example, some computers may have special hardware for convolution. The **LFM** for edge detection, or any other operation requiring convolution, would have to select the version of the program that uses the special hardware when available and a different version under other conditions.

Since these **LFMs** are independent units they are easily added or modified so that new algorithms can be added to DeVious with ease. This allows DeVious to use any existing computer vision software. Currently DeVious makes use of the IKS system and work is underway to include the KHOROS and KBVision systems.

The **LFMs**, the **SPM**, and the **SM**, along with a user interface module and the code to access the underlying machine and network protocols make up the DeVious system.

4. IMPLEMENTATION

DeVious is based on the client/server paradigm. The *server* process is responsible for coordination and communication between the units in the system. The server can run on any computer available to the system and may share it with the clients. The server's coordination responsibilities include maintaining the data structures for the plan, keeping track of all the clients, and scheduling the tasks. Communication between the components, is achieved through message passing. Messages are passed from a client to the server or the server to a client, but never from one client to another client.

A *client* handles the execution of an individual task. Clients are started by the server and may be added or removed at any time during the lifespan of a job. A client can only handle only one task at a time; however, it is possible to run multiple clients on a single computer with multiple CPUs.

A *front end* is a special type of client that provides the users with a window into the system, allowing them to submit jobs, add or remove tasks and monitor the progress of the job through the system. A server can have only one active front end at any given time. However, it is possible to stop the front end while a job is in progress and return to it later, possibly on a different computer, and re-connect to the server with a new front end.

4.1. Integration

The various modules are combined in a layered, modular architecture. That is, each of modules is fully independent of the others. Communication is achieved

through a small set of predefined messages. This allows existing modules to be replaced, either to fix bugs or to test new algorithms, by new modules so long as the new modules respond to the message set defined for it.

In addition to breaking the software into modules, the components of DeVious were also layered to make porting to new operating systems as painless as possible. All operating system dependent code is isolated in the *hardware layer* which constitutes only a small part of the whole system. Porting to a new operating system requires only that this layer be replaced. The system is currently implemented in C++ on Unix workstations.

5. RESULTS

DeVious has been thoroughly tested and has demonstrated the feasibility and flexibility of its design. However, several of the modules have not yet had all the proposed features implemented. Most notably the **SPM** was left only as a skeleton; the user must supply the plan. Evaluation of several different planning systems is currently under way and at least one will be incorporated soon.

The test environment consisted of 23 machines, fifteen of them Sun Sparcstations and seven Digital decStations. Each machine had at least 16 MBytes of RAM and a local disk. There was also one Sun MP/670 with 4 processors and 64 MBytes of main memory. A standard image size of 512×512 was used throughout the testing process. The system was tested with about a hundred tasks available in the IKS vision system. Almost all of the IKS tasks were used, those left out either required user interaction or could not be compiled on one of the target operating systems. Results show that DeVious provides significant improvements in speed for the user.

Three different types of plans were tested. First the overhead incurred by DeVious was measured by running a plan which had fifty tasks and no parallelism. With this plan there can be no speedup and any difference between the execution of this plan and the same fifty tasks without DeVious is due to overhead. Using this method we measured an overhead of 17%. Experience with DeVious has shown that the above number is overly pessimistic and the actual overhead is closer to 5% in most real world applications.

Next, DeVious was tested with plans that were completely parallel. Some, which did not require disk I/O, were used to test DeVious's performance independent of network or disk bandwidth. In these tests we achieved close to optimal speedup. A second completely parallel plan was tested with tasks that did re-

quire I/O. In this case the performance matched the no-I/O case for introduction of the first five or six CPUs. After that the speedup gained by adding more CPUs dropped off substantially. This indicates that disk access becomes a limiting factor, and a better data distribution mechanism is needed.

Finally, DeVioUS was tested with plans similar to what is used in real vision applications. These plans consisted of between fifteen and twenty tasks with varying degrees of parallelism. These tests also achieved marked speedup. However, we also observed performance drop-off similar to the previous case. Three reasons were identified for this drop-off. Disk contention was still a major problem, however critical paths in the plan and weaknesses in level scheduling algorithm also played a part. These problems can be addressed by improving the plan generation phase and providing better information for the scheduler.

6. SUMMARY

The DeVioUS experience has already provided us with important insights into the issues involved in implementing a practical distributed vision system. The significant improvement in performance indicates that the fundamental principles and design behind DeVioUS are sound, and it can now provide a framework from which the individual components of distributed vision systems can be studied.

Work on DeVioUS is still ongoing. Currently, we are porting DeVioUS to an environment of workstations connected by an extremely high speed ATM network. This will reduce the network congestion and make the distribution of tasks and data more efficient. This will also allow us to experiment with task migration, new data distribution protocols and distributed scheduling. We are also working on the development of intelligent planners in order to make DeVioUS a complete and fully featured distributed vision system.

7. REFERENCES

- [1] Charles Herzfeld. Grand challenges: High performance computing and communications. A Report by the Committee on Physical, Mathematical, and Engineering Sciences, Washington, D.C., 1992.
- [2] Shahriar Negahdaripour and Anil K. Jain, editors. *Final Report of The NFS Workshop on the Challenges in Computer Vision Research; Future Directions of Research*, June 7-8 1991.
- [3] Phillip R. Romig III. DeVioUS: A distributed environment for vision tasks. Master's thesis, University of Nebraska-Lincoln, May 1993.
- [4] Phillip R. Romig III and Ashok Samal. DeVioUS: A distributed environment for vision tasks. *Submitted to Software-Practice and Experience*, May 1993.
- [5] Katsushi Ikeuchi and Martial Hebert. Task oriented vision. Technical report, Carnegie Mellon University, 1991. CMU-CS-91-163.
- [6] Jon A. Webb. Steps toward architecture-independent image processing. *Computer*, 25(2):21-31, February 1992.
- [7] C.H. Chien and L.J. Lin. Paradigm: An architecture for distributed vision processing. In *Proceedings of 10th International Conference On Pattern Recognition*, volume 2, pages 648-653, June 1990.
- [8] Leah H. Jamieson et al. A software environment for parallel computer vision. Technical report, Purdue University, 1992.
- [9] Ashok Samal. Design, implementation and performance evaluation of a dynamically reconfigurable, integrated, parallel computer vision system. In *Proceedings of International Conference on Pattern Recognition*, pages 521-523. IEEE, 1990.
- [10] Ashok Samal. Design and implementation of a dynamically reconfigurable, integrated, parallel vision system. In *Proceedings of the International Conference On Automation, Robotics, and Computer Vision*. IEEE, 1990. 921-925.
- [11] Mikhail J. Atallah et al. Models and algorithms for coscheduling compute-intensive tasks on a network of workstations. *Journal of Parallel and Distributed Computing*, 16(4):319-327, December 1993.
- [12] Alok N. Choudhary and Ravi Ponnusamy. Guided scheduling schemes for image understanding tasks for shared and distributed memory multiprocessors. *The Fifth International Parallel Processing Symposium*, pages 260-264, May 1991.
- [13] Krishna P. Pelkhale and Prithviraj Banerjee. A scheduling algorithm for parallelizable dependent tasks. *The Fifth International Parallel Processing Symposium*, pages 500-506, April 1991.
- [14] E.G. Coffman, Jr., editor. *Computer and Job-Shop Scheduling Theory*. John Wiley and Sons, New York, 1976.