Result Reuse in Design Space Exploration: A Study in System Support for Interactive Parallel Computing [*]

Siu-Man Yau[*], Kostadin Damevski[†],
Vijay Karamcheti[*], Steven G. Parker[†], Denis Zorin[*]
[*] Courant Institute of Mathematical Sciences, New York University
[†] Department of Computer Science, University of Utah

Abstract

This paper presents a system supporting reuse of simulation results in multi-experiment computational studies involving independent simulations and explores the benefits of such reuse. Using a SCIRun-based defibrillator device simulation code (DefibSim) and the SimX system for computational studies, this paper demonstrates how aggressive reuse between and within computational studies can enable interactive rates for such studies on a moderate-sized 128-node processor cluster; a brute-force approach to the problem would require two thousand nodes or more on a massively parallel machine for similar performance. Key to realizing these performance improvements is exploiting optimization opportunities that present themselves at the level of the overall workflow of the study as opposed to focusing on individual simulations. Such global optimization approaches are likely to become increasingly important with the shift towards interactive and universal parallel computing.

1 Background

The growing availability of low-cost, universal parallel computing resources, including commodity processors with 4-16 cores, GPUs with support for general-purpose computing and heterogeneous multicore chips, enables small groups of researchers or even individual researchers to have dedicated access to large amounts of compute power. This availability is expected to change usage models for parallel computing, shifting the traditional emphasis on batch calculations to interactive applications. Such change requires reconsidering how parallel software systems are structured and how resources have to be managed for new interactive workflows.

Our ongoing work with the SimX computational study system [27] provides a representative platform in which to investigate these issues. Recognizing that computer simulation has become an integral part of the scientific method, SimX supports a scientific exploration process that manifests itself as *computational studies* built out of multiple *computational experiments* corresponding to individual runs of simulation software. Examples of such studies range from exploration of design spaces in engineering to molecular simulations for drug design.

As an example of the different considerations that come into play, the performance criterion driving the design of the SimX system is the need to provide meaningful results to the researcher at a timescale that permits the researcher to interactively drive the exploration process in studies involving tens of thousands of experiments. Satisfying this requirement requires managing resources at the level of the overall computational study instead of individual experiments. Unfortunately, with the few exceptions listed in Section 5, prior research has not examined how one might improve the performance of entire studies under high levels of parallelism.

This paper describes SimX support for study-level resource management using the specific context of Design Space Exploration (DSE) computational studies. DSEs, used in various disciplines such as automotive design [20], mechanical engineering [25], electrical engineering [11], and medicine [19], rely on multiple executions of a simulation code using different input parameters to discover a region of interest. Each execution would simulate, for example, the dynamics of a car crash under variously-shaped body frames, the deformation and tip force of an bimorph actuator built using piezoelectric materials of different shapes and

[*]Supported by a collaborative NSF grant comprising CSR-0615225, CSR-0614770, and CSR-0615194.

material properties, the performance of a microprocessor under different designs, or the current delivered to the heart by variously-configured defibrillators. The objective of the study is to identify the 'best' of these simulations, as identified by restrictions on the ranges of input parameters and/or output performance metrics. The interactivity requirements come about because the researcher usually starts off with only a fuzzy set of requirements, which get refined as the study unfolds.

The common theme underlying SimX support for DSE studies is the aggressive *reuse* of information provided by previous simulations to guide the current action of the system. SimX exploits reusable information across the entire spectrum of its response to a researcher's changing definition of the study parameters — from deciding how best to explore a portion of the design space all the way down to the execution of a single simulation. This information helps in a number of different ways, from guiding the search to allow exploration of design spaces whose size is otherwise too high for the computational resources assigned to handle the problem, to reusing internal simulation state to jump start the execution of a new simulation. An interesting aspect of SimX's reuse support is the fact that it trades off parallelism for speed: nominally, the simulations making up a DSE are independent of each other; yet, introducing dependencies amongst them improves the overall performance of the study.

We present the realization of these mechanisms in a SCIRun-based implementation of the SimX system [28], and demonstrate how, by exploiting re-use, one can achieve response times on the order of tens of seconds under various scenarios in an interactive defibrillator computational study using only a 128-processor cluster (with leftover capacity); a brute-force approach would require two thousand nodes or more on a massively parallel machine for similar performance.

The remainder of this paper is organized as follows. Section 2 describes how design space exploration is supported by our system and presents the defibrillator design application [19], which will serve as a running example throughout the paper. A taxonomy of different types of reuse and their implementation are presented in Section 3. In Section 4, we evaluate the effectiveness of various types of reuse using the defibrillator design study running on the SimX system. Finally, we discuss related work and conclude in Sections 5 and 6.

2 Design Space Exploration with SimX/SCIRun

The design space exploration platform described in this paper combines the SimX System Software for Interactive Multi-Experiment Computational Studies [27], with the SCIRun problem-solving environment [18]. SimX is realized as a set of SCIRun components, which augments the standard set and together supports *design space exploration* computational studies, discussed in additional detail below. The SimX/SCIRun system is designed to enable interactive computational studies on small- and medium-sized (32 to 128 nodes) clusters.

Defibrillator design. We use a defibrillator design computational study as our primary test example. This study is a typical design space exploration (DSE) problem common in several engineering applications. An individual computational experiment [19, 13] models the application of defibrillator stimuli and resulting propagation of electric current across a human torso.

The goal of the study is to find positions of two defibrillator electrodes on the surface of the torso and a voltage difference between electrodes, which together form a five-dimensional *design space* that balances several conflicting *performance metrics*:

1. minimize percentage of heart tissue that experiences current flow above a damage threshold;
2. maximize the percentage of heart tissue for which the electric current is above the activation threshold needed for defibrillation; and
3. maximize the uniformity of the potential gradient in the heart tissues, defined as the ratio of the maximum gradient found in the heart to the average gradient in the heart.

