Multiscale Software Visualization for Computational Science and Engineering

1. General background. Computational Science and Engineering (CS&E) is a fast growing and increasingly important field of research as shown by its impact in fields including medicine, biology and bioengineering, energy, chemical engineering, climate modeling, and physics. The scale of today's CS&E problems was almost unimaginable ten years ago. It is not uncommon for problems to process billions of data points over thousands of time steps. To address such massive problems, new highly complex computer architectures and software systems have been developed. These systems come in diverse forms—heterogeneous processing environments (e.g. CPU/GPU hybrid environments), many-core systems (hundreds of thousands of cores), a variety of memory configurations (i.e. shared vs. distributed memory, with various hierarchical memory schemes), and complex network configurations connecting the nodes of a cluster. Developing, debugging, and optimizing software for these systems is increasingly difficult as different hardware configurations lead to complex interactions, for even the simplest piece of software. Unforeseen processing bottlenecks and application errors can develop during execution. *Given current techniques, it can be extremely difficult to locate the source of errors due to the volume of runtime data and complex interactions applications produce.* We propose overcoming these challenges by introducing **new techniques for visualizing the evolution of running applications, enabling new forms of analysis** on complex hardware and software systems with the potential to significantly increase productivity in all fields of CS&E, **enabling more science for the time and money spent**.

The impact of traditional debugging tools on software developer productivity is well-known. Developers find that products such as GNU Debugger (GDB), Microsoft Visual Studio, NVIDIA's graphics hardware debuggers [7] and TotalView Debugger [13] have shortened software development cycles. Such tools work by providing a transparent view of an application with controls for inspecting the details of its execution. The detailed, hardware-oriented level at which these tools operate assists in searching out the specific sources of errors.

Performance analysis (or *profiling*) is frequently used to understand the complex interactions of software and hardware. Profilers such as GNU GProf, Apple Shark, and Intel VTune work by revealing which portions of an application's source code consume the most execution time. Profilers can also collect information from hardware sources, giving a big-picture view of program execution. For example, VAMPIR [6] profiles by collecting statistical performance data from parallel programs and displaying it in summary views. Such profilers can correlate portions of source code with their actual execution time, but generally cannot discern specific causes for reduced application performance. While they do not produce incorrect results, such performance degradations are still considered errors—yet the techniques for diagnosing their causes are not as mature as the ones for finding conventional logic or design errors.

Software visualization is a new approach [12] for better understanding specific aspects of program behavior. Examples include visualizing certain logic errors [4, 5, 10], the evolution of running software [11], and performance-critical subsystems [1]. Most software visualizations do not work on a large enough scale for modern supercomputers, nor do they attempt to connect specific causes for poor performance to the programming structures that precipitate them.

2. Technical significance of the research. CS&E continues to call upon experts in High Performance Computing (HPC) (also called petascale or exascale computing) for solving large computational problems. The capabilities of software and hardware HPC systems have grown exponentially while the debugging and profiling tools have not keep pace with the needs of these systems that produce such a large volume of runtime data.

The overarching goal of this project is to develop new algorithms that further advance debugging and profiling techniques by visualizing hardware and software events in a multiscale, global-to-local manner, enabling developers to observe general behavioral patterns in an application's execution, then perform finer-grain investigations to identify specific causes for that behavior. This includes summary visualizations of the entire supercomputer, all the way down to local visualizations describing the behavior of a single core of a single node of the supercomputing cluster (see Figure 1). For instance, visualizing the evolution of memory interactions on a supercomputer may help determine if memory layouts and access patterns are positively or negatively affecting performance. These new methods promote the search for causes of application errors to a visual analysis using new visualization techniques.

As supercomputers grow more complex, the interconnection between components begins to exhibit chaotic behaviors. Our observations of these systems has inspired our new approach to software debugging and profiling, giving developers an intuitive understanding of system behavior by applying new visualization techniques (see Section 3). This idea parallels medical diagnosis. Developers are provided a high-level summary visualization of their application as it runs, much as a physician would use an ECG or full body X-ray/CT. The developer monitors application progress, identifying problems with overall execution and performing further examination at lower levels of execution (e.g. an individual node of a cluster)—much like the physician orders further detailed testing after discovering abnormalities from preliminary tests. As monitoring continues, the developer has the option to explore yet lower levels of system interaction, or return to higher-level summary views, until a full diagnosis and treatment plan can be determined for any errors within the system. The governing idea is that the visualization should look as natural and intuitive as possible, providing a faithful sense of what is occurring during program execution.

