Title

Resource Allocation for Steerable Parallel Parameter Searches: an Experimental Study

Authors

Faerman, Marcio
Birnbaum, Adam
Casanova, Henri
et al.

Publication Date

2002-08-18

Peer reviewed

Resource Allocation for Steerable Parallel Parameter Searches: an Experimental Study

Marcio Faerman [1] Adam Birnbaum [2]
Henri Casanova [1,2] Fran Berman [1,2]

[1] Department of Computer Science and Engineering, University of California, San Diego
[2] San Diego Supercomputer Center

mfaerman@cs.ucsd.edu [birnbaum,casanova,berman]@sdsc.edu

Abstract

Computational Grids lend themselves well to parameter sweep applications, which consist of independent tasks, each of which calculates results for a separate point in parameter space. However, it is possible for a parameter space to become so large as to pose prohibitive system requirements. In these cases, user-directed steering *promises to reduce overall computation time. In this paper, we address an interesting challenge posed by these user-directed searches: how should compute resources be allocated to application tasks as the overall computation is being steered by the user? We present a model for user-directed searches, and then propose a number of resource allocation strategies and evaluate them in simulation. We find that prioritizing the assignments of tasks to compute resources throughout the search can lead to substantial performance improvements. We present experimental results obtained with software developed as part of the Virtual Instrument project, and discuss the impact of our findings on future Virtual Instrument implementations.*

1 Introduction

An increasingly common class of scientific applications is that of *Parameter Sweep Applications*: applications that consist of large sets of *independent* computational tasks. Each task typically evaluates a multi-dimensional *objective function* at a point in a multi-dimensional parameter space. Parameter Sweep Applications arise in many fields, e.g. Computational Fluid Dynamics [32], Bioinformatics [3, 16], Particle Physics [23], Protein Folding [28], etc. From a parallel computing perspective, these applications present many advantages due to their simple structure. They can tolerate high network latencies due to the lack of inter-task communications. They can benefit from simple fault-detection/restart mechanisms due to the lack of hard task synchronization requirements. Therefore, they are ideally suited to emerging *Computational Grids* where resources are widely distributed and loosely coupled. Nevertheless they pose interesting challenges in terms of scheduling and deployment, some of which we have addressed in our previous work [11, 12]. That work resulted in the APST software [4], which is currently being used in production for several parameter sweep applications.

Based on our experience, we have seen that in many cases parameter spaces can become too large to be entirely computed even on large-scale platforms. This is due both to extensive parameter value ranges, and to high dimensionality of the parameter space itself. Furthermore, even though exhaustively computing values over an entire parameter space would be ideal, users are often mostly interested in finding certain patterns in a parameter space (e.g. a set of parameter values that leads to certain values of the objective function). Therefore, an appealing approach is to *search* the parameter space for these patterns, in order to save both time and compute resources. We denote such applications as *Parameter Search Applications*. A large number of algorithms have been developed and used successfully for such searches [13, 37, 5, 15, 6, 7, 18, 21, 26]. These algorithms, often called guided-search algorithms, can be very effective when the user's objectives are clearly specified (e.g. minimize a function).

This work was supported by the National Science Foundation under Award ACI-0086092.

In this work we focus on scenarios in which the user *interactively steers* the search. In many real-world applications, the user's objectives cannot be precisely specified, or are not even known at the onset of the search (e.g. users are looking for something "interesting"). This subjectivity makes it impossible to automate the search process, and it is necessary to involve the user directly. For instance, one can let the user periodically indicate which regions of the parameter space seem more or less likely to contain solutions or features of interest, so that the search can concentrate on promising regions. We denote such search scenarios as *user-directed searches*. In this paper we make the following contribution. We propose and evaluate a scheduling approach that improves the performance of user-directed parallel parameter space searches on distributed computing platforms. Our goal is *not* to develop novel search/optimization algorithms, but rather to investigate techniques to allocate appropriate computing resources to tasks of a user-directed search.

Our work is in the context of the Virtual Instrument (VI) project [9, 40], which focuses on a computational biology application: MCell [25]. MCell is a simulation framework that is currently used by neuroscientists to explore potentially large parameter spaces for subjectively "interesting features". The exploration of very large parameter spaces is one of a number of challenges that must be overcome in order to deploy large-scale MCell simulations on Computational Grid platforms [10]. This paper explores user-directed steering and resource allocation methods that will have a direct impact on the VI project, enabling new classes of MCell simulations.

This paper is organized as follows. In Section 2 we present our experimental methodology, discuss user-directed searches, and state our assumptions. Section 3 introduces the Virtual Instrument project, describes its current scheduling strategy, and discusses experimental results. Section 4 presents a resource allocation strategy that implements task priorities, along with results of simulations built to evaluate the strategy. We discuss related work in Section 5 and future work in Section 6.

2 Background

2.1 Methodology

Evaluation of scheduling strategies for user-directed searches faces an inherent difficulty: specific behaviors are unknown until users have access to production software. Steering in the VI will be based on iterative feedback from the user. The user will begin running the application with a small set of parameter space points. As the application runs and results are returned, the user may add new points for evaluation, and may also assign different levels of importance to each point. Once the VI software has been released to a large community of MCell users and is being used in a production environment, we will be able to evaluate different scheduling strategies in real steering scenarios that make use of this user input. However, for the time being we chose to *simulate* user behavior.

In order to simulate user behavior, we replace the search for subjective "interesting" features with an iterative search for the minimum of the well-known, "hard-to-optimize", Griewank function [36, 19]. The Griewank function we use in this paper, g_σ, is defined as:

$$g_\sigma(x_1, ..., x_D) = 1 + \frac{\sigma}{4000} \sum_{i=1}^{i=D} x_i^2 - \prod_{i=1}^{i=D} \cos(\frac{x_i}{\sqrt{i}}),$$

where σ is a real that parameterizes the overall shape of the function. Figure 1 displays the 1-D griewank function for a few values of σ (we use different scales on the x-axis so that the overall shape of the function is easily visible). In the rest of the paper we use the 2-D Griewank function for experiments and simulations.