We refer to the three-dimensional space spanned by these performance metrics as the *performance space*. Performance metrics can be *parameterized* to represent slightly different problems. E.g., in order to model defibrillation for a patient with fat tissues surrounding a heart, the activation and damage thresholds can be set to a higher level, making it more difficult to activate and damage the heart tissues.

The *simulation code* receives as its input the electrode positions and voltages, and solves a 3D Poisson equation solved to obtain the potential distribution in the body; the potential gradient is used to obtain the electric current. Then the *performance evaluation code* takes the result of the simulation code, and uses the performance metric parameters to find the performance metrics for that experiment.

Typically, multi-objective optimization problems of this type [15, 25] aim to identify the set of input parameters resulting in *Pareto-optimal* designs (Pareto frontier) i.e. those for which improving one performance metric can only be achieved at the expense of another, e.g., a defibrillator configuration is Pareto-optimal if activation cannot be improved while holding uniformity and damage level fixed, and vice-versa. The user applies additional, often imprecisely defined, criteria to pick a final design on or close to the Pareto frontier. One criterion the user may use can be the *admissible region* - a sub-region in the performance space where any designs that achieve performance metric that fall into that space are disregarded, e.g., defibrillator settings that activate less than 5% of the heart, no matter how well they perform in other performance metric, are automatically discarded.

Interactive design space exploration. The SimX/SCIRun system provides an interactive environment for exploring the design space and identifying relevant Pareto-optimal configurations.

In our model of interactive design space exploration, the user, using the partial results presented by the system, interactively adjusts how the problem is specified (see Figure 2 for an example workflow). Specifically, the user can adjust the following parameters: (1) *performance metric parameters* (e.g., activation threshold and damage threshold); (2) *admissible regions* for performance metric (e.g., minimal acceptable percentage of tissue activated); (3) region of exploration in the design space (e.g., a 2D subset of the design space where the position of one of the electrodes and the shock voltage are fixed, and the area on the torso where the other electrode can be placed is bounded).

We regard each change of parameters by the user as a new *study*, i.e., each study is defined by a region in design space, a set of performance metric parameters, and a region in performance space (the admissible region). For each study, the system computes and visualizes the Pareto frontier, reusing the computation from previous studies to the maximal extent possible.

SimX/SciRun system overview. The overall structure of the SimX/SCIRun system (Figure 2) at the process level includes a *manager* process, multiple *worker* processes and a fixed number of *S̲patially-I̲ndexed S̲hared O̲bject L̲ayer (SISOL) servers*, which provide a shared state repository used by the system. The manager and worker processes are assembled from components provided by SCIRun and SimX. SISOL servers are entirely application independent.

To build an application within the SimX/SCIRun framework, the user needs to design a visualization/user interaction module, hosted in the manager, a simulation module, hosted in the workers, and optionally a *sampler* module, also hosted in the manager. The sampler module is responsible for deciding, based on the current set of results, which design-space point will be evaluated next. Unlike visualization and simulation modules, this module is less application-specific and the same module can be used for a broad range of problems. SimX currently provides two sampler modules: a simple parameter sweep sampler and an adaptive sampler, described in additional detail in Section 3.2.

The SCIRun problem solving environment provides a variety of high-level components facilitating implementation of simulation and visualization/user interaction (UI) modules, the latter at both the level of individual simulations as well as the entire design space exploration (Some examples of the UI elements for the defibrillator design example can be seen in Figure 2.) SimX plugs in naturally into the SCIRun architecture by exposing its interfaces in the form of SCIRun modules that can be directly hooked up as required with the simulation and UI modules.

During design space exploration, once the user specifies a study parameter, the SimX manager obtains from the sampler a set of design space points to explore and distributes them to workers along with information on performance metric parameter settings. Worker processes execute the simulations, optionally retrieving state about previously run simulations from the SISOL servers to support the reuse optimizations described in Section 3, and send the performance metrics evaluated back to the manager. Upon receiving the performance metrics, the manager registers the performance metrics with the sampler and obtains additional design space points to explore. The communication between manager and worker processes is accomplished through asynchronous interfaces; a worker can overlap invocation of a new simulation with communication of the prior simulation's results to the manager.

From an application programmer point of view, SimX functionality is exposed via a set of sampler and SISOL interfaces. User-supplied samplers need to implement the sampler interface, so that it can be called by the manager, the most important functions of which are `getNextPointToRun`, `registerResults`, and `getEvaluated`. The latter function retrieves design space points for which simulations

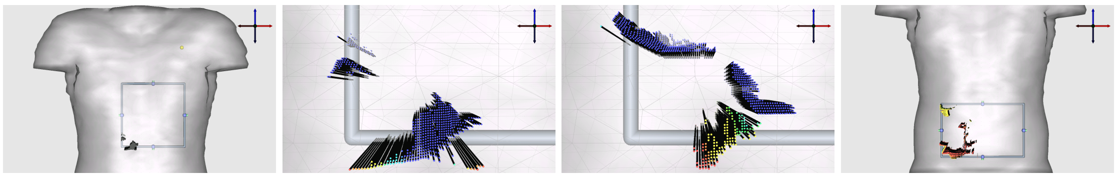

Figure 1. Interactive DSE: To begin, the user selects an area in front of the torso, sets the back electrode's position, and sets the shock voltage (far left). As the initial Pareto-optimal placements are discovered (center left, zoomed-in view), the user increases the activation threshold voltage, and the system responds by displaying the new Pareto optimal points (center right). The user can also select an entirely different area to explore (far right).