Both Dr. Johnson and Dr. Rosen are experts in visualization and scientific computing. The software visualization subfield represents a new direction of research in which both Dr. Johnson and Dr. Rosen are well-equipped to leverage their expertise and existing collaborations, resulting in a significant impact in many other fields of scientific and engineering research.

3. Description of methods. To provide summary visualizations of performance-critical subsystems, we will employ techniques from *organic visualization* [2]—the idea that visual elements can self-organize into representations of the evolution of a whole system, with special cues for noteworthy events, such as changes in motion, size, color, or proximity of elements (see Figure 2). We believe that such techniques will be expressive and informative when the visual elements are adapted to abstractions of software and computer system components. We will design a suite of techniques to enable visual analysis for many scales of computer hardware, from computing clusters communicating over a network, down to the internals of a single-processor core.

Collecting runtime data from applications is another important component of this work. The most common approach is to insert additional instructions (known as instrumentation) into software to collect this data. The instrumentation can be performed manually (by hand during development), statically (automatically during compile time) [9], or dynamically (automatically during run time) [3].

Together, these two frameworks—one for collecting runtime data from applications, and the other for designing, testing, and deploying visualization strategies—can be used to reconstruct an application's behavior. The open and challenging problem remains how to connect these two separate frameworks to allow for visual investigation of how various aspects of applications interact, arranging the data to reflect performance characteristics and highlighting noteworthy performance-related events. By combining these two frameworks, we begin to achieve the goal of clearly conveying high-level logic and performance errors and their underlying causes to develop-

ers, enabling them to eliminate these errors and streamline their software for our ever-accelerating supercomputing age.

4. Impact of this research. This project has the potential to redefine the way in which software developers debug and profile their applications, making software development more efficient. Debugging enables faster software development cycles, more stable software, and software that is more likely to produce correct results. Profiling helps produce optimized software, enabling it to process larger datasets, deliver results faster, and address more computationally complex problems.

Impact in HPC. Providing a platform for optimizing parallel software is an important problem in HPC. Supercomputers are expensive to run, requiring a huge initial investment, long-term support staff, and a high electrical needs for the hardware itself, along with large scale cooling needs. Inefficient software makes poor use of computing resources, wasting money, time, and energy. The ability to improve utilization would save millions of dollars, conserve valuable research hours, and reduce the environmental impact of supercomputer energy usage.

Impact in CS&E. Simulation is an increasingly important branch of science and engineering. This work aims to shorten both the development time and run time of such simulation software—in essence, enabling more science for the time and money spent. As a further benefit to users of time-shared supercomputers, results are delivered faster, decreasing waiting time for access to such resources, and thus increasing resource efficiency. Finally, better optimized software allows for processing larger, more complex data sets, enabling scientists and engineers to both calculate more accurate solutions to problems and attack harder problems once thought too complex for available computing capabilities.

5. Specific goals, objectives and anticipated results. We divide our project goals into three categories—short, medium, and long-term goals.

Our short-term goal within the next year is to collect initial results for pursuing extramural funding opportunities. This includes developing algorithms that enable meaningful summary visualization of memory and cache performance for desktop level single-core software. These summary visualization algorithms would require the developer to manually instrument the programs, but would automatically produce the visualization from application runtime data. As a further proof of concept, we will perform case studies on various small, but commonly used, algorithms to verify the efficacy of our proposed techniques.

Our medium-term goal (1–3 years) is to develop algorithms to extend our summary visualization techniques to multiple hierarchical scales for system-wide examination down to minute subsystem examination. For example, our approach would show a summary visualization of the entire memory hierarchy and allow a developer to examine lower-level components such as individual memory cache lines. We plan to make our entire system automatic by developing algorithms for collecting program data. We further hope to collaborate with other members of the SCI Institute, the University, and the boarder CS&E community in performing case studies on larger software projects.