There is a large number of existing algorithms for performing guided searches [13, 37, 5, 15, 6, 7, 18, 21, 26]. Our goal is to test scheduling and resource allocation strategies that will enable user steering, allowing users to reach useful results more quickly by allocating more compute resources to promising areas of parameter space. We are not trying to develop novel search algorithms, since the searches in question will ultimately involve subjective user goals and criteria. Based on discussions with MCell researchers, we have developed a search procedure that is representative of what a user might do when exploring an MCell parameter space. We believe that this search procedure, described in the next section, provides a useful basis for evaluating steering and scheduling techniques.

2.2 Search Strategy

We present here the overall process for a user-directed search. The search starts with the evaluation of the objective function for a number of points uniformly scattered over the parameter space of interest. These evaluations present the user with an initial, very sparse, view of the parameter space. Based on judgments of these results, the user can assign *levels of importance* to regions on the parameter space. The levels of importance indicate whether some regions are more promising than others and should therefore be explored sooner. The search proceeds by evaluating the objective functions at several points within regions with high levels of importance. In effect, this increases the search resolution within those regions. This process is repeated iteratively until the user has found a (small enough) region or a parameter space point that meets his/her subjective criteria. The search really consists of a hierarchy of sub-searches

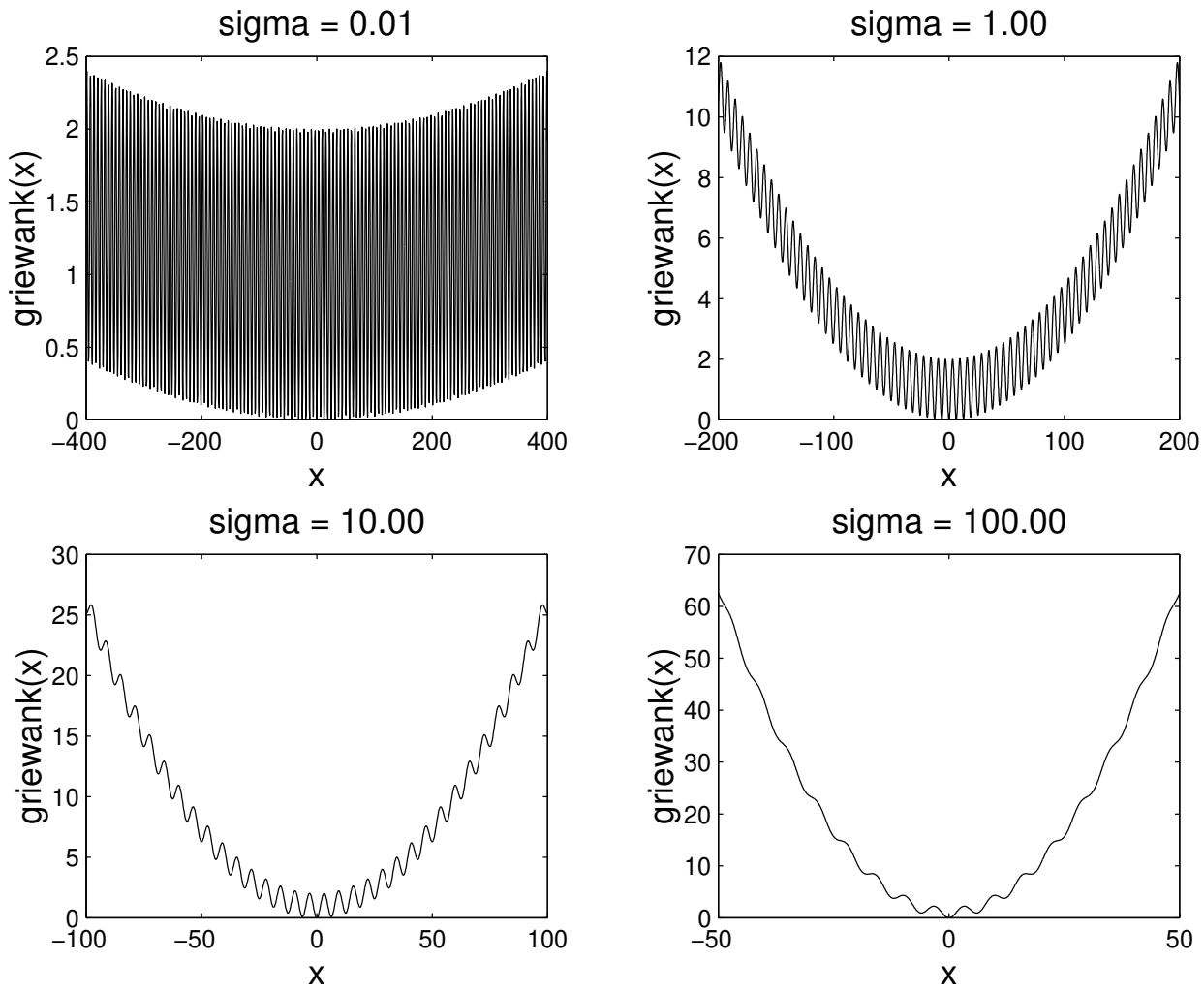

Figure 1. The 1-D Griewank function for $\sigma = 0.01$, $\sigma = 1.00$, $\sigma = 10.00$, and $\sigma = 100.00$.

that progressively prune search regions and refine the search resolution. This is depicted in Figure 2 for a 2-D parameter space. We show three levels of search, where points with high levels of importance are shown in black.

In the instantiation of the search procedure used in this paper, the search starts with 4 points at coordinates $(100, 100)$, $(100, -100)$, $(-100, -100)$, and $(-100, 100)$ in the x-y plane. These 4 points form generation 0 of the search. Each point is associated with an objective function *value* (in our case-study, the Griewank value evaluated at that point). At each generation, local random searches are conducted in neighboring regions of points from the previous generations. These regions are spherical and of decreasing radius at each generation. For the sake of simplicity, we call the objective function value of the point that has triggered the exploration of a region the *parent value* of new tasks in this region. In each region, the local search consists in evaluating 10 random points. At most the 2 points with the best objective function evaluations are propagated to the next generation. If such a local search does not return any point that has a lower objective function value than the parent value, then the local search is repeated up to a maximum number of times: $maxTrials$. If $maxTrials$ is reached and no improvement has been seen, then we use a standard random restart mechanism by selecting a new, random, parameter space point. A smaller $maxTrials$ leads to a more random and exploratory search, whereas a larger $maxTrials$ leads to a more thorough search.

