Research Statement

Philip A. Wilsey

1 Prologue

The principle objectives of my studies are the advancement of knowledge and understanding of **high performance computing**, **parallel and distributed systems**, and **embedded systems** by the: (i) development of pragmatic methodologies; (ii) development of prototype tool sets; and (iii) introduction of these topics into the graduate and undergraduate curricula. My intent is to better understand the world of computing and to communicate that understanding (and the process to by which one can add to that understanding) to my students and colleagues. In many cases, my understanding begins with a crude idea to improve a system; the idea is applied and analysis begins. Ideally, the experimental investigations provide new insight to the fundamental behaviors of the system so that future investigations can work from stronger fundamentals. Key to my work is careful adherence to the scientific method. Over the years, I have been surprised to learn that I need to revisit this with new students. It is something we are all taught, but something that is not always applied.

As an experimentalist, I believe strongly in sharing my experimental system software so that others can replicate my research results. Consequently, I have some experience developing and distributing software systems that others can use. I encourage my students to follow a test-first design strategy and inline code documentation (often using doxygen or an equivalent). Most recently I have started hosting our software projects as public repositories on github so that others can easily access, contribute to, and replicate our work. I encourage and welcome contributions by others.

While I write this document in first person singular, almost everything I do involves at least one or more student or faculty co-investigator. My door is open: stop by and spend some time.

2 Overview

I am an experimentalist working in high performance computing, parallel and distributed systems, high dimensional data clustering, embedded system, and medical devices. I am studying the optimization of compute services to better exploit multi-core and many-core processors. I have also been studying the opportunities that virtualization offers for optimizing guest O/S services in Beowulf clusters to better support parallel applications. Within these studies, I have started examining the prospects of using big.LITTLE processors to construct low-cost clusters for executing fine-grained message passing applications. I have also been studying the application of feedback control to optimize distributed system operation. The focus of my investigations has been Parallel and Distributed Simulation (PADS) using optimistic synchronization protocols and, in particular, the Time Warp Mechanism.

In addition to my work with parallel and distributed computing and systems, I have also been working with several colleagues and the local medical community to develop point-of-care medical devices to assist patient diagnosis, treatment, and monitoring. Particular emphasis in these areas have been the development of point-of-care devices to: (i) use photo-metric analysis to help diagnose stroke, (ii) to assist in assessing neurological health and mild TBI, (iii) point of care devices to assist in the monitoring of asthma medication compliance, and (iv) field deployable devices for multi-metal assessment of blood. I have also been working with a group of engineering and medical researchers to develop a "medical problems" clinic to identify and pursue opportunities for medical device development.

3 Current Research Projects

3.1 Parallel and Distributed Simulation

My primary research studies have been with optimistically synchronized, parallel and distributed discrete-event simulation (PDES) using the Time Warp mechanism. My initial studies focused primarily on dynamic methods to tune the sub-algorithms and run-time parameters of Time Warp. These studies continue. More recently I have also started to focus on the challenges and opportunities of multi-core and many-core processors as well as virtualizable Beowulf clusters for parallel simulation. I discuss some of my ongoing parallel simulation work more fully below.

Run-time control and optimization of Time Warp: This work addresses the construction of a generic Time Warp simulation kernel that can support a broad range of simulation models with a wide range of execution time characteristics. Initially a general purpose C++ API was designed for building simulation models and then an implementation of the API was developed and optimized. The simulation kernel contains both sequential and parallel implementations for comparative analysis. This kernel is called WARPED2 and it is hosted at https://github.com/wilseypa/warped2. The software contains several of the critical dynamic adjustment algorithms for Time Warp and it has recently undergone an extensive modification to support threaded and parallel execution on multi-core and many-core Beowulf clusters. Simulation models for WARPED2 are located in a separate github archive.

The Pending Event Set: The pending event set in WARPED2 is deployed so that the lowest time-stamped events from every LP are placed into one or more event scheduling queues (called Least Time-Stamp First or LTSF queues) for execution. The initial solution had only one LTSF queue, but lock contention for the LTSF queue became a detriment to performance once the number of worker threads exceeded 5-7 (on a 48-core system). The current design instantiates multiple LTSF queues and assigns unique subsets of the LPs to each LTSF queue. The system then binds a collection of "worker" threads to a specific LTSF queue that process events only from that queue. Not only does this organization alleviate contention for the LTSF queue, it also provides a convenient mechanism for implementing a lightweight load balancing mechanism between the LTSF queues within the processing node. This solution is working well, but we continue to examine it for further optimization.

