

Research Statement (V. Freeh)

My research embraces and develops new computer architectures, bringing new features to systems. Over the years my students and I have incorporated these features into useful and usable system abstractions that provide a more capable foundation on which to build application programs.

Throughout my career, I have been working with cutting-edge architectures. *Filaments* is one of the very first projects to support parallel programming on commodity workstation clusters (the fastest growing supercomputer architecture). The *Mona/Magi* project targeted intelligent devices, such as active memory and smart peripherals. Another project, the *Web Booster*, is one of the first of a new breed of web server architectures. Finally, the *Morph* architecture represents a radically different approach to building a power-aware microprocessor. New architectures will always appear, providing more challenges and opportunities; someone must be there to create the systems on which applications depend.

My research has been featured in the popular press. Reports on *parasitic computing* have appeared on National Public Radio, BBC Radio, Australian Broadcasting Company Radio, and the Associated Press. There is interest on several levels. The popular press is most interested in the ethical and legal issues. However, many in the technical community are interested in the new paradigm where one computer unwittingly performs calculations for another computer.

This research is also capturing industry attention. Three of my Ph.D. students, who worked on *Mona* and *Magi*, founded a startup company that was purchased by Intel in the beginning of 2001. That company built intelligent memory modules, and uses a software model that is essentially the *Mona/Magi* abstraction. I am developing a collaboration with Micron, which is prototyping an active memory part. The collaboration focuses on adapting the *Mona/Magi* interface to their part, in order to deploy it as an “accelerator” for host computations, such as encryption. Additionally, a company has expressed interest in licensing the patent-pending *Web Booster* technology.

The paragraphs below discuss eight projects that I lead. They are listed roughly in chronological order. The references to papers and awards correspond to the numbers in my vita.

Filaments One of the reasons parallel programming is difficult is the vast variety of machines on which a program must run. *Filaments* is a library package that can be used to create architecture-independent parallel programs—that is, programs that are portable *and* efficient across vastly different parallel machines. *Filaments* virtualizes the underlying machine in terms of the number of processors and the interconnection. This simplifies parallel program design in two ways. First, programs can be written (or generated) with the focus on the parallelism inherent in the application, not the architecture. Second, programs can be written that use familiar shared-variable communication. Most importantly, applications programmed in *Filaments* run efficiently. The *Filaments* package achieves much of its efficiency by making decisions at runtime. Runtime decision making leads to a simpler interface—because decisions are implicit. Additionally, it can lead to better decisions—because more information is available at runtime. Moreover, a runtime system can provide dynamic, adaptive solutions—such as load balancing—that cannot be provided statically.

There are four primary contributions of *Filaments*. First, the *Filaments* runtime uses fine-grain, over-specification of parallelism to abstract away the number of processors. Second, the *Filaments* distributed shared memory (DSM) system was the first multi-threaded DSM, which allows overlapping of communication with computation, mitigating communication latency. Third, runtime decision making enables *Filaments* to adapt to the applications needs and execute efficiently. Fourth, it was one of the first systems to efficiently support parallel computing on clusters.

Notes: This research project has produced three journal articles [2, 3, 4], three conference articles [8, 10, 19], and a users’ manual [29] for the software release [31].

Mona and Magi An *extensible* system can be customized for a particular task. Customization can improve performance, and it can provide greater functionality. The *Modify-on-Access* (Mona) filesystem is an extensible filesystem. Mona uses modular streaming operations—called *transformations*—to seamlessly perform actions on behalf of a user process during a read or write. As a result, the Mona filesystem extends the capabilities of conventional filesystems. Mona raises the level of the filesystem: It creates a new abstraction, the *active* filesystem.

Magi is an adaptation of Mona that pushes computation onto intelligent devices. The Magi system can distribute computations between the host processor and device processor. This system has a primitive mechanism for re-distributing computation to balance load between host and device. One of the more interesting outgrowths of this research is that Magi allows experiments regarding the tradeoffs between bandwidth, latency, and computation. For example, compression can be performed at either end (host or device) in order to reduce bandwidth. Often, the best way to partition work between host and device is not clear. Magi provides an excellent vehicle for evaluating these tradeoffs.