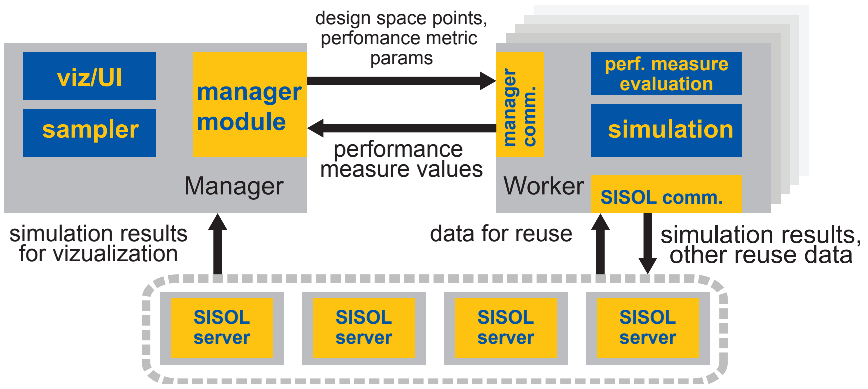

Figure 2. Overall SCIRun/SimX system structure; processes are shown in gray, SimX components in orange, application-specific components in blue.

were already performed with associated performance metrics.

SISOL gets its name from the fact that objects are named and indexed using spatial coordinates, a natural fit to the design space and performance metric elements that typically get stored in the layer. The SISOL interface, available to both the application-independent manager and worker modules as well as the application-specific sampler and simulation modules, supports the operations `Insert`/`Remove` for inserting/removing an object at a given location in the spatial database, `QueryClosest` for returning a list of object handles associated with nearby points in design space, and associated functions for retrieving and storing object data. Table 1 shows the high-level SISOL API.

These SISOL interfaces are currently implemented using a variable number of stand-alone data servers and a directory server; the architecture can be additionally scaled by partitioning the object namespace. The data servers store the actual objects, all in volatile memory, while the directory server maintains information about which spatial coordinate is stored on which data server. After an individual study is completed results are not discarded to enable inter-study reuse of the kind described in Section 3.2.

3 Reuse Opportunities in Design Space Exploration

In our initial experiments with the defibrillator design study, we quickly found that the default parameter-sweep sampler would yield study response times that were far from interactive, often requiring the user to wait tens of minutes or longer before receiving feedback that could drive refinement of the performance metrics. This led to an extended period of iterative experimentation, where we investigated several different approaches for bringing the study response times down to acceptable interactive rates (tens of seconds or lower). This section summarizes what we discovered to be the most effective approaches, all of which share the same underlying key idea: that of *maximally reusing information contained in previous simulation runs to substantially improve the study performance moving forward.*

SIGNATURE	FUNCTION
`int CreateSet(int setID, int typeID, int arity, double *weights, int capacity)`	Create object set of `arity` dimensions to store objects of type `typeID`. The `weights` array specifies a weighted Euclidean distance metric.
`int RegisterSet(int setID, void** objSet)`	Registers client as participant; retrieves object set metadata in `objSet`.
`void UnregisterSet(void* objSet)`	Unregisters client.

`void Insert(void* objSet, double* coords, void* obj)`
`void Remove(void* objSet, double* coords)`
	Insert/remove an object into/from the set.

`void* StartRead(void* objSet, double* coords)`
`void* StartWrite(void* objSet, double* coords)`
`void EndRead(void* objSet, double* coords, void* obj)`
`void EndWrite(void* objSet, double* coords, void* obj)`
	Start/end a read/write operation on an object.

`void QueryClosest(void* objSet, double* coords, int numToLookup, int* numRetrieved, double** retrCoords)`	Query for up to `numToLookup` closest neighbors

Table 1. Interface of the spatially-indexed shared object space layer (SISOL).

In general, the reuse of previous results benefits interactive design space exploration in two ways: by reducing the number of simulations that have to be performed for a given study and by reducing the work needed for a given simulation. We start by listing the broad types of reuse opportunities that were found to be most beneficial, and then describe the SimX/SCIRun system support that was needed to derive this benefit.

3.1 Types of Reuse

Intra-study reuse in the sampler module. In the course of a single study, simulation results can be used to guide design space sampling strategies, often yielding the same overall results while exploring only a tiny fraction of the overall design space. This observation manifests itself in the design of a smarter sampler module in SimX, the Active Sampler (an early version of which was described in [27]), which first performs a coarse parameter sweep over the region of design space to be explored, and uses the performance metric of those points to construct a coarse approximation of the Pareto frontier. It then refines the grid to explore the neighbors of those points, eliminating "unimportant" parts of the design space early in the study.

Note that the Active Sampler artificially introduces dependencies amongst the simulations explored in the study, by not issuing design space points corresponding to a finer resolution sweep until it receives results for the coarser sweep. This reduction in parallelism is justified by an overall benefit in study response times.

Inter-study reuse in the sampler module. If a user performs an action that causes the system to switch from exploring one study to another study (e.g., changes a performance measure parameter such as activation threshold), we can use the results from the first study to aid the second study. For example, the system can take the approximation of the Pareto frontier obtained in the first study to act as the initial Pareto frontier approximation for the second, reducing the total number of experiments needed. This reuse, beneficial in situations where the second study is a refinement of the first (as is typical for design space exploration), has the effect of replacing part or all of the coarse-grained Active Sampler exploration for the second study with results from the first study.

Inter-study result reuse in the manager module: performance metric reuse. Recall from Section 2 that the user inputs during design space exploration can consist either of changes in performance metric parameters, establishing new admissible performance space regions, or requesting exploration of a different design space region. In the latter two scenarios, it is possible that the *same* simulation be re-issued with the *same* performance metric parameters for a different study. When that happens, the manager module can simply look up the performance metrics from the list of previously-completed experiments, and thus cut down the number of simulations that need to be allocated to Worker processes.