The goal of this work is to investigate whether it is possible to reach useful results more quickly by allocating more compute resources to areas of parameter space that the user judges to be more promising. Since our user simulation is built to minimize the Griewank function, we assign levels of importance to points in the parameter space inversely cor-

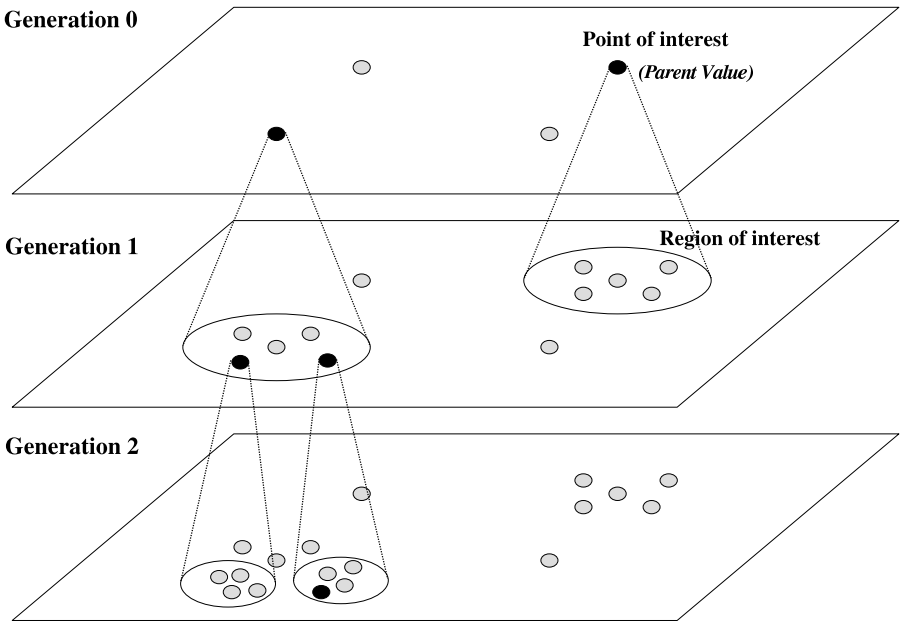

Figure 2. Overall User-directed Search Strategy

related to the parent values. The scheduler will then assign compute resources to regions of parameter space based on their importance. The search algorithm presented here combines global and local search strategies, similarly to other well known search strategies [18, 21, 22]. Our search model is configurable, allowing us to adjust the accuracy and resolution of the search process, using parameter $maxTrials$. Moreover the "bumpiness" of the Griewank function is configurable as well: the parameter σ controls relative height between the bottoms of basins of attraction. In the rest of the paper, we will use the above algorithm with different values of $maxTrials$ to minimize Griewank functions with different values of σ in order to simulate a range of user steering behaviors.

2.3 Assumptions

We make a number of important assumptions in this paper. In the search strategy described in the previous section, we use objective function values to determine the levels of importance for the search. This assumes that there is some spatial correlation of the objective function, meaning that function values and parameter space point locations are in some way correlated. Without such correlation, the only viable strategy is a purely random search. For MCell, and many other applications, the assumption of spatial correlation is valid.

In addition, our scheduling strategy assumes that each function evaluation, or task, requires the same amount of computation and that compute resources are homogeneous. This assumption allows us to test a resource allocation strategy in which the order of task assignments is equivalent to the order of task completion. It is straightforward to extend this strategy to account for tasks with various execution times and resources with various capabilities (see Section 4). Note that in this work we currently ignore issues pertaining to application data movement and location: we focus solely on the steerability of the application. We have addressed data location issues in our previous work [11, 12] and developed a number of scheduling heuristics. We will integrate this work with those heuristics in future work.

Another assumption implied here, is that the execution time of all function evaluations is known a-priori. In other words, given a point in the parameter space, we assume that we know how much computation is needed for evaluating the objective function at that point. This assumption holds for MCell, and is therefore justified for our work on the VI project. However, for other applications, there will undoubtedly be some degree of uncertainty. In addition, prediction errors for task execution times are often due to dynamic changes in resource load. This is especially common in Grid environments where resources are distributively owned and shared among users. These considerations imply that resource allocation decisions will be based on

possibly inaccurate predictions for task execution times. A number of authors, including ourselves, have investigated the impact of such prediction inaccuracies on scheduling, and at the moment we leave this issue for future work. Our assumptions in this work are substantial, but necessary to gain initial understanding of the principles of resource allocation for steerable parameter search applications.

3 The Virtual Instrument Project

3.1 Overview

The Virtual Instrument (VI) project is a collaborative effort between the University of California, San Diego, the Salk Institute, the Pittsburgh Supercomputing Center, the University of Tennessee, Knoxville, and the University of California, Santa Barbara. The overall goal of the project is to provide a Grid execution environment for the MCell application. MCell [25, 24, 35, 34] is a computational biology simulation framework that is currently used by neuroscientists to study diffusion and chemical reactions of molecules in living organisms. An MCell run typically consists of multiple repetitions of a simulation at each of a large set of parameter value combinations. Therefore, MCell fits the Parameter Sweep model. Even though MCell provides the basis for running simulations, its capabilities are currently limited in terms of scale, ease-of-use, and interactivity. The objective of the VI project is to address those three limitations in order to enable MCell simulations at an unprecedented scale. In addition, we wish to enable a new class of MCell simulations: parameter space searches. Therefore, the VI software should allow for easy deployment of large-scale MCell simulations on Grid resources, should enable interactive user-directed steering, and should use appropriate scheduling strategies to achieve high performance. Finally, the VI must provide a framework in which MCell users can build their simulations and mine the simulation results via a simple graphical user interface.

In our earlier work, we developed a generic environment for scheduling and deploying Parameter Sweep Applications: APST [12]. APST is used in production for MCell and for a number of other applications for medium- to large-scale parameter sweep runs. Because it is generic, APST fails to address most of the MCell-specific limitations listed above. When designing the VI architecture we were able to learn from and improve on APST's principles, while taking into account specific ways in which neuroscientists develop, run, and analyze MCell simulations.

