Case Study: Resource Steering in a Visualization System

Ed H. Chi*, John Riedl

Computer Science and Engineering Department, University of Minnesota
4-192 EE/CS Building, Minneapolis, MN 55455
[echi,riedl]@cs.umn.edu

Abstract. Visual computational steering environments extend traditional visualization environments by enabling the user to interactively steer the computations applied to the data. In this paper, we develop a new type of computational steering. "Resource steering" extends current visual steering techniques by providing machine resource estimation and control to the user. With resource steering, the user controls the execution of the computation on a parallel or distributed computer based on experimentally or theoretically derived estimates of the parallel performance of the computation. We demonstrate this extended steering model by applying it to an information visualization system that analyzes genetic sequence similarity reports. We show how our extended steering model enhances the user's ability to control visualization computations.

1 Introduction

The process of discovery includes trial-and-error followed by deductive insight. Computational steering in visualization enables users to interactively control the computation as it is being visualized, enabling us to apply human intuition and experience to the analysis of data sets. Although existing visual steering environments enhance computational interactivity, users do not receive feedback on computational resource usage, and have no control over how resources are used.

It is important to be able to view and interact with the data as it is generated [10, 13]. Early work include Interactive Zoner, which is an interactive grid generator running on a workstation that is connected to a 2D flow simulation running on a supercomputer [2]. Marshall et. al.ś system enable users to change the parameters of an algorithm as the computation proceeds [9, 15]. The user has two types of control. She uses the interactive control to direct the simulation by changing variables or the simulation condition. She can also manipulate the visualization, such as rotating, translating, zooming, and scaling. (Figure 1)

Recent research has refined computational steering. Jablonowski et. al. described a system where a program could be partially rewritten to support steerable visualization in IRIS Explorer [7]. They provided a framework for a computation written in FORTRAN to include directives for steering the algorithm. Mulder et. al. discussed a customizable widget system where attributes of the widgets are linked to steerable algorithmic variables, and direct manipulations of the widgets result in state changes in the variables [11]. Brodlie et. al. implemented steering along with a history tree that allowed the user to return to previous computation results [3]. Many other systems for visual steering have been developed, such as CUMULVS (Oak Ridge), VIPER (TU

* Work done while at U of M, current contact: chi@acm.org, Xerox PARC, Palo Alto, CA 94304

Muenich), SCIRun (Utah), Progress and Magellan (Georgia Tech). Several of these pay attention to distributed computing. The SCIRun system provide some interactive visual and quantitative feedback of performance metrics [12]. Pablo and Falcon are some recent work on parallel performance metrics for steering systems provide library routines for instrumenting source code to extract performance data [14].

Visualization applications should extend computational steering to provide resource usage feedback and control to the user for two reasons. First, visualization systems utilize expensive and scarce computing resources, which means users must carefully control resource usage. Visualization applications should provide ways to manage these resources during the entire user session. Second, users interacting with visualization environments can initiate long computations with simple manipulations of the interface. When users initiate computation, they often cannot predict the amount of system resources needed or the length of time required to complete the computation.

We introduce an extended model that incorporates several types of computational steering. Previous steering techniques can be classified into two types: (1) Parameter steering is the specification of initial parameters to the algorithm, and (2) algorithm steering is changing some aspect of the algorithm during computation. We introduce a new steering technique called *resource steering* that allows the user to control the amount of computational resources utilized. Based on resource limits chosen by the user, the model predicts how long the computation will take, and how efficiently it will make use of computing resources.

We tested our idea by extending a computational biology visualization system [5, 4] to including these types of computation steering. The visualization application helps biologists analyze genetic sequence similarity. Previously, biologists used the system in batch mode by creating data sets off-line. In the new system, biologists initiate and steer computations. The computations use an appropriate number of processors, based on the user's selection of desired response time or efficiency.

2 Resource Steering under Our Model

Our steering model is shown in Figure 2, which depicts the interactions between the user and the visualization analysis loop. First, the user initiates an analysis session by

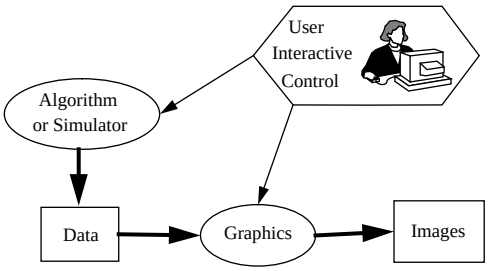

Fig. 1. Existing steering model. The simulation generates data that is given to a visualizer to generate the images on the screen. The user can interact with parameters of the algorithm and manipulate the objects on the screen.

specifying the initial parameters and conditions. Then the parameters are fed into the simulation algorithm to start the simulation process. The parallel machine, in response to the request, starts multiple processors on the simulation, and then generates the data. The data is then processed and given to the visualizer to present to the user.