Inter-study result reuse in the simulation modules: simulation result reuse. If the user changes only the performance metric parameters, it is possible that the *same* simulation be re-issued with *different* performance metric parameters for a different study. In those cases the worker process can lookup the simulation result from a list of previously-completed simulations instead of running the simulation code, and re-compute only the performance metrics. In the specific example of the defibrillator study, this corresponds to reusing the potential distribution values obtained as a solution to the 3D Poisson equation.

Intermediate result reuse in the simulation module. Even when *different* simulations are issued with *different* performance metric parameters, there are still reuse opportunities. This type of reuse heavily depends on the internals of the simulator code, and although applicable to other numerical simulations, is the least generic. The defibrillation simulation used in our experiments involves solving three linear systems, each corresponding to setting the boundary condition of the front electrode to the three surface mesh nodes nearest to it. Since many simulations set boundary conditions to the same mesh nodes, we store the result of each invocation of the solver, and the simulation code can looks up the solutions to skip solving those systems that had previously been solved.

Preconditioner reuse in the simulation module. If the solution of the exact linear system is *not* available, reuse is still possible. Each system is solved using an iterative solver with a preconditioner, a compact approximation to the matrix inverse. The result of a simulation from a nearby point in design space can be used to initialize the iterative solver, decreasing the number of iterations. If for different points in design space the system matrices are similar (e.g. nearby electrode placement on torso) or identical (same placement, but different voltage distributions), we can use a previously computed preconditioner, assuming it was saved along with the results.

As may be apparent, the levels of reuse range from generic to specific, with performance metric and simulation result reuse in the Worker process being most generic: any interactive multi-experiment study fitting into the formulation described in Section 2 can take advantage of them. Reuse in the sampler is specific to Pareto optimization DSEs, while the intermediate result reuse and preconditioner reuse require the internal simulator code to store and retrieve information specific to the application. As a result, the system support for reuse is also many-layered, as we describe below.

We note that although the specifics may differ somewhat, the overall notion of result reuse is broadly applicable beyond our context of computational studies. Any parallel workload, which involves the repeated execution of similar kinds of computation can benefit from a similar approach, where one focuses on the "delta" of computation that needs to be performed beyond what already exists as opposed to computing the result from scratch. Examples of domains where this strategy can be applied include physics simulations in 3D games, or periodic analytics over incrementally changing data sets in the financial and web contexts.

3.2 System Support for Exploiting Reuse

SimX exploits the different kinds of reuse by making heavy use of the sampler and SISOL interfaces introduced in Section 2. SISOL stores a history of the ongoing design space exploration; interestingly, this information is pure overhead but is justified by its use for reducing overall study run time.

Reuse support in the Manager process. The SimX Manager process incorporates support for inter- and intra-study sampler reuse, as well as performance metric reuse.

Each time a new study is created, the manager process creates a new sampler object to perform it. Old samplers are not deleted, as long as there is sufficient memory to store them. Each Active Sampler keeps an approximation of the Pareto frontier on the experiment results it has received, and refines the frontier as the study is successively refined. The frontier approximation is stored entirely in the manager process, as it is typically relatively compact.

Intra-study reuse is exploited in the design of the Active Sampler, which uses Pareto frontier approximations from coarse parameter sweeps to guide exploration over finer resolutions of the design space, thereby reducing the number of simulations that need to be run.

For *inter-study reuse in the sampler*, for each new study, the manager looks for a previously-performed study that is 'closest' to the new one (the maximum amount of overlap in region explored, or the minimum amount of change in performance metric parameter or fixed design space dimension value) and uses that study's Pareto frontier approximation to initialize the newly-created sampler. To accommodate this usage, the SimX sampler interface was generalized to support application-neutral specifications of design spaces and performance metrics associated with a sampler, and to enable proximity calculations between samplers. Note that the original sampler doesn't need to have completed its study - to be useful, all it needs to provide is a Pareto frontier approximation that is close enough to that of the new study.

To enable *performance metric reuse*, the manager process keeps a cache of all simulations performed and their results, i.e., a mapping of the <design space point, performance metric parameter> tuple to parameter space point. For most realistic studies, this mapping is small enough to be maintained in memory on the manager process. When the sampler issues an experiment, the manager process looks up this mapping first to see if a result is already available.

Reuse support in the Worker process. *Simulation result reuse* requires worker processes to store their simulation results into SISOL, and make them available for use by subsequent simulation worker processes. Spatial indexing used in SISOL is a natural fit for the object store: simulation workers use the design space point's coordinate to store their simulation results and check if the results for this design space points were already computed. Although not exercised by the results presented in this paper, SimX's SISOL layer supports smart replacement of these results when object store space is at a premium. Information about the current region of interest is used to associate a dynamically changing priority with each object (as a function of its coordinates): objects farther away from these regions of interest are earlier candidates for replacement.

The system support for *intermediate result reuse* is similar to that of the simulation result reuse; SimX/SCIRun uses the same spatially indexed storage for solutions of the intermediate linear systems, and other data (in our example, preconditioners).

The modular nature of SimX/SCIRun, and the generic properties of different types of reuse allows these types of reuse to be automatically managed by SimX/SCIRun. Intermediate result reuse can be applied with minimal programming effort: the user needs to provide their own code that makes use of the intermediate result, using the SimX/SCIRun SISOL interfaces for saving and retrieving these results. Again, our implementation of this support envisages a more general use: a facility is provided for individual Worker processes to filter out portions of the data, good approximations to which are already known to be present within SISOL.