We are also examining the data structure used to implement the LTSF queue. Initially WARPED2 contained configurations to use either an STL *multi-set* or *splay tree* for the LTSF queue. The system has recently been extended to also support the use of a *ladder queue* (ladder queues are a variant of calendar queues that have self-balancing capabilities built into the sizing of the calendar months). Experiments show that the ladder queue data structure performs better than either of the other two data structures. While the ladder queue has the obvious advantage of reducing the number of elements (events) sorted (at any one time) for execution, we believe it also has a number of additional properties that make it attractive for further optimization. For example while the initial implementation has a single lock around the entire ladder queue, migrating the locks down into the rungs of the ladder should help to further reduce contention. This is currently being investigated.

Another key advantage of the ladder queue is that the structure is relatively independent on the time granularity of the simulation model. Ideally, the range of event timestamps for the buckets that define each epoch of the ladder should help ensure that the events within any bucket of the ladder queue are causally independent. This idea is related to the lookahead property of conservatively synchronized parallel simulation. The lookahead concept results from a static analysis of the simulation model that guarantees a time window ahead of any source timestamp in which no additional events will be generated (effectively a minimum time delay guarantee on the event processing latency). While not as strong as lookahead, our key hypothesis for this structure is that *the bucket size for the ladder queue will typically encompass a range of event timestamps that are causally independent*. Thus, we plan to explore a modified version of the ladder queue that removes all sorting and relies on a weak causal relation of the events within a single bucket to significantly improve the performance of this critical data structure. When the sorting requirement is removed, it should be possible to use atomic moves to manage the buckets, resulting in a fully wait-free, contention minimal pending event list data structure. Note that if two events happen to be causally dependent in the same bucket, simulation correctness is preserved by the rollback that occurs when the later event is encountered for an earlier local time.

Transactional Memory: Lastly, we are beginning new studies on the opportunities of replacing the locked critical regions in the WARPED2 kernel with transactional memory access that is now available in the new Intel Haswell processors.

3.2 Virtual Machines for Fine-Grained Parallelism

Over the years numerous studies have pursued the development of application specific operating systems/run-time environments for performance critical applications. This is especially true for parallel computing. Unfortunately such solutions require that the (expensive) parallel hardware be entirely dedicated for that run-time environment. Virtualization presents an interesting opportunity for developing a full custom solution without the requirement of a fully dedicated hardware platform. With virtualization, a shared hardware platform can be safely configured with a standard setup *and* with multiple, distinct application specific environments. However, in order to be effectively used for performance critical applications, the overheads of the virtualization services used by the performance critical applications must be minimal or, ideally, non-existent.

Network Latency in Virtual Environments: For our work with parallel simulation on clusters, the network latency in the virtual environment must be as good or better than native. Network virtualization does, however, come at a cost. Our initial studies with para-virtualized network drivers shows that latency performance between virtual machines is no better than about 70% of native. Furthermore, even with Direct Connect our studies show that performance rarely exceeds 95% of native. Therefore, in order to deploy a virtual machine for fine-grained parallel computing, additional performance gains must be found.

Because we are working in a virtual machine framework, the replacement of the TCP/IP virtual machine device drivers with InfiniBand based Ethernet (IBoE) drivers may allow us to increase performance to, and possibly above, native. Using these drivers, we were able to measure message latency performance that was slightly better than native (TCP/IP based) performance. We are encouraged, but not entirely satisfied.

Our current attack on virtual network performance is to couple the IBoE driver with PCI-passthrough (or Direct Connect) and polling to achieve very low message latency. While this requires that the Virtual Machine Manager (VMM) assign a dedicated network card to the virtual machine, this assignment is temporary and the NIC is returned to the VMM upon completion of the application's execution. Furthermore, adding an extra network card to a compute node is a very low cost expenditure (on the order of $10), so we believe it is a worthwhile cost in many cluster deployments. Using this model, we have been able to reduce message latency between applications in a virtual machine so that they are 60% lower than native.