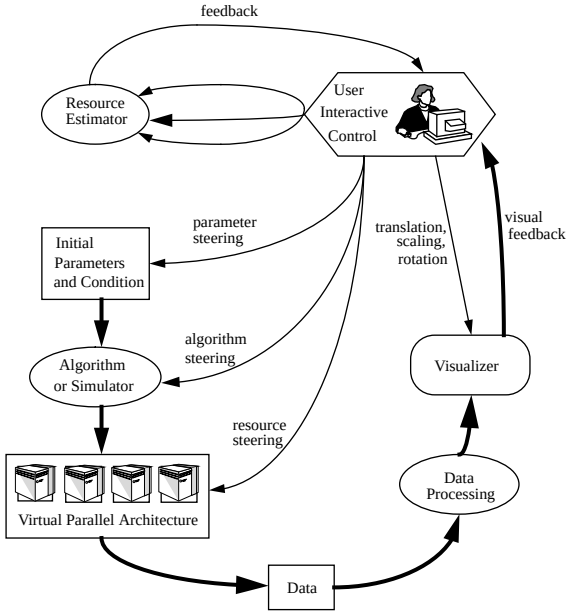

Fig. 2. Extended steering model. Thin arrows represent flow of control, while wide arrows represent the flow of data.

Previous computational steering can be described in two ways: (1) In *parameter steering*, the initial input parameters and conditions are specified for every run of the algorithm. Interactive manipulation of initial input parameters to the algorithm is often mentioned as a simple form of computational steering. In some cases, the computation can be parallelized effectively to bring the computation time down to interactive speed (tens of seconds or a few minutes), making the interactivity with initial input parameters or conditions significant. (2) In *algorithm steering*, some aspect of the algorithm is changed during the execution of the simulation. An example of algorithm steering is the specification of the number of iterations of an iterative numeric method, or perhaps an entirely different algorithm.

In *resource steering*, the computation engine itself is modified either during the execution or from run to run. This feedback can be useful in many situations, such as when the amount of computational resources is limited, or the user and the administrator of the system is interested in obtaining good efficiency. If the response time is too slow, the user might want to increase the number of processors working on the computation. The user may also be interested in the efficiency of the system, especially if she is paying for supercomputing time.

We can provide this feedback by studying the performance of the parallel system under different conditions, and then predicting the performance. For example, given some initial parameters, the current state of the algorithm, and the machine characteristics desired (the number of processors), the estimator returns the expected response time. Alternatively, given some desired response time, the estimator returns the number of processors needed to meet that requirement.

In our model, the three types of steering parameters are given to a resource estimator as constraints. The resource estimator then returns an estimation of the amount of resources needed. The resource estimator uses a *resource estimation function* that maps from the steering parameters to the amount of resources needed. Here we will first describe how we can obtain a performance model for the resource estimation function. Then we discuss how to use the model in a resource estimator.

2.1 Establishing a performance model

The *resource estimation function* is a function that takes the parameter space as input, and returns the runtime (e.g. runtime = r.e.f.($X_1, X_2, X_3, \ldots$), where X_i is the input parameters). To predict performance, we can obtain the resource estimation function by using either a theoretical model or an experimental model.

Theoretical Model To develop a theoretical model, we can derive the parallel runtime T_p from the algorithm that solves the problem. The parallel runtime is a function of the parameters and the number of processors. Theoretical derivations have been verified and shown to model the parallel system reasonably in many cases [8]. For example, consider the problem of adding n numbers on a p-processor machine arranged on a hypercube network [8]. Let us assume that it takes T_a units of time to add two numbers and T_c units of time to communicate one number between two directly connected processors. Each processor is assigned n/p numbers initially, and this will take $(n/p - 1)T_a$ to sum at each processor. After the local results are known, it takes $\log p$ steps to add these partial sums together, where each step takes one addition and one communication. Thus, knowing n and p, we can compute the parallel runtime. The parallel runtime is:

$$T_p = (n/p - 1)T_a + (T_a + T_c)\log p \tag{1}$$

The theoretical derivation for many types of algorithms is available in the parallel computing literature, such as sorting, graph algorithms, dynamic programming, fast Fourier transforms, numerical algorithms, and discrete optimization algorithms [8]. If the scalability of a parallel algorithm is known, we can use it to predict the parallel runtime, given the initial parameters and the number of processors.