4 Evaluation

To quantify the effects of various types of reuse on a real-life multi-experiment computational study, we used the SimX/SCIRun system to conduct the defibrillator design study described in Section 2. For each of the experiments, we ran the system on a homogeneous IBM eServer cluster comprising 256 nodes, each with two 64-bit 2.2 GHz PowerPC 970 processors and 2 GB RAM, interconnected via a Myrinet network. The setup involved a single manager process, one SISOL directory server and four SISOL data server processes, and varying numbers of simulation worker processes, each hosted on a separate physical processor.

4.1 Scenarios

To measure the behavior of the system in the face of user interaction, we emulated user behavior using the notion of *runs*. A run consists of two studies, a 'before' study preceding a user action and an 'after' study following the action. When we conduct a run, we run the 'before' study to a prespecified level of completion, switch to the second study (simulating a user's key-click), and measure the amount of time, number of simulations issued, and number of solves performed for the second study to complete. We use the time from key-click to obtaining a fixed accuracy in the computed results (e.g. Pareto frontier approximation) as a measure of the responsiveness of the SimX/SCIRun system.

Each usage interaction scenario additionally includes two types of experiments. In the first type, we fix the 'before' and 'after' studies, but vary the level of completion that the 'before' study achieves before triggering the simulated user action. These experiments allow us to study the sensitivity of each type of reuse to the amount of accumulated simulation results. In the second type of experiments, we keep the same 'after' study, but vary the 'before study' so that the distance between the two studies is changing. The distance measure depends on the scenario: it could be the amount of change in performance metric parameter values, percentage overlap of the explored area, or the amount of change in value of one of the fixed design space dimensions. These experiments allows us to study the sensitivity of each type of reuse to variation in the study objectives.

To ensure realistic user behavior, we designed four usage *scenarios* - collections of related runs in which the user tries to a accomplish a task. Each scenario demonstrates the benefits of one or more types of reuse:

Scenario A: Setting the activation threshold voltage. In this scenario, the user fixes the position of the back electrode, the voltage, and defines the area where the front electrode can be placed. As his action, the user changes the activation voltage (in order to model, for example, a patient with heart tissues that are difficult-to-activate) (Figure 2). Simulation results can be reused in this case, since only the performance metric parameter is changed across the studies. One expects the total number of simulations run by the worker processes to decrease as the level of completion of the 'before' study increases, or the distance between the

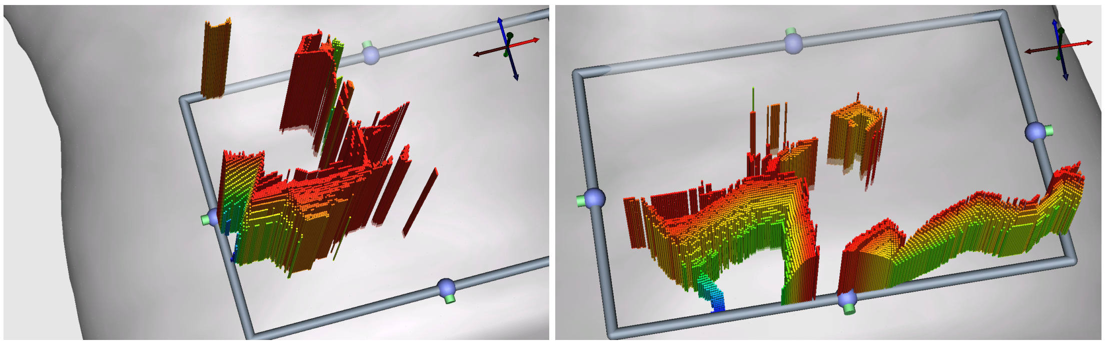

Figure 3. Pareto optimal points in the ’after’ studies of Scenario B (left) and C (right).

’before’ and ’after’ studies decreases.

Scenario B: Moving the explored region. In this scenario, the user fixes the position of the back electrode and the voltage. The user takes the action of sliding the area for the front electrode placement downward (See Figure 3). This action enables the reuse of performance metrics in the area of overlap between the two regions, since, in those areas, the same experiments with the same simulation parameters and the same performance metrics would have been performed for the ’before’ study. The number of simulations sent to the worker processes should decrease as the level of completion of the ’before’ study increases, or the distance between the ’before’ and ’after’ studies decreases.

Scenario C: Moving the back electrode. In this scenario, the user fixes the voltage and the placement area for the front electrode. As the user action, the user moves the back electrode toward the right shoulder. (Figure 3). Since the ’after’ study is expected to have a similarly-shaped Pareto frontier as the ’before’ study, using the ’before’ study’s frontier, even a partially-completed one, can reduce the total number of experiment required to do the ’after’ study. As a result, the number of simulations issued should decrease as the level of completion of the ’before’ study increases, or as the distance between the two studies decreases.

Scenario D: Increasing the shock voltage. In this scenario, the user wishes to get a coarse idea of how the Pareto-optimal placement of the front electrode is affected by the shock voltage. He fixes the back electrode’s coordinates, selects a voltage setting, and performs a parameter sweep of the front electrode placement across the entire area in front of the torso. The user changes the shock voltage and performs the sweep again. Since during the ’after’ study the preconditioners of the ’before’ study will already have been computed, the simulation workers in the ’after’ study can reuse these preconditioner results. This reuse is expected to decrease the run time of solves in the ’after’ study.

Change in Activation Threshold	Number of required simulations	Response Time (sec)
sweep	16385	1633
cold	1665	213.6
57%	1381	141.9
43%	1158	126.1
30%	423	44.69
20%	444	40.89
11%	260	27.32
3%	248	23.30

Table 2. Response time of Scenario A, as a function of the amount of change in performance metric parameter.