3.2 VI Software Architecture

The Virtual Instrument software is constructed of three principle components: a software daemon that coordinates Grid resource usage; a user interface for users to build, run, monitor, stop, and analyze the results of MCell simulations; and a database for bookkeeping. These components can run on separate machines, while application tasks and data live on widely distributed Grid resources.

Figure 3 depicts the interactions of the VI Daemon, the VI Interface, and the VI Database. The VI Daemon interacts with Grid resources and services to start and control remote jobs, move data between distributed storage locations, and monitor resources. Grid resources are depicted on the right-hand side of the figure with running MCell processes. The Daemon embeds a scheduler that allocates MCell tasks to resources; this paper focuses on a resource allocation strategy that is implemented as part of that scheduler. The Daemon uses the VI Database to store information such as the available resources, the user-defined specifications of running MCell projects, and the status of these running projects, including their pending tasks. The Database also stores MCell output that can be visualized and analyzed by the user and used to steer further simulations. All raw and intermediate MCell output is left in place in Grid storage, as depicted on the right-hand side of the figure. The VI Interface allows the user to initiate, run, and steer MCell simulations. In addition, the VI Interface can invoke OpenDX [30] so that the user can visualize simulation results. All details on the VI software design are available in [9].

At the moment, the VI software consists of approximately 20,000 lines of C++ and is heavily multi-threaded. We opted for MySQL [27] to implement the VI Database as it is very fast, is well accepted by the Linux community and provides a standard API. The VI Daemon uses Ssh and Globus's GRAM [14] for starting/monitoring remote jobs, Scp and GridFTP [2] for moving application data onto the Grid. We have developed similar interfaces to a number of Grid services as part of the APST project. We will integrate them in future releases of the VI project. The VI Interface is about to be completed, and at the moment we provide a text-only user interface for evaluation purposes. A beta version of the VI software was released to a limited number of MCell users/developers in February 2002 for evaluation and comments. More information can be found on the project's Webpage at [40].

3.3 Initial VI Scheduling Approach

We focus on the parameter space search scenario as defined in Section 2.2, where users assign levels of importance to regions of the parameter space. As the VI scheduler allocates compute resources to each region, it makes intuitive sense for it to try to complete more promising regions sooner in the hope of finishing the search early. The simplest way in which this can be achieved is to evaluate parameter space points in the order of importance of their

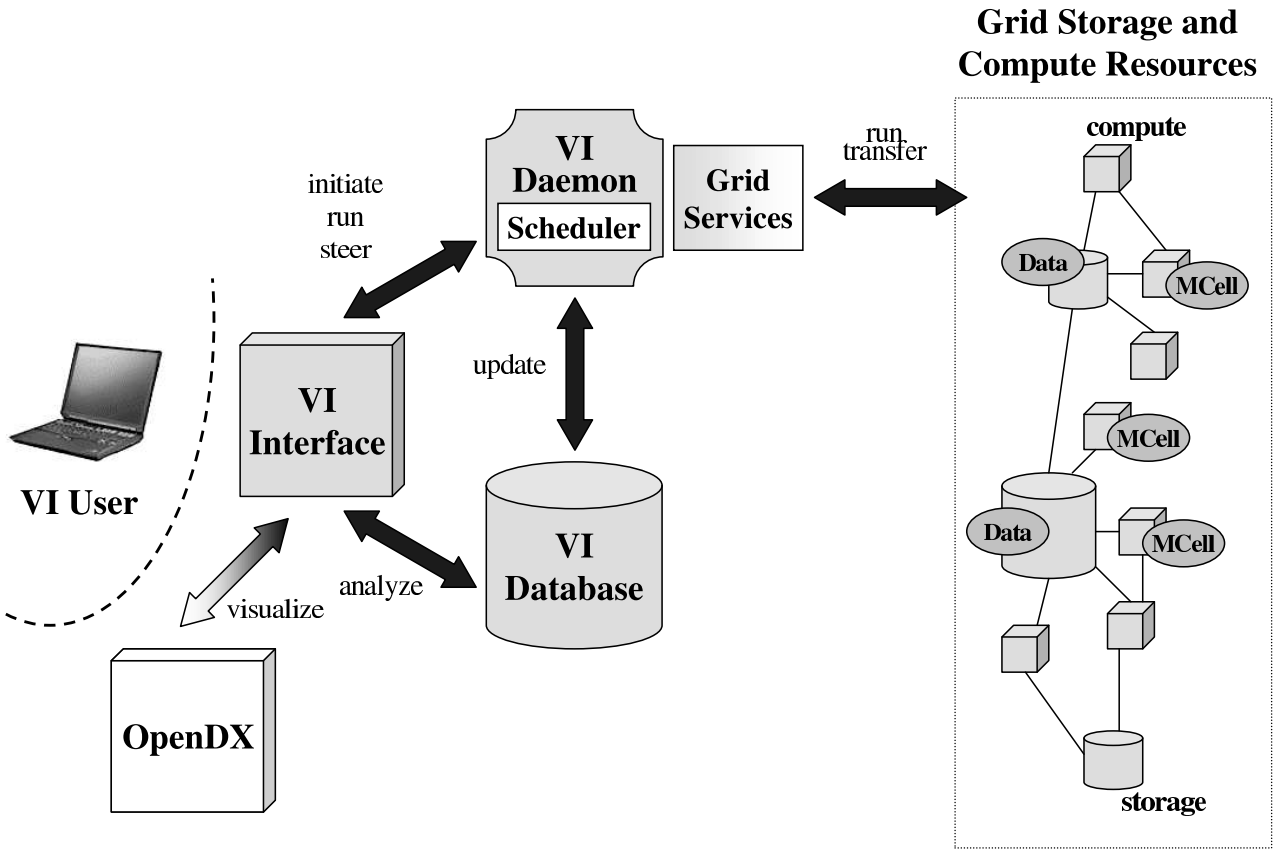

Figure 3. The Virtual Instrument: the three main components are the VI Daemon, the VI Database, and the VI Interface.

region. When a compute resource becomes available, the scheduler assigns it the most important available parameter space point for computation. As the search proceeds, the user or user agent identifies new regions, which are inserted into the scheduler's prioritized list of parameter space points. This strategy is straightforward to implement; no sophisticated job control is required, since computations all run to completion after their launch.