Mona/Magi has attracted interest from industry partners. As mentioned, there is Intel—which bought my students’ company—and Micron—which is discussing a collaboration. Additionally, Cray is considering the Mona/Magi interface for their next-generation product planned for 2010. This work is being conducted in the *Cascade* project, which is funded by a DARPA High-Productivity Computing Systems grant that is directed by Cray.

The primary contributions of the Mona/Magi project are as follows. First, it supports both kernel and user-level extensions, providing flexibility beyond any other extensible system. Second, it supports fine-grain (per file) extensions dynamically. Third, unlike other systems, it is fully configurable—supporting arbitrary fan-in/out and expanding/contracting streams. Magi is one of the first systems created for intelligent devices.

Notes: This research project has been supported by four awards (1, 3, 5, & 10), the third is an NSF CAREER Award. It has produced two journal articles [6, 7], three conference articles [12, 14, 18], and a users’ manual [30] for the software release [33]. One Master’s Thesis came from this work [Kendall].

PIM Runtime System Processing in memory (PIM) combines logic and memory in the same chip. There are several advantages to this, such as reduced cost, power, and latency. A system using PIM creates a fundamentally different machine because memory is *intelligent* or active. There are new tradeoffs to be explored in a PIM system. For example, on a page fault, there are now two actions that can be taken. The data can be fetched, as in a conventional machine, or the computation can be pushed. The PIM runtime system is exploring the terrain exposed by intelligent memory.

There are two primary contributions of this project: inter- and intra-node runtime development. The inter-node runtime unifies multiple PIM nodes and provides for integration. For this runtime, we are developing a new memory model and semantics, which is necessary to fully utilize active memories. A significant development is the idea of *embedded meta-data*, which enables data structures to be self-describing. Embedded meta-data allows for more autonomy, which in turn enables PIM nodes to be more capable and effective. The intra-node runtime is creating several new paradigms for managing the new hardware features and providing for the unique needs of a memory runtime system.

Notes: This research project has been supported by three awards (2, 4, & 6). It has produced four conference and workshop articles [11, 13, 15, 24] and a Master’s Thesis [Schermerhorn].

Web Booster A modern web server experiences very peaky loads, with peaks up to ten times the average load. A site built for peak capacities is expensive and underutilized, but building for less risks losing or alienating customers. The *Web Booster* is a novel web server architecture that handles peak loads without

over building sites. A booster node sits between clients and a web server. An *accelerator*—which is a downloadable kernel module—executes on the web server, and interfaces with the web booster. Unlike caches and proxies, the booster can be instantaneously activated, allowing it to service transient peak loads.

There are three primary contributions of this work. First, a detailed study of the overheads of request processing. Second, the design and implementation of the Web Booster architecture, which can instantaneously offload work during peak periods. Third, several novel optimizations were designed and implemented, such as a server packet cache. Also, this is an important line of research. There are at least seven startup companies that promote web server architectures for decreasing server load. Our research, although quite similar, was independently developed. It is to our knowledge the only academic effort in this area.

Notes: This research project has been supported by an IBM Faculty Development Award, that was subsequently renewed (7 & 9). It has produced a conference article [17] and a Ph.D. Dissertation [Panteleenko]. Additionally, three articles have been submitted [26, 27, 28].

Power Aware Runtime System In most applications, raw processing performance is not the paramount issue, rather power is more critical. For example, high-end machines cannot dissipate the heat that they generate, and portable machines cannot afford to squander their limited energy. Consequently, power performance is the new frontier. We are developing PARTS (the power-aware runtime system) in order to leverage the power-aware features of new architectures, such as the AMD Mobile Athlon, Transmeta Crusoe, Intel X-Scale, and the Notre Dame *Morph* project. These microprocesors provide an energy “gear,” which sets the energy/performance ratio. It is easy to determine what gear to use in periods of peak or zero performance needs. But what gear to use on intermediate loads, and how important it is to have accurate information about the dynamic situation, is largely an open issue. State of the art power-aware runtime systems are at best *ad hoc* and primitive.

In our initial undertaking, we developed a kernel patch for Linux that dynamically throttles processor performance—thereby reducing the power demand—in order to extend battery life in a laptop computer. Using feedback regarding the energy consumed, this system guarantees that the energy in a battery will not be exhausted before a target time. An additional undertaking is creating a power-aware filesystem. It places frequently used disk blocks on an active, power-efficient disk and unused (or less used) blocks on (hopefully) inactive disks. Using filesystem traces and disk performance characteristics, this partitioning scheme is projected to reduce the power consumption in a heavily-used filesystem by more than an order of magnitude.