Experimental model We can also use an experimental model of the algorithm for establishing the performance model. Performing experiments to obtain the resource estimation function is often necessary for a variety of reasons. For example, (a) the theoretical derivation might be difficult; (b) there are unknown variables in the computation; and (c) there is a need to establish the validity of a theoretical model or a need to be more accurate. Therefore, often an experimental model is desirable.

The major drawback of experimental models is that they are relatively difficult to obtain accurately. However, the difficulty can be reduced with carefully planning of

experiments. We can establish an experimental performance model by first systematically listing all the steering parameters, and then varying the steering parameters and studying the effect on the parallel runtime. Imagine the parameter space coupled with an axis measuring runtime. We can sample this parameter space on a grid, and measure the runtime at each point on the grid. Then we obtain the runtime characteristic as a multi-dimensional surface in this parameter space. Given this rough sketch of the estimation function, we can interpolate between the points to obtain an runtime estimation. This will work well if there are not too many sample points (e.g. there are only a handful of parameters, and the ranges of each parameter is relatively narrow.) In our case study, we used this method to study the effect of each of the steering parameters on the parallel runtime. The limitation of this approach is that this parameter space may not be continuous, since other factors such as number of page faults can result in substantial jumps in runtime. Experimental modelling works well if the steering parameters are mostly independent from each other, and do not affect the runtime in some unexpected way when combined.

2.2 Resource Estimator: Using the performance model to estimate resources

We can use the performance model in a variety of ways, since given all except one variable of the resource estimation function, we can determine the value of that last variable. For example, (1) We can use the performance model straightforwardly to obtain an estimated runtime. Since the user often cannot predict the length of a computation, this provides the necessary feedback to the user. (2) We can use the model to predict the number of processors p needed when given the desired response time. In order to do this, we need to invert the estimation function with respect to p. We can obtain the inverted function directly if we know the precise runtime equation for the function. Or we can compute p implicitly by using a simple for-loop going from $p = 1$ to the maximum number of processors available, and stopping when p can satisfy the required runtime. (3) We can use the estimation function to predict the speedup and the efficiency. In an experimental model, given T_p, we can directly compute speedup and efficiency as $S = \frac{T_s}{T_p}$ and $E = \frac{S}{p} = \frac{T_s}{pT_p}$, respectively. Given the runtime equations, we can directly compute the algebraic representation of speedup and efficiency. For example, in the example of parallel addition above:

$$S = \frac{nT_a}{(n/p-1)T_a + (T_a + T_c)\log p} \tag{2}$$

$$E = \frac{nT_a}{(n-p)T_a + (T_a + T_c)p\log p} \tag{3}$$

So given the number of processors p, we can obtain the parallel runtime and the efficiency of the system. We can again invert the efficiency estimation function to calculate how many processors we can utilize when we allow the efficiency to drop.

The above examples show that the user can specify any one of the resource steering parameters (e.g. runtime, number of processors, and efficiency), and get an estimation of the other parameters. In high resource load situations, users can use resource steering to obtain efficient performance. If the user is paying for the computing time, this method gives them the power to manage the classic tradeoff between efficiency and speed. This

allows the users of the system to have strict control over the utilization of computing resources.

3 An Application of Resource Steering

Here we describe a problem solving environment for genetic sequence similarity computation that incorporates the above ideas and illustrate the techniques for controlling resource usage.

Computational Model Molecular biologists utilize known sequence data in large genomic databases and similarity algorithms to help determine the function of new sequences. BLAST [1] is a well-known similarity algorithm. We first derived a rough theoretical derivation of the BLAST parallel performance model. The BLAST sequence comparison algorithm first serially computes some hashing information, then the database is partitioned into many pieces, which are then given to each of the processors on-the-fly to compare. Thus, the parallel runtime is the time for the serial component plus the parallel database searching component. We expect the parallel component to be nearly linearly scalable, since the database is large compared to the size of the query sequence. However, because the database content is not uniform, and we are not confident whether dynamic load-balancing contributes a large overhead, an experimental study is needed to help us refine and validate the theoretical model. Such validation is required in many applications.