Note that the simple algorithm we just described is only valid when compute resources are homogeneous, and when the evaluations of parameter space points all require the same amount of computation. These are two of the assumptions that we use in this paper, as stated in Section 2.3. The rationale of the algorithm is that tasks complete in order of their respective importance levels; when resources and tasks are heterogeneous, it is not always possible to respect such strict ordering. In that case, it is straightforward to allocate resources in a way that best approximates that ordering. We will discuss this issue in an upcoming paper.

3.4 Experimental Evaluation

To evaluate the scheduling strategy described above we implemented the search algorithm presented in Section 2.2 to simulate a user-directed search. The implementation behaves as an agent which interacts with VI software on behalf of a user. We used the Meteor cluster deployed at the San Diego Supercomputer Center as part of the NPACI Rocks project [29]. That cluster consists of 90 dual 733-900MHz Pentium PCs interconnected via Myrinet. Using several

subsets of that cluster, we performed the following experiments. We simulated the user-directed search for values of σ ranging from 10^{-2} and 10^{6}, and for $maxTrials = 32$. Our choice for the value of $maxTrials$ stems from the fact that we wanted to simulate a thorough search strategy. The simulation results that will be presented in Section 4.2 will justify that choice further. We then performed back-to-back runs of the VI software with two different resource allocation strategies: (i) random resource allocation, where tasks are not sorted in any particular order; (ii) resource allocation as presented in the previous section, where tasks are sorted. We compute the time elapsed until the search finds a value which is within a small threshold of the global minimum (recall that the global minimum of the Griewank function is 0). For these experiments we used a threshold value of 0.01.

Overall, our results were inconclusive. It seems that the naive task ordering performed by the scheduling strategy described in Section 3.3 does not lead to clear improvements for many values of σ. However, we observed a few interesting features in our results. Those features suggest that taking into account levels of importance for resource allocation may be promising. For instance, for large values of σ, we observed marginal to large improvements due to task ordering in some cases. However, this is not sufficient to make a good case for our initial resource allocation strategy as we seek to provide improvements across the board. Therefore, we investigated a more sophisticated resource allocation strategy that needs to be implemented as part of the VI scheduler. That strategy is the object of the next section.

4 Priority-based scheduling

4.1 Task Priorities

In the scheduling approach described in section 3.3, the scheduler sorts new tasks that need to be dispatched according to their parent values. Tasks with higher levels of importance are executed earlier and therefore more promising search regions produce results sooner. However, in addition to the order in which tasks are dispatched, we claim that an important factor impacting the performance of the search is the **fraction of resources** allocated to search regions. Thus, we conjecture that a resource allocation strategy that assigns fractions of compute resources to regions based on their levels of importance may reduce overall search time. One intuitive rationale is that letting several tasks share compute resources concurrently may lead to a broader exploration of the search space, in a shorter period time.

To test this conjecture, we developed a model where different fractions of compute resources can be allocated to tasks. We then used this model to evaluate the effectiveness of several resource allocation strategies that map regional levels of importance to resource fractions. We have implemented a simulator that uses the Simgrid toolkit [8, 33] to simulate the execution of the search procedure of Section 2.2 with resources that can be continuously divided among tasks. We describe this approach in the next section.

4.1.1 Simulation Model

In Section 3.3, our computing platform consisted of a number of individual compute resources, and each resource was running only one application task at a time. In this section, we present results for an ideal platform model that makes it possible to associate fractions of resources to tasks. The ideal platform consists of a single, "continuously partitionable" processor. In other words, it is possible to assign any fraction of the computing platform to a given task. Larger resource fractions correspond to more computing power enabling a task to complete earlier. As we will see in our results, this model allows us to gain basic understanding of the potential of priority-based scheduling for parallel searches.

We make the initial assumption that it is possible to achieve fine-grain, accurate job control over a distributed computing platform such as the Grid. Larger fractions of the compute platform allocated to a task could be interpreted in practice as:

- more time slices assigned to a task sharing a processor with other tasks and/or,
- longer time slices and/or,
- task allocated to faster processor and/or,
- task allocated to less loaded processor.

However, our model is ideal in that we ignore the overhead of such fine-grain job control, which we discuss in Section 4.3. Therefore, our results provide us with a "best-case" scenario.

4.1.2 Resource Allocation Strategies

The resource allocation strategies we investigate are *priority-based*: tasks within search regions with higher level of importance are assigned higher priorities. The higher the task priority, the larger the fraction of resources that will be allocated to the task. Let us illustrate how priority-based scheduling could be easily employed for the user-directed search described in Section 2.2.

The user identifies regions and assigns them levels of importance. For instance, if the user has identified 3 regions that should be explored and assigned levels of importance 2, 2, and 1, then the first two regions should each get 40% of the resources, and the last region should get 20%. Our approach is to assign a priority to each point of the parameter space, corresponding to the level of importance of the

region to which the point belongs. If each of the current n points being computed has a priority $Priority_i$, then point i should get a fraction of the entire compute resource equal to $Priority_i / \sum_{j=1}^{n} Priority_j$. This allows many tasks to use the compute resource concurrently, but at different rates.

In our simulated user scenario we seek to minimize the Griewank function (see Section 2.1). Therefore high task priorities are associated with low objective function values. The priority of a task in computed according to its *parent value* (recall from Section 2.2 that the search strategy proceeds in generations – new tasks are offsprings of parameter space evaluations in a previous search generation). We investigate the relationship between parent values and task priorities by using a conventional no-priority policy:

Policy 0 – **No-priority** policy:
$$Priority_{task} = 1,$$

as a basis for comparison with the following three priority-based policies:

Policy 1 – Inverse **linear** policy:
$$Priority_{task} = Value_{parent}^{-1},$$

Policy 2 – Inverse **sub-linear** policy:
$$Priority_{task} = Value_{parent}^{-1/2},$$

Policy 3 – Inverse **super-linear** policy:
$$Priority_{task} = Value_{parent}^{-2},$$

where $Priority_{task}$ is the priority assigned to a task, and $Value_{parent}$ is the parent value (in our case it is a Griewank value).