The primary contribution of this project is a working prototype, implemented in the Linux 2.4.18 kernel, that throttles power consumption to extend battery lifetime. A second contribution is the design of a power-aware filesystem. Another contribution of this project has been identifying the potential for energy saving. There is tremendous opportunity to save energy, but current mechanisms are leaving most of this energy savings on the table. Lastly, this effort has produced some application-specific scheduling policies that provide improved energy savings.

Notes: This research project has been supported by an award from DARPA (8). It has produced two conference articles [16, 23] and a Master’s Thesis [Minerick]. Also, a prototype system has been released [34].

Optimized Hybrid Messaging The predominant choice for high-performance machines is clusters of shared-memory multiprocessors (SMPs). Such clusters are popular in department-wide as well as national laboratory settings. Because an SMP cluster is a hybrid (with both shared- and distributed-memory) that employs two memory models, it is hard to program, debug, and tune. As a result, many programmers write pure message passing programs for such a hybrid cluster.

Consequently, most high-performance message passing systems (e.g., MPI) have added efficient local messaging (i.e., IPC) in order to support SMP clusters. However, these systems do not take full advantage

of shared memory. Our *Optimized Hybrid Messaging* (OHM) system provides mechanisms for optimized message passing, such as 0-copy and delayed copy-on-write. These optimizations can reduce the cost of local messaging by an order of magnitude.

Additionally, we have developed a program optimizer that optimizes message passing programs. For example, in order to send a 0-copy message, the message data must be created in shared memory. The optimizer transforms programs such that the data structure is initially allocated in shared memory. Other transformations involve moving and aggregating send and receive calls. Interestingly, many of these program transformations optimize remote communications as well. Therefore, this optimizer and the local message passing optimizations will also optimize remote message passing. This is particularly useful when the remote message passing interface supports user-to-user transfer, such as done by VIA.

One primary contribution of this work is the development of several shared-memory message passing mechanisms. Another contribution is the ongoing development of rules for selecting the optimal mechanism based on a handful of attributes about the message data (such as size and access pattern). The third contribution is several program transformations which improve the efficiency of message passing programs. Last, Cascade is supporting this work as well as the Mona. Our research is especially interesting to Cray because shared-memory systems do not scale sufficiently for the project Cray plans for 2010.

Notes: This research is supported by an award from DARPA (10) and one is in preparation to NSF (12).

Parasitic Computing *Parasitic computing* is a recent discovery in which one computer (the parasite) is able to extract computation from another computer (the host) merely by engaging it in communication. This is a fundamental paradigm shift from distributed computing because the target computer is an unwitting participant. Parasitic computing has started a world-wide discussion on the ethical, legal, and technical ramifications regarding Internet parasites.

There are three contributions of this work. One is in detection, where we have developed rules that automatically catch some parasites. Another is a catalog or taxonomy of exploits of public protocols. We have found numerous parasitic techniques. Finally, we are developing additional techniques for parasitic computing. The result of this research will be a greater understanding of the fundamental elements of Internet communications, which will lead to a more robust and capable Internet.

Notes: This research has produced a journal article [5] and a magazine article [25]. It has also drawn significant attention from the popular press.

Understanding Open Source Software Development This multi-disciplinary project studies the open source software (OSS) phenomenon, developing empirical models and simulations of the interactions between developers and software projects. The project has three interrelated objectives: 1) to develop a conceptual, explanatory model of the OSS development phenomenon, 2) to develop an understanding of the social and task dynamics that predict developer (and community) attachment and retention with the explicit goal of being able to model or predict emerging characteristics, and 3) from a social network perspective, to model OSS developers as a collaborative network with dynamic attachment properties. The project draws on computer science, operations research, and sociology.

Our primary contribution is that we have shown that the OSS developer network can be modeled as a collaborative network (similar to actor networks). Additionally, we have developed agent-based models of this network.

Notes: This project is just beginning under a recently awarded NSF grant (11). This research has produced three conference and workshop articles [20, 21, 22].