Low-LAtency Minimal Appliance Operating System (LlamaOS): Leveraging our studies with low latency messaging in virtual machines, we have developed a general purpose Low-LAtency Minimal Appliance Operating System (LlamaOS). LlamaOS uses the standard Linux tool chain and supports development in C, C++, Fortran, and MPI. Many applications can be built directly into LlamaOS without modification (or with minimal modification). The system is still under active development and is changing rapidly.

3.3 Clusters with big.LITTLE Processors

Beowulf Clusters are generally composed of nodes costing thousands of dollars each. These expensive systems delivery high performance and they are almost always configured with standard operating systems and networking infrastructure for shared use by a community of users. Thus, any application specific tuning or optimizations must generally be done only within the user space so that the system can be effectively shared and used by all users. While this configuration operates effectively for coarse-grained and scientific computing needs, the service delivered for finer-grained applications or applications better served by non-conventional O/S services is generally lacking. Fortunately the commercial market is now delivering inexpensive, small form factor hardware that enables the construction of low-cost Beowulf Clusters that can be tuned for specific applications. Harnessing the power in such systems can, however, be challenging.

One interesting prospect for building low-cost Beowulf clusters is the emerging ARM solutions with big.LITTLE processors. Currently development boards with these components can be purchased for as little as $100–$400 each. At these costs, the construction of a (for example) 64 node cluster would be only a few thousand dollars. Despite their low cost, however, these systems come with some significant computing power. The current odroid-xu, for example, comes in quad-quad configuration of big.LITTLE processors for just under $200 each. While the current configuration of these processors is such that only the big or LITTLE quad can be operating at a time, Samsung has announced plans to release a software patch to enable full quad-quad processing with these devices.

Network Latency: One of the more significant challenges to fine-grained parallel computing on Clusters is network latency. Often these latencies (using Ethernet) can be in the 10s to 100s of microseconds even for small messages. We have already shown that reducing this latency can have dramatic impact on the run time performance of Time Warp synchronized parallel simulations. In one study with an x86 cluster, we were able to reduce latency by 60% which resulted in a 40% decrease in run time of our Time Warp applications. Key to the latency reduction was a replacement of the TCP/IP protocol network driver with an Infiniband-over-Ethernet (IBoE) driver. We also configured the IBoE driver to poll the network card instead of being triggered by hardware interrupts. While not generally usable in a shared cluster environment, its uses in a low-cost cluster environment could be possible.

In this study, we are developing a IBoE network driver for the odrid-xu platform. Our experiments will explore configuration of the driver in both polling and interrupt driven modes of execution. Furthermore, we plan to explore binding the polling driver to a dedicated LITTLE core.

Task allocation across the big.LITTLE processors: There have been several studies to develop full custom operating systems for executing Time Warp based parallel simulation. While we will avoid the design and development of a full custom O/S, we do anticipate customization of some key services of the Linux and Time Warp kernels for our needs. In particular, we are currently studying customization of five main services, namely: (i) the aforementioned network services, (ii) the construction of a Time Warp specific memory management services and fossil collection, (iii) global time management and termination detection, (iv) pending event set management and scheduling, and (v) process management and the migration (load balancing) of LPs to remote nodes for processing. A critical element in these studies is the allocation of the various processing components to the big.LITTLE cores. Initially we plan to bind these services to the LITTLE cores for execution, leaving the big cores assigned to event processing. However, explorations of other deployments will also be investigated.

3.4 High-dimensional Data Clustering by Random Projection Hashing

RPHash is a streaming algorithm for data clustering based on frequent itemset counting of locality sensitive hash (LSH) collisions generated by multi-probe, randomly projected data vectors. The proposed algorithm called Random Projection Hashing or RPHash, trades sub-optimal computationally performance for improved memory efficiency over other clustering algorithms. Memory efficiency, a premium in streaming and distributed algorithms allows RPHash to be run on a variety of streaming data environments. Two Additional features, temporal weighting and privacy are also provided intrinsically by RPHash. By performing multiple projections utilizing recent advances in the fast Johnson Lindenstrauss transform (FJLT) we are able to attain a temporally weighted, streaming clustering algorithm with log-linear complexity growth and intrinsic data security. The system being designed will be highly scalable and compatible with commercial cloud-based distributed computing resources (*e.g.*, Amazon EC2). In addition, the system will focus on multi-dimensional medical data to ease of use for researchers and medical professionals with an applicability to a wide variety of biological data analysis problems.