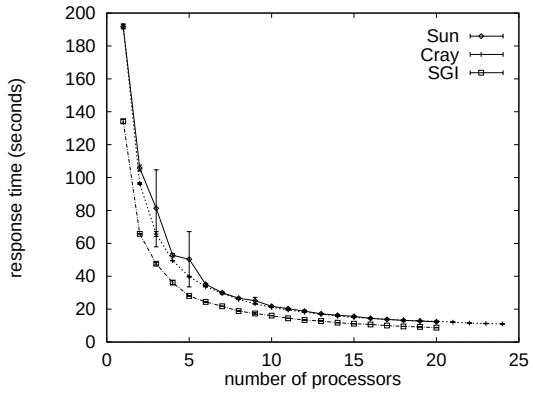

Fig. 3. Response time for SGI Challenge, Sun SparcCenter 2000, and Cray CS6400 with a 500 base sequence using parallel BLASTX SMP algorithm.

For BLAST, we obtained and evaluated the experimental model using several different length sequences on three Shared-Memory Parallel machines (SGI Challenge XL, Sun SparcCenter 2000, Cray CS6400). The parallel response time curve is plotted in Figure 3. We refined the performance model to include other parameter of the algorithm. Sequence comparison is linearly scalable up to tens of processors, so we can parallelized heavily to bring the computation close to interactive speeds (tens of seconds or several minutes). For example, a BLAST process that took nearly 1.3 hours on a single processor completed in only 3.5 minutes with 24 processors on Cray CS6400 [6]. Short sequences that used to take 5–10 minutes can be done in tens of seconds.

Application Usage Walk Through Previously, we presented a system called AlignmentViewer for visualizing similarity information between a single DNA sequence and a large database of other DNA sequences [5]. Each report consists of an input sequence and many *alignments*. An alignment indicates a region of similarity between two sequences. Each alignment has a matching vector and twelve variables. AlignmentViewer uses three spatial axes and one temporal axis. Any of twelve variables can be mapped to any of the four axes. The temporal axis allows the user to construct animations with respect to the temporal variable [5]. The matching vector is represented by a comb-like glyph. For more details on the visualization representation, please see our previous papers.

Fig. 4.1: Steering Control Panel: For response time ¡ 20 sec., this sequence requires 10 processors. The efficiency is estimated =89%.

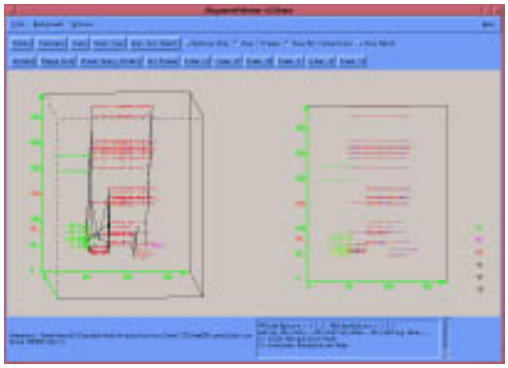

Fig. 4.2: AlignmentViewer's visual representation of PAM250 result for an *Arabidopsis thaliana* sequence 172C2T7.

Let us demonstrate a real scenario of a biologist steering the computation of a sequence. The system asks the user how to process the input sequence using the steering control panel (Figure 4.1, initial parameters in the top half, resource steering controls in the bottom half). The default of one processor is predicted to require 152 seconds, but the biologist wants the response time to be shorter. She drags the response time slider to 20 seconds. The estimating function then computes the required number of processors and the resulting efficiency to meet that request. As shown in Figure 4.1, this sequence requires 10 processors to get the response time under 20 seconds on a SGI Challenge XL machine. The efficiency will be an estimated 89 percent, which is above the 60 percent minimum specified by the system administrator. The user then presses the "Yes" button to start the algorithm running with 10 processors. The result of the computation is visually presented to the user (Figure 4.2).

This sequence report has many good alignments, represented by the abundance of red teeth on the combs. This suggests that the database sequences we found are all closely related. After some analysis, the biologist decided to re-run the algorithm with parameters of closer evolutionary distance. The biologist obtains a new visualization by running the sequence algorithm again with parameter matrix=PAM60 and asking for at least 90 percent efficiency.