4.2 Simulation Results

In the simulation results presented here, we assumed that each function evaluation would require 1 second of computation if it could run on the entire computing platform by itself. Therefore, if a task, due to its priority, is assigned 1% of the compute resources, it takes 100 seconds to complete.

Figure 4 show results comparing the use of the four priority policies for several values of σ and of $maxTrials$. The search is run until a Griewank value below some threshold (in this case 0.01) is found. The average time to reach this threshold is plotted on the y-axis with a logarithmic scale. We show plots for $maxTrials$ values of 1, 4, 16, 32 and 64. On each plot, σ is on the x-axis and we show bars for the search time for the four priority policies. Since our search algorithm is randomized, each data point is obtained as an average over 100 repetitions.

We present speed-up results in Tables 1, 2, and 3. Those results show the ratios between the overall search time with no priorities and the overall search time with the linear, sub-linear, and super-linear priority policies. Therefore values over 1.0 mean that priority-based resource allocation leads to a performance improvement. We show those values in boldface in the tables.

The first observation we can make is that the use of priorities leads to drastic improvements for large σ for all three priority-based policies. Furthermore, the sub-linear policy ($Priority_{task} = Value_{parent}^{-1/2}$) appears to be very promising – if $maxTrials > 4$, it performs better than the no-priority policy, even when σ is small (see Table 2).

For $maxTrials \leq 4$ the local search over a region is less accurate, since local basins are sampled with lower density and it is harder to detect local minima. Therefore, biasing search regions with priorities might not be as effective as it is when $maxTrials$ is larger. Larger $maxTrials$ provides more thorough local searches and hence more accurate information about local minima. In this case the biasing property of priority strategies improves search performance more effectively.

One curious factor is that for small σ the linear and super-linear priority policies performed much worse than the no-priority policy (see Tables 1 and 3). We suspect that these two policies *over-bias* the search regions. When σ is small the values at the bottom of the local basins of attraction become similar (see Figure 1), making it more difficult to use objective function evaluations to interpret how close the search process is to the global minimum. Therefore we think that in this context many low objective function results erroneously indicated a "false" proximity to the global minimum. The linear and super-linear policies increase the priority in the neighborhood of such misleading results more drastically than the sub-linear policy, thereby wasting a lot of computational power which could be used for exploring other (more useful) search regions.

4.3 Discussion

We observed in our experiments that the initial resource allocation strategy presented in Section 3.3 did not lead to consistent improvements across the board, a result that we also verified in simulation (results omitted in this paper). We then conjectured that a resource allocation approach that allocates fractions of the compute resources to tasks concurrently would be effective. We confirmed this conjecture in simulation for a sub-linear priority policy when the search is marginally thorough (i.e. for $maxTrials > 4$).

However, our simulation results were obtained with an ideal platform model. Indeed, we assumed fine-grain job control with no overhead, which allows the allocation of fractions of resources precisely corresponding to priorities dictated by the resource allocation strategy. In practice, such a strategy can be implemented by assigning time slices of compute resources to tasks. This could be done at the level of the operating system (e.g. with process priorities).

$maxTrials$	σ								
	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}
1	0.94	0.96	0.93	0.79	**1.03**	0.51	0.65	**1.05**	**1.68**
4	0.69	0.25	0.60	0.13	**7.90**	**9.66**	**6.74**	**10.79**	**22.83**
16	0.40	0.56	0.18	0.13	**7.38**	**7.85**	**6.84**	**11.46**	**22.26**
32	**1.18**	0.55	0.26	0.16	**7.31**	**6.71**	**5.90**	**10.05**	**19.01**
64	**1.36**	0.77	0.25	0.17	**7.35**	**6.12**	**5.64**	**9.69**	**18.21**

Table 1. Speedup due to priority-based scheduling where $Priority_{task} = Feedback_{parent}^{-1}$.

$maxTrials$	σ								
	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}
1	0.84	**1.03**	0.93	0.76	0.91	**1.06**	0.92	**3.43**	**4.65**
4	0.75	0.47	0.36	0.61	**4.12**	**5.77**	**8.44**	**13.96**	**27.34**
16	**1.53**	**2.15**	**1.22**	**1.71**	**4.08**	**5.12**	**7.83**	**13.55**	**24.70**
32	**1.15**	0.97	**1.14**	**1.69**	**4.05**	**5.00**	**7.80**	**13.73**	**25.07**
64	**1.37**	**1.05**	**1.25**	**1.69**	**4.06**	**5.00**	**7.76**	**13.71**	**25.03**

Table 2. Speedup due to priority-based scheduling where $Priority_{task} = Feedback_{parent}^{-1/2}$.

$maxTrials$	σ								
	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}
1	0.99	0.95	0.94	0.87	0.98	0.36	0.46	0.72	**1.08**
4	0.31	0.14	0.10	0.05	**3.27**	**3.95**	**2.83**	**5.69**	**12.76**
16	0.20	0.22	0.09	0.06	**1.60**	**3.04**	**3.37**	**7.07**	**14.93**
32	0.81	0.21	0.08	0.04	**3.49**	**3.65**	**3.26**	**6.88**	**14.33**
64	0.44	0.16	0.08	0.05	**3.61**	**3.66**	**3.13**	**6.61**	**14.03**

Table 3. Speedup due to priority-based scheduling where $Priority_{task} = Feedback_{parent}^{-2}$.