The long-term goal of our project (3–5 years) will be to extend support for our techniques to supercomputers with hundreds of thousands of nodes, heterogeneous processing environments, and diverse memory architectures, such as MPI and NUMA environments. One successful tradition of the SCI Institute is the production and publishing of software by its researchers and staff developers. We plan to continue this practice by transforming our techniques into powerful tools that will be distributed as publicly available software packages, making our techniques accessible to the CS&E and HPC communities at large.

6. Extramural Support. High Performance Computing and Computational Science and Engineering continue to be well-funded areas of research. We plan to use this seed grant funding for pursuing preliminary results which will lead to future extramural funding from any number of fields. The scope of this project primarily qualifies it for HPC solicitations, but also for many computer systems, computer graphics, and visualization solicitations as well. Our exact plan of action depends upon the timing and quality of our preliminary results, but some solicitations we will pursue are as follows:

- NSF "Computer and Network Systems (CNS): Core Programs," NSF 10-573, due annually between September and December based upon proposal size. Under Computer Systems Research, the solicitation specifically states: "Understanding highly parallel computing systems also requires innovative methodologies and tools for quantitative and qualitative characterization, evaluation, monitoring and prediction of system behavior at different levels, including the implications of workloads in multi-core system design." This statement seems to target just the sort of innovative algorithms we plan on developing.

- NSF "Computing and Communication Foundations," NSF 10-572, due annually between September and December based upon proposal size. The Software and Hardware Foundations cluster of the solicitation requests: "SHF cluster encourages proposals that transcend traditional areas, import ideas from other fields, or capture the dynamic interactions between the architecture, language, compiler, systems software, and applications layers." Software visualization, as we plan to use it, fits this description of cross-cutting technology leveraged against new problems.

- U.S. Army Engineer Research & Development Center "Broad Agency Announcement," 2010. The High Performance Computing and Networking (ITL-3) cluster solicits proposals involving many issues we plan to address, including heterogeneous computing clusters and managing resource utilization.

- As High Performance Computing continues to be a popular field of research, other funding agencies (e.g., Department of Defense and Department of Energy) also request high-performance computing proposals in some of their solicitations.

Dr. Johnson has an extensive history of success in receiving external support for his research. Dr. Rosen has yet to receive any external support.

7. References

[1] Edward E. Aftandilian, Sean Kelley, Connor Gramazio, Nathan Ricci, Sara L. Su, and Samuel Z. Guyer. Heapviz: interactive heap visualization for program understanding and debugging. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 53–62, New York, NY, USA, 2010. ACM.

[2] Benjamin Jotham Fry. Organic information design. Master's thesis, Massachusetts Institute of Technology, May 2000.

[3] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. pages 190–200, Chicago, IL, June 2005. Programming Language Design and Implementation (PLDI).

[4] Emerson Murphy-Hill and Andrew P. Black. An interactive ambient visualization for code smells. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 5–14, New York, NY, USA, 2010. ACM.

[5] Colin Myers and David Duke. A map of the heap: revealing design abstractions in runtime structures. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 63–72, New York, NY, USA, 2010. ACM.

[6] W.E. Nagel, A. Arnold, M. Weber, H.-Ch. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–89, January 1996.

[7] NVIDIA Corp. NVIDIA introduces industry's first debugger and profiler for GPU computing. `http://www.nvidia.com/object/io_1239219734947.html`, April 2009.

[8] Michael Ogawa and Kwan-Liu Ma. code_swarm: a design study in organic software visualization. *IEEE Transactions of Visualization and Computer Graphics*, 15(6):1097–1104, October 2009.

[9] P. Pereira, L. Heutte, and Y. Lecourtier. Source-to-source instrumentation for the optimization of an automatic reading system. *J. Supercomput.*, 18:89–104, January 2001.

[10] George G. Robertson, Trishul Chilimbi, and Bongshin Lee. Allocray: memory allocation visualization for unmanaged languages. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 43–52, New York, NY, USA, 2010. ACM.

[11] C. Stolte, R. Bosch, P. Hanrahan, and M. Rosenblum. Visualizing application behavior on superscalar processors. In *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, pages 10 –17, 141, 1999.

[12] Alexandru Telea. Introduction to the special issue of selected papers from softvis'2008. *Information Visualization*, 8(2):85–86, Summer 2008.