3.5 Medical Devices

Recently I have been working with the local medical community to explore the design and development of medical devices to improve the quality of patient care. My initial work in this are has placed a particular emphasis on the development of devices to perform photo-metric analysis to assist the diagnosis of stroke and jaundice patients. We are also working to develop monitoring and analysis systems to help diagnose patients with mild TBI (traumatic brain injury) and other neurological disorders. I have also joined a group working to instrument an asthma inhaler to assist in the treatment of juveniles with asthma. The device will be designed to monitor inhaler use, environmental factors, and capture patient feedback to assist medical personnel to customize and optimize the patient's treatment. Most recently I have joined a multi-disciplinary group of engineers and medical personnel to explore the broader problems and opportunities that the medical community faces. This group holds regular brainstorming sessions split between problem identification and potential solution identification.

4 Past Research Activities

Massively Parallel SIMD/MIMD Multiprocessing: The project investigated the validity of the assertion that multiprocessors with a small, fixed number of control units can efficiently support both the SIMD and MIMD processing paradigms. The resulting multiprocessor must efficiently support both SIMD and MIMD processing, while minimally hindering the performance in either mode. The approach will be to restrict the complexity of the processing elements (PE) and use a fixed number of control units that does not increase with the number of PEs. This problem required a careful consideration of the MIMD instruction set to be interpreted. In particular, the execute phases of the interpreted instructions must be made as reusable as possible to maintain high PE utilization. While software optimizations such as these produced speedups, we also identified a few simple, inexpensive, hardware modifications that would dramatically boost MIMD performance at virtually no cost to SIMD performance.

Object-Oriented Extensions to VHDL/VHDL-AMS for Systems Level Modeling: I had the good fortune that Dr. Peter Ashenden decided to spend a his sabbatical year studying with me. This association began a series of investigations to seamlessly extend VHDL and VHDL-AMS for systems level modeling. These investigations borrowed heavily from Ada 9X (generics, inheritance) and CSP (communication channels). The chief challenge to these investigations were to develop and refine extensions that presented strong additions to the modeling capabilities of VHDL (specifically into the system level domains) and that were syntactically and semantically consistent with the existing VHDL standard (at the time VHDL-93). I expanded those studies to VHDL-AMS with Dr. Greg Peterson.

A Formal Model of VLSI Systems Compatible with VHDL: This work uses an interval temporal logic to formally define a machine model that is congruent with the hardware description language VHDL. The emphasis of this project has been toward the development of a machine models that characterizes the dynamic behaviors of VHDL. The model is unique in that it is not derived from the uniprocessor simulation kernel generally used to define VHDL behaviors. The semantics of this model was embedded into the PVS proof maintenance system and we were able to reason about equivalences between two VHDL models. Thus we were able develop compiler optimizations for parallel simulation (see SAVANT below) and show their correctness with the proof maintenance system.

In addition to the dynamic model, the project also developed a formal statement of the static semantics of VHDL that shows when a VHDL model is well typed. Furthermore, a reduction algebra was developed to formally define those structures in VHDL that are defined to be equivalent in the language standard. The results of these investigations are summarized in a book published in 1999 by Kluwer Academic.

SAVANT Tools: An Extensible Intermediate for VHDL accompanying analysis and simulation tools: This work studied the development of an extensive intermediate form for VHDL (called AIRE) and its creation/support through a C++ compiler framework called SAVANT. The SAVANT tool chain contained a front-end compiler/analyzer that translated VHDL into AIRE. The SAVANT implementation of AIRE used class derivation and a cloning mechanism to incorporate extensibility into the back-end. The back-end of the system had two derived extensions, one was a VHDL code generator, the other was a C++ code generator that produced code that could be executed on the WARPED2 parallel simulation kernel.

Epilogue

In general, I like to work with other investigators, be they students, faculty, or other researchers. I find that these associations expose my ideas to more critical analysis and that I end up conducting much better work as a result. To all my past collaborators, thank you; my life has been much richer because of you.