4.2 Effectiveness of Reuse

In this section, we investigate the sensitivity of reuse to the amount of performance history and the change in study parameters.

Table 2 shows the response time of the system in Scenario A as a function of the change in activation threshold voltage between ’before’ and ’after’ studies. As the amount of change decreases, the system takes advantage of the richer performance history to decrease the number of simulations needed. For baseline comparison, the ’cold’ column corresponds to running the ’after’ study without running the ’before’ study first, and the ’sweep’ column corresponds to running the ’after’ study without any reuse. With maximal reuse, we see up to 80x improvement in response time. These runs were conducted with 32 processors.

Table 3 shows the response time of the system in Scenario B as the percentage completion of the ’before’ study varies. As expected, due to performance metric reuse, the further along the ’before’ study, the fewer simulations need to be issued for the ’after’ study, and hence the lower the response time. For a large overlap of the exploration region (81.25%), performance metric reuse is able to

% Completion	Number of required simulations	Response Time (sec)
0	4467	238.5
18%	3521	176.7
37%	2725	139.7
56%	2112	113.6
74%	1292	84.20
93%	304	49.20
100%	13	16.80

Table 3. Response time in Scenario B, as a function of the percentage completion of the 'before' study

Displacement of back electrode (mm)	Number of required simulations	Response Time (sec)
sweep	16385	646.855
cold	7284	129.775
60	7666	154.108
50	7381	142.283
40	6762	130.37
30	6884	132.08
20	6868	127.14
10	6867	122.981

Table 4. Response time in Scenario C, as a function of the displacement of the back electrode.

bring down the response time to tens of seconds on a 32 processor cluster.

Table 4 shows how the response time of the system changes in Scenario C as the back electrode is moved further (these runs were conducted using 128 processors). The difference between the Pareto frontier in the two studies diminishes as the distance decreases, which leads to a decrease in the number of simulations needed in the 'after' study jumpstarted with the Pareto frontier of the 'before' study. The 'cold' and 'sweep' baselines are as described earlier and show an interesting effect: if the distance between the 'before' and 'after' studies is too large, attempting to reuse the Pareto frontier can be detrimental to the response time. This phenomenon is described in Section 4.3.

Table 5 shows the per-experiment run time in Scenario D as a function of amount of increase in defibrillation voltage. Using the preconditioners computed in 'before' studies, the simulation workers can improve their solver performance in the 'after' by as much as 35%, resulting in a 11.6% improvement in per-simulation run time.

Increase in shock voltage (V)	Avg. run time (secs)	Avg. solve time (secs)
cold	5.53	4.30
30	5.37	4.125
20	5.15	3.854
10	4.89	3.79

Table 5. Average per-simulation and per-solver run time in Scenario D, as a function of the change in electrode voltage.

4.3 Scaling Behavior with Reuse

In order to individually quantify the benefits from the different kinds of reuse discussed in Section 3.1, and to understand what impact (if any) the introduction of dependencies between study simulations has on overall scaling behavior, we measure the response time of the SimX system for different numbers of simulation workers.

Table 6 lists the various configurations that were used in the experiments, with one or more forms of reuse enabled.

Figure 4 shows the scalability log-log plot of the SimX system's response time under Scenario A, in a run where the activation threshold is increased by 3.33%. We studied all of the configurations except Configurations 2 and 4. The plot shows a two-orders-of-magnitude improvement between Configurations 7 (No reuse) and 6 (No Inter-study reuse), indicating that the Active Sampler by itself can cut down response time by almost a factor of 10. However, Configuration 3 (No Simulation Result reuse) shows that by adding Simulation Result reuse, we can further reduce the response time by approximately three-fold. The most interesting aspect of this plot is seen when comparing Configurations 5 (No Sampler Inter-study reuse) to 1 (Full Reuse). It shows that, in this case exploiting reuse actually hurts the response times for systems employing a small number of processors. This behavior results because a core assumption for exploiting this kind of reuse is that the 'before' study's Pareto frontier approximation is a good approximation of the 'after' study's Pareto frontier approximation. When, as in this case, the 'before' study's Pareto approximation has many design points that are not Pareto optimal in the 'after' study, the 'after' study may well need to run more experiments than if it had started with a coarse grid. However, even in this case, the coarse grid approach taken by Configuration 5 still has a drawback: the sampler won't be able to determine the area of interest until all the results from the coarse grid have

Configuration		Sampler	Sampler	Perf.	Simulation	Intermediate
No.	Name	Intra-study	Inter-study	Metric	Result	Result
1	Full	Yes	Yes	Yes	Yes	Yes
2	No Perf. Metric	Yes	Yes	No	Yes	Yes
3	No Simulation Result	Yes	Yes	Yes	No	Yes
4	No Intermediate Result	Yes	Yes	Yes	Yes	No
5	No Sampler Inter-study	Yes	No	Yes	Yes	Yes
6	No Inter-study	Yes	No	No	No	No
7	No Reuse	No	No	No	No	No

Table 6. SimX Configurations used in Scalability test

come in. With sampler inter-study reuse, all the points in the initial approximation are already on a fine grid, therefore there is no dependency between them. As a result, Configuration 1 scales better and overtakes Configuration 5 at 16 simulation workers. For 128 simulation workers, Configuration 1's response time is 31.4% shorter. Notice that, even assuming perfect scaling, it would have taken Configuration 7 (No Reuse) more than two thousand simulation workers to achieve the 20 second response time accomplished by Configuration 1 (Full Reuse).