A new feature of our system implements the subtraction of two visualized data sets, showing alignments found by one data set but not the other. By using "detail-on-demand", the biologist select an alignment and then view the details of that alignment in a separate window. The result is that we have found "motifs" of a protein called "peroxidase." Motifs are short regions that have been preserved with little change over evolution, presumably because their existence is important to the function of the protein.

The above example showed that the resource steering works well in practice. Our molecular biology application incorporates computational steering in a visualization system that analyzes genetic sequence similarity reports. Using this model, the user is able to change the initial parameters of the algorithm and estimate what effect the parameters will have on the performance of the system. Biologists have been regularly using this system since late 1996. The feedback we have received from them is that using this visualization system has increased their ability to analyze large amounts of similarity data by at least two or three folds.

4 Conclusion

Existing systems do not provide feedback on the efficiency of the computational engine. This feedback is important for two reasons. First, if the user initiates a simulation, she should have information that tells her how long the simulation will take. Second, since computing resources can be scarce, users need to be able to specify the upper limit on the usage of resources. In this paper, we described a particular approach to computation steering that enable users to closely monitor resource usage. Resource steering can be used in many situations, offering the user the ability to fine-tune the efficiency and response time. Sometimes the system administrator may impose minimum efficiency on the computing resources. The system may even provide an estimate of the amount of dollars a particular computation may cost. As depicted in the case study, the new combined approach enhances our ability to integrate computation with visualization.

Acknowledgments This work has been supported in part by the National Science Foundation under grants BIR9402380 and CDA9414015. We wish to thank members of the Arabidopsis sequencing group at Michigan State University and the genomic database group at the University of Minnesota for their advice and suggestions.

References

1. S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.
2. G. Bancroft, T. Plessel, F. Merritt, and V. Watson. Tools for 3d scientific visualization in computational aerodynamics at NASA Ames Research Center. In *Proc. SPIE 1083: Three-Dimensional Visualization and Display Technologies*, pages 161–172, 1989.

3. K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay. Grasparc — a problem solving environment integrating computation and visualization. In *IEEE Visualization '93*, pages 102–109. IEEE CS Press, 1993.
4. E. H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *Proceedings of the Symposium on Information Visualization '97*, pages 17–24,116. IEEE CS, 1997. Phoenix, Arizona.
5. E. H. Chi, J. Riedl, E. Shoop, J. V. Carlis, E. Retzel, and P. Barry. Flexible information visualization of multivariate data from biological sequence similarity searches. In *Proc. IEEE Visualization '96*, pages 133–140, 477. IEEE CS, 1996. San Francisco, California.
6. E. H. Chi, E. Shoop, J. Carlis, E. Retzel, and J. Riedl. Efficiency of shared-memory multiprocessors for a genetic sequence similarity search algorithm. Technical Report TR97-005, University of Minnesota Computer Science Department, 1997.
7. D. Jablonowski, J. Bruner, B. Bliss, and R. Haber. VASE: The visualization and application steering environment. In *IEEE Visualization '93*, pages 560–569. IEEE CS Press, 1993.
8. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Benjamin Cummings, 1994.
9. R. Marshall, J. Kempf, S. Dyer, and C.-C. Yen. Visualization methods and simulation steering for a 3D turbulence model of Lake Erie. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 89–97, 264. SIGGRAPH, 1990.
10. B. McCormick et al. Visualization in scientific computing. In *Computer Graphics*, volume 21. ACM Press, November 1987.
11. J. Mulder and J. van Wijk. 3D computational steering with parametrized geometric objects. In *IEEE Visualization '95*, pages 304–311. IEEE CS Press, 1995.
12. S. Parker, D. Weinstein, and C. Johnson. The SCIRun computational steering software system. In E. Arge, A. Bruaset, and H. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 1–44. Birkhauser Press, 1997.
13. D. Reed, C. Elford, T. Madhyastha, E. Smirni, and S. Lamm. The next frontier: Interactive and closed loop performance steering. In *Proceedings of the 25th Annual Conference of International Conference on Parallel Processing*, 1996.
14. J. Vetter and K. Schwan. Progress: A toolkit for interactive program steering. In *Proceedings of the 24th Annual Conference of International Conference on Parallel Processing*, pages 139 – 142, 1995.
15. C.-C. Yen, K. Bedford, J. Kempf, and R. Marshall. A three-dimensional/stereoscopic display and model control system for great lakes forecasts. In *IEEE Visualization '90*, pages 194–201. IEEE CS Press, 1990.