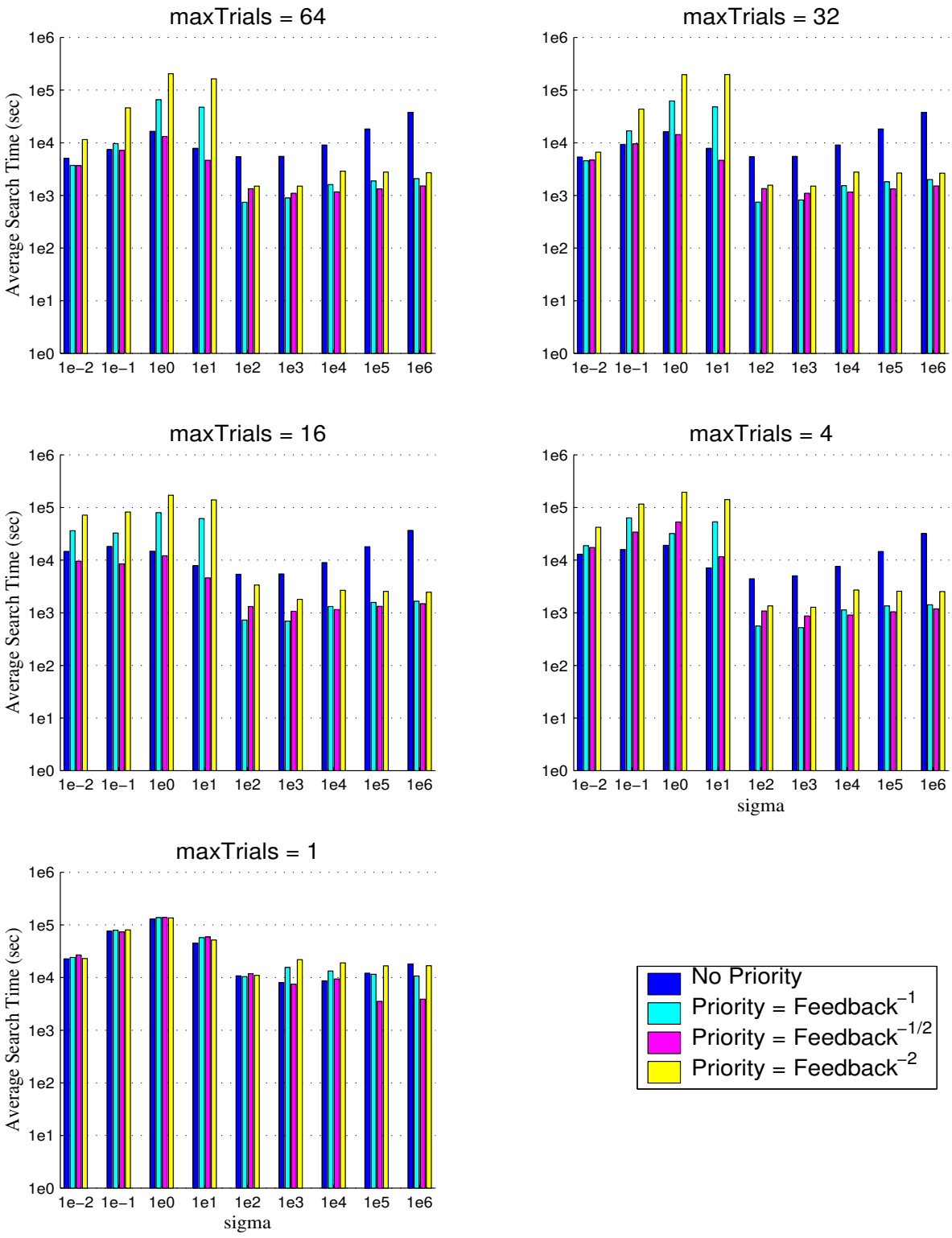

Figure 4. Average simulated search times vs. σ for the four priority policies and for $maxTrials = 1, 4, 16, 32, 64$. Both axes are logarithmic scales.

This is possible if tasks do not consume large amounts of RAM and the behavior of the operating system scheduler is well understood, which is debatable with current systems. Alternatively, the VI could explicitly context-switch tasks on resources by checkpointing/resuming. This can be done at various granularity levels. The lower the granularity, the closer the schedule will be to the one evaluated in our simulation results, at the cost of higher overhead. We will therefore develop a simple cost model for the context-switching overheads and perform simulations in order to answer two following questions. What is the best granularity for explicit context-switching? How does the overall performance of the search compare with a simple non-prioritized resource allocation? In other terms, how much of the improvement due to using priorities, as shown in our simulation, will withstand the context-switching overhead?

Note that in order to implement priority-based resource allocation in a practical setting, it is required that application tasks be checkpointed on-demand. MCell provides on-demand checkpointing already, which will make it possible for the VI to context-switch MCell tasks during a search. We will present experimental results obtained with MCell in a future paper.

5 Related Work

Our work is related to parallel search algorithms [5, 13, 37, 6, 7, 18, 22, 26]. However, our work focuses on the improvement of the performance of search algorithms *from the perspective of computing resource allocation*. Land [21] and Hart [18] investigate search algorithms that combine local and global search methodologies. Their search algorithm biases the search in favor of more promising regions of the parameter space. Our approach is complementary to their work since we use biasing strategies to determine resource allocations.

This work is also related to a number of computational steering efforts [31, 20, 38, 39, 17]. The VI project itself is related to projects that provide software for deploying large scale application onto the Computational Grid. In particular, it is related to projects that target parameter sweep applications [12, 1, 41]. The VI software builds on these efforts in order to enable parameter space searches.

6 Conclusion and Future Work

In this paper we have investigated scheduling approaches that can improve the performance of user-directed parallel parameter space searches on distributed computing platforms. Our goal was *not* to develop novel search/optimization algorithms, but rather to develop techniques to allocate appropriate compute resources to tasks of a user-directed search. We instantiated a search strategy that is representative of user-directed searches and that we used to simulate a range of user behaviors. Our fundamental idea for resource allocation is that "promising" regions of the parameter space should be explored more intensively. In our model, promising regions are defined as ones that have high "levels of importance". This work was done in the context of the Virtual Instrument project which we introduced briefly in Section 3. As part of that project we implemented and evaluated a simple resource allocation strategy that sorts application tasks to be completed by their level of importance and assigns them to available compute resources in order. We then conjectured that a resource allocation approach that defines priorities to allocate fractions of compute resources to tasks concurrently should be more effective. We verified that conjecture in simulation and thereby made a good case for priority-based resource allocation.