Figures 5 shows the scalability log-log plot of the SimX system's response time under Scenario B, in a run with 81.25% of overlap in the explored region. We studied all of the configurations except Configurations 3, 4, and 5. One observes poor scaling for Configuration 1 (Full reuse). This is to be expected: with 81.25% overlap area in the 'before' and 'after' studies, most of the experiments' performance metric are already cached in the manager process, thus only a small percentage of simulation results used in the 'after' study require running new simulations (about 0.3%). The already low response time (under 20 sec.) cannot be decreased much further. If Performance Metric reuse is turned off (Configuration 2) more work needs to be done and scaling is closer to optimal, while the response time increases. Without Simulation Result reuse (Configuration 6), the response time increases by an order of magnitude, and removing sampler intra-study reuse results in an additional four-fold increase.

Figure 6 shows the scalability log-log plot of the SimX system's response time to Scenario C, in a run where the back electrode is moved by 10mm. We studied all of the configurations except Configurations 2, 3, and 5. The plot shows that even with full reuse (Configuration 1), the system can scale up to 128 simulation workers. It also shows that turning off intermediate result reuse (Configuration 4) would increase the system's response time by more than two times. Comparing Configurations 4 and 6 shows that most of the benefits from inter-study reuse come from intermediate result reuse. It also exhibits the same phenomenon as in Scenario A, where inter-study reuse hurts the response time for lower number of processors. Finally, without any reuse, the system would have taken more than seven times longer to respond.

Summarizing, the scaling studies: (1) highlight the importance of exploiting reuse only in situations where the usage scenario is likely to yield benefit; (2) demonstrate that more than one form of reuse needs to be exploited to yield overall response times that support interactivity; and (3) show that despite introducing dependencies in a parallel workload that did not have any

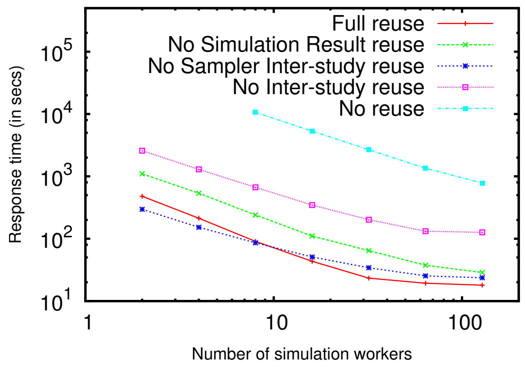

Figure 4. Scalability of Scenario A

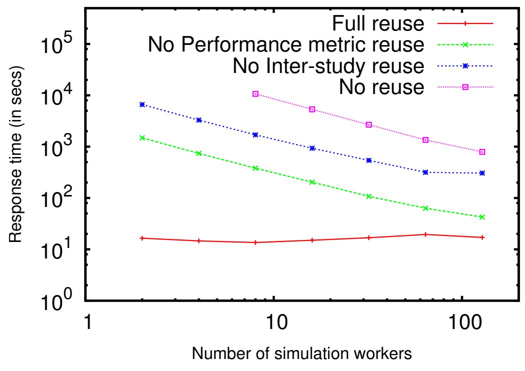

Figure 5. Scalability of Scenario B

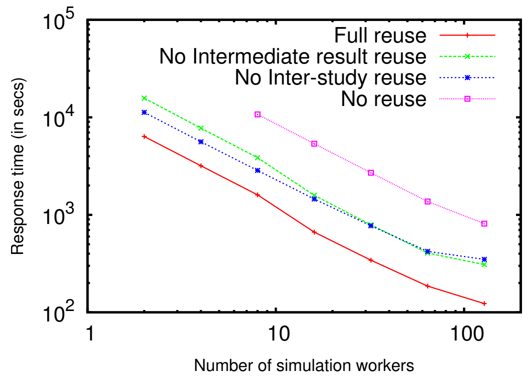

Figure 6. Scalability of Scenario C

to start with, the SimX system's exploitation of reuse achieves dramatic reductions in response time while still leaving behind enough parallelism to productively employ a moderate-sized parallel machine.

5 Related Work and Discussion

In its specific goal of supporting interactive computational studies involving a large number of computational experiments, the SimX system builds on and extends prior work on infrastructures for computational steering and grid-based parameter sweep applications.

Computational steering infrastructures permit users to continually inspect the state of running computations on parallel machines, and "steer" the computation as appropriate, e.g., by varying internal parameters of the computation. While several example systems exist, including Falcon [7], CUMULVS [10], UINTAH [6], DISCOVER [14], CSE [23], RealityGrid [4], gViz [26], and SCIRun [17, 12], they have primarily focused on the visualization and steering of individual, or a small number of experiments, as opposed to entire computational studies.

Grid-based parameter sweep infrastructures provide scheduling support for running the same application with a change in parameter values across distributed resources. Example systems such as Nimrod [2], Condor [22], Globus [16], and NetSolve [5], primarily focus on fault tolerance, resource management, and load balancing issues on grid computing infrastructures. These systems typically assume that the underlying simulations are independent and hence can be run in batch mode, ignoring interactivity considerations and without exploiting possibilities for reuse explored in this paper. Two exceptions are the work on Nimrod/O [1], which reuses previous results in a gradient descent-like guided search approach to identify the set of parameter points optimizing an objective function, and Virtual Instruments [8], which permit users to explicitly guide search of the parameter space by identifying regions of interest. Our SimX work combines elements of both systems but considerably extends them to enable reuse across user interactions, support more sophisticated objectives such as identification of Pareto-optimal points, and exploit reuse opportunities at the level of individual parameter point evaluations.

In its broader goal of illustrating the system techniques that enable interactive performance for a compute-intensive workload whose baseline turnaround time differs from what is desired by two to three orders of magnitude, our work highlights the importance of taking into consideration the role that individual computations play in the context of the *overall workflow*. Unlike traditional optimization strategies, which improve the performance of a single standalone computation, our work demonstrates that considering a collection of computations simultaneously yields many more optimization opportunities.