[13] TotalView Technologies. Case studies. `http://www.totalviewtech.com/support/case_studies.html?via=resources`, October 2010.

8. Appendix A

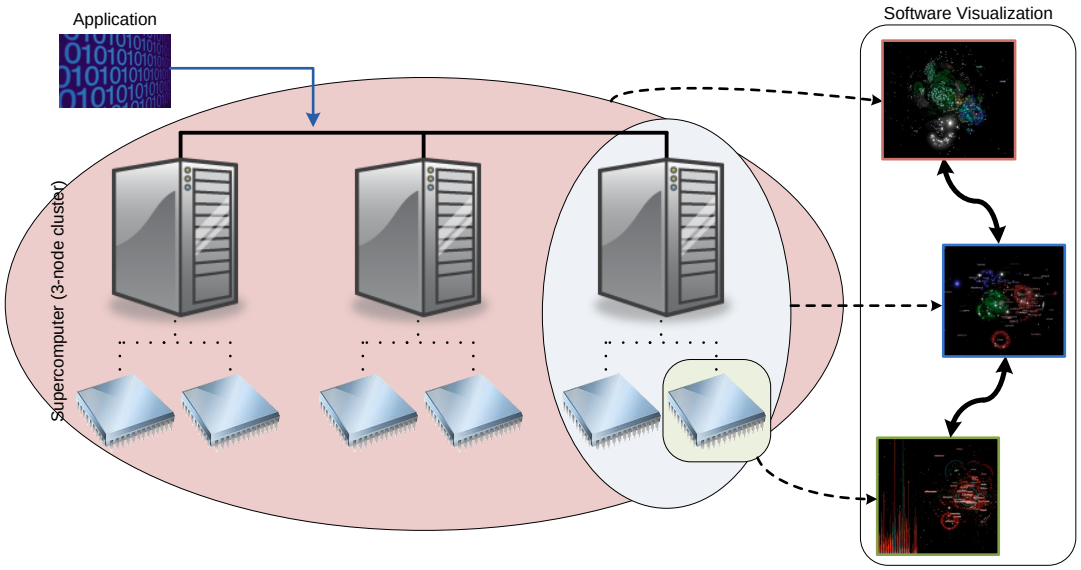

Figure 1: Overview of our proposed system. A scientific or engineering application would be executed on a supercomputer. We will then provide algorithms which will visualize the runtime data produced at multiple scales: from a system-wide summary visualization, to a mid-scale single node visualization, down to a single processor visualization. Example visualizations from Ogawa and Ma [8].

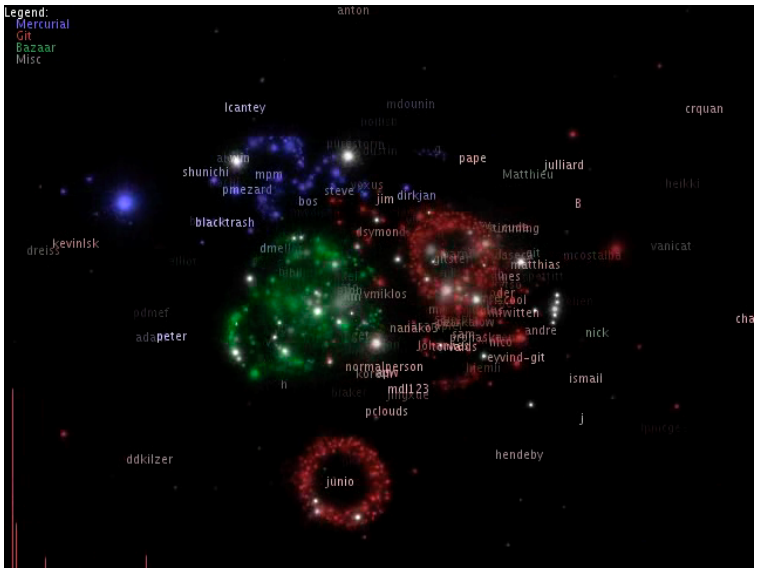

Figure 2: Example of an organic visualization showing a comparison of usage of different version control systems. Each system forms its own colored "galaxy," while developers (white dots) may move between them as they work on different projects. The relative size of the "galaxies" encodes the volume of activity within each system, while motion encodes particular activities as they occur [8].