To further validate our approach we will perform experiments in a broader context: we will evaluate priority-based resource allocation for several other representative search strategies over a variety of objective functions. We will investigate the questions raised in Section 4.3 and develop a model for the overhead involved for context-switching of application tasks. We will then re-evaluate our approach both in simulation and with the VI software using MCell as a case-study. We will also evaluate the use of priority-based resource allocation for a number of platform models that are more realistic than the ones used in this work. Our ultimate goal is to produce a version of the VI software that improves the performance of parameter space searches while using a Computational Grid platform.

Acknowledgement

We wish to thank the members of the MCell and the Virtual Instrument team for their help. We are also grateful to the NPACI Rocks team for allowing us to use their cluster for experiments. Finally, the authors wish to thank Rick Belew, Charles Elkan, and Andrew Kahng for their help with understanding search applications.

References

[1] J. Abramson, D. Giddy and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico*, pages 520–528, May 2000.

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chernavak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing*, 2001.

[3] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 2990.

[4] APST Webpage. `http://grail.sdsc.edu/projects/apst`.

[5] T. Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[6] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical report, Carnegie Mellon University, 1995.

[7] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proc. of the 12th Intern. Conf. on Machine Learning*, 1995.

[8] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[9] H. Casanova, T. Bartol, F. Berman, A. Birnbaum, J. Dongarra, M. Ellisman, M. Faerman, E. Gockay, M. Miller, G. Obertelli, S. Pomerantz, T. Sejnowski, J. Stiles, and R. Wolski. The Virtual Instrument: Support for Grid-enabled Scientific Simulations. Technical Report CS2002-0707, Dept. of Computer Science and Engineering, University of California, San Diego, June 2002.

[10] H. Casanova, T. Bartol, J. Stiles, and F. Berman. Distributing MCell Simulations on the Grid. *The International Journal of High Performance Computing Applications*, 14(3):243–257, 2001.

[11] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.

[12] H. Casanova, G. Obertelli, H. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the Grid. In *Proceedings of Supercomputing*, November 2000.

[13] J. A. Chandy and P. Banerjee. Removing the genetics from the standard genetic algorithm. In *Proc. of the 12th Intern. Conf. on Machine Learning*, 1995.

[14] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[15] D.Aldous and U. Vazirani. Go with the winners' algorithms. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, 1994.

[16] R. Durbin, S. Eddy, A. Krogh, and G. J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, A tutorial introduction to hidden Markov models and other probabilistic modelling approaches in computational sequence analysis*. Cambridge University Press, 1998.

[17] G. Geist, J. Kohl, and P. Papadopoulos. CUMULVS: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, 1997.

[18] W. E. Hart. *Adaptive Global Optimization with Local Search*. PhD thesis, University of California, San Diego, 1994.

[19] W. E. Hart and R. K. Belew. *Adaptive Individuals in Evolving Populations, Models and Algorithms*, chapter Optimization with Genetic Algorithm Hybrids that Use Local Search, pages 483–494. Addison-Wesley Publishing Company, Inc., 1996.

[20] D. Jablonowski, J. Bruner, B. Bliss, and R. Haber. VASE: The Visualization and Application Steering Environment. In *Proceedings of SuperComputing 1993*, pages 560–569, 1993.

[21] M. W. S. Land. *Evolutionary Algorithms with Local Search for Combinatorial*. PhD thesis, University of California, San Diego, 1998.

[22] R. H. Leary and J. P. K. Doye. Tetrahedral global minimum for the 98-atom lennard-jones cluster. *Physical Review E*, 60(6), December 1999.

[23] A. Majumdar. Parallel Performance Study of Monte-Carlo Photon Transport Code on Shared-, Distributed-, and Distributed-Shared-Memory Architectures. In *Proceedins of the 14th Parallel and Distributed Processing Symposium, IPDPS'00*, pages 93–99, May 2000.

[24] MCell Webpage at the Pittsburgh Supercomputer Center. `http://www.mcell.psc.edu`.

[25] MCell Webpage at the Salk Institute. `http://www.mcell.cnl.salk.edu`.

[26] Z. Michalewicz and D. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.

[27] MySQL Webpage. `http://www.mysql.org`.

[28] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. Humphrey, A. Fox, and A. Grimsaw. Studying Protein Folding on the Grid: Experiences using CHARM on NPACI Resources under Legion. In *Proceedings of the Tenth IEEE International Symposium on High- Performance Distributed Computing (HPDC- 10)*, 2001.

[29] NPACI Rocks Webpage. `http://rocks.npaci.edu`.

[30] OpenDX Webpage. `http://www.opendx.org`.

[31] S. Parker, M. Miller, C. Hansen, and C. Johnson. An integrated problem solving environment: The SCIRun computational steering system. In *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31), vol. VII*, pages 147–156, January 1998.

[32] S. Rogers and D. Ywak. Steady and Unsteady Solutions of the Incompressible Navier-Stokes Equations. *AIAA Journal*, 29(4):603–610, Apr. 1991.

[33] Simgrid Webpage. `http://grail.sdsc.edu/projects/simgrid`.

[34] J. R. Stiles, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Monte Carlo simulation of neurotransmitter release using MCell, a general simulator of cellular physiological processes. In J. M. Bower, editor, *Computational Neuroscience*, pages 279–284, New York, NY, 1998. Plenum Press.

[35] J. R. Stiles, D. Van Helden, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Miniature endplate current rise times <100 μs from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proc. Natl. Acad. Sci. U.S.A.*, 93:5747–5752, 1996.

[36] A. Torn and A. Zilinskas. *Global Optimization*, volume 350 of *Lecture notes in computer science*. Springer-Verlag, 1989.

[37] P. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[38] J. Vetter and K. Schwan. PROGRESS: A Toolkit for Interactive Program Steering. In *Proceedings of the 1995 International Conference on Parallel Processing*, pages 139–149, 1995.

[39] J. Vetter and K. Schwan. High Performance Computational Steering of Physical Simulations. In *Proceedings of IPPS'97*, pages 128–132, 1997.

[40] Virtual Instrument Webpage. `http://grail.sdsc.edu/projects/vi_itr`.

[41] M. Yarrow, K. McCann, R. Biswas, and R. Van der Wijngaart. An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid. In *GRID 2000, Bangalore, India*, December 2000.