Taking advantage of these opportunities requires extended runtime system support. This support ranges from global resource managers that simultaneously determine what needs to be computed next and where, to maintaining a database of cached intermediate computation results that improve the performance of these computations in numerous ways. Reuse techniques supported in SimX/SCIRun build on ideas of a number of previous systems that have demonstrated the overall concept of reusing information from prior exploratory or useful program executions. Examples of such work include reusing access information from past iterations for periodic load balancing and data locality optimizations in irregular parallel computations [21], use of continuous system-level profiling to drive a host of profile-based compile-, link-, and run-time optimizations [3], and more recently, active measurement-based customization of numerical libraries to the underlying machine environment [24, 9].

6 Conclusion and Next Steps

This paper has described system support for and the benefits from aggressively exploiting reuse in the SimX computational study system. The goal of our work is to achieve interactive design space exploration studies involving thousands of individual computational simulations. By introducing controlled dependencies amongst the simulations, the reuse-driven optimizations achieve response time improvements ranging from 5x to 80x on different defibrillator design scenarios, yet leave enough opportunities for parallel computation to achieve reasonable speedups on a 128-node cluster. Such improvements become possible when considering optimization opportunities across the entire workflow of the study, as opposed to at the level of individual simulations.

While this paper has demonstrated a number of reuse optimizations for computational study systems, additional opportunities still remain untapped, which we intend on exploring in the near future. For example, it appears possible to combine a deeper analysis of the previous results with a higher-level understanding of the internal structure of the underlying simulations to build an analytical model of the study's performance space, and thereby enable tradeoffs between simulation result accuracy and run time.

References

[1] D. Abramson, A. Lewis, T. Peachey, and C. Fletcher. An automatic design optimization tool and its application to computational fluid dynamics. In *Proc. SC'01*, 2001.

[2] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. HPDC*, 1995.

[3] J.-A. M. Anderson. et al. Continuous profiling: Where have all the cycles gone? *ACM Trans. Comput. Syst.*, 15(4):357–390, 1997.

[4] J. M. Brooke. et al. Computational steering in RealityGrid. In S. Cox, editor, *Proc. UK e-Science All Hands Meeting*, 2003.

[5] H. Casanova and J. Dongarra. Netsolve: A network server for computational science problems. *Intl. J. Supercomp. Appl. and High Perf. Comp.*, 11(3):212–223, 1997.

[6] J. Davison de St. Germain, J. McCorquodale, S. Parker, and C. Johnson. Uintah: a massively parallel problem solving environment. In *Proc. HDPC*, 2000

[7] W. Gu. et al. Falcon: on-line monitoring and steering of large-scale parallel programs. In *Proc. Symposium on the Frontiers of Massively Parallel Computation, 1995*, 1994

[8] M. Faerman, A. Birnbaum, H. Casanova, and F. Berman. Resource allocation for steerable parallel parameter searches. In *Proc. Grid'02*, 2002.

[9] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. IEEE*, 2005.

[10] I. Geist, G.A., J. Kohl, and P. Papadopoulos. Cumulvs: providing fault tolerance, visualization, and steering of parallel applications. *Intl. J. Supercomp. Appl. and High Perf. Comp.*, 11(3):224–35, 1997.

[11] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183, 2004.

[12] C. Johnson, R. MacLeod, S. Parker, and D. Weinstein. Biomedical computing and visualization software environments. *Comm. ACM*, 47(11):64–71, 2004.

[13] R. MacLeod, C. Johnson, and P. Ershler. Construction of an inhomogeneous model of the human torso for use in computational electrocardiography. *IEEE EMBS Intl. Conf.*, pages 688–689, 1991.

[14] V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications. *Concurrency: Practice and Experience*, 13(8–9):737–754, 2001.

[15] A. Messac. Physical programming: Effective optimization for computational design. *AIAA J.*, 31(4):149–158, 1996.

[16] J. Nabrzyski, J. Schopf, and J. Weglarz, editors. *Grid Resource Management: State of the Art and Future Trends*. Kluwer, 2003.

[17] S. Parker and C. Johnson. SCIRun: a scientific programming environment for computational steering. In *Proc. SC'95*, 1995.

[18] S. Parker, M. Miller, C. Hansen, and C. Johnson. An integrated problem solving environment: the SCIRun computational steering system. In *Proc. HICSS*, 1998.

[19] J. Schmidt and C. Johnson. DefibSim: An interactive defibrillation device design tool. In *Proc. EMBS Conf.*, 1995.

[20] M. Scott and E. Antonsson. Preliminary vehicle structure design: An industrial application of imprecision in engineering design.

[21] J. P. Singh. et al. Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-hut, fast multipole, and rasiosity. *J. Parallel Distrib. Comput.*, 27(2):118–141, 1995.

[22] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 2004.

[23] R. van Liere, J. Mulder, and J. van Wijk. Computational steering. In *Proc. HPCN*, 1996.

[24] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[25] B. Wilson, D. Cappelleri, T. W. Simpson, and M. Frecker. Efficient pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, 2(1):31–50, 2001.

[26] J. Wood and K. Brodlie. gViz - visualization and steering for the grid. In S. J. Cox, editor, *Proc. UK e-Science All Hands Meeting*, 2004.

[27] S. Yau, E. Grinspun, V. Karamcheti, and D. Zorin. Sim-X: Parallel system software for interactive multi-experiment computational studies. In *Proc. IPDPS* 2006

[28] S. M. Yau, E. Grinspun, V. Karamcheti, and D. Zorin. Simx meets scirun: A component-based implementation of a computational study system. In *IPDPS*, pages 1–6, 2007.