Student Research Poster Competition
March 7, 2008

Table of Contents

Thanks to

Microsoft®

for sponsoring the prizes

Aggressive Server Consolidation through Pageable Virtual Machines

Anton Burtsev Mike Hibler Jay Lepreau

School of Computing, University of Utah

Abstract

We extend the Xen virtual machine monitor with the ability to host a hundred of virtual machines on a single physical node. Similarly to a demand paging of virtual memory, we page out idle virtual machines making them available on demand. Paging is transparent. An idle virtual machine remains fully operational. It is able to respond to external events with a delay comparable to a delay under a medium load.

To achieve desired degree of consolidation, we identify and leave in memory only a minimal working set of pages required to maintain the illusion of running VM and respond to external events. To keep the number of active pages small without harming performance dramatically, we build a correspondence between every event and its working set. Reducing a working set further, we implement a copy-on-write page sharing across virtual machines running on the same host. To decrease resources occupied by a virtual machine's file system, we implement a copy-on-write storage and golden imaging.

1 Introduction

Historically virtual machine monitors (VMMs) choose isolation as a primary goal. This design choice prohibits almost any form of resource sharing. The only shared resource is a physical CPU. Memory and disk space are statically preallocated upon creation of a virtual machine (VM). As a result 5-10 virtual machines easily exhaust memory resources of a physical host.

At the same time, there are multiple examples of virtual environments, where it is desirable to keep hundreds of virtual machines around. In many cases VMs are used occasionally, but have to be available on demand. Examples include long-running experiments in network testbeds like Emulab, months-long debugging and development sessions, experiments with unlimited number of nodes, when virtual machines are allocated on demand, honeypot installations, public machines in data centers and organizations, where users are allowed to keep a running copy of a VM, which is used occasionally.

Existing solutions optimize utilization of hardware resources by relying on combination of load balancing and migration. Application of these techniques is limited as they assume full load as a default case. Several systems increase server consolidation by using memory sharing across virtual machines. Potemkin [3] implements a virtual honeypot capable of hosting hundreds of virtual machines on a single physical node. However, it limits all virtual machines to be started from the same memory image. VMWare virtual machine monitor implements a copy-on-write memory sharing [4] across virtual machines running on the same node. However, it's unable to achieve desired degree of consolidation as it doesn't attempt to page out memory of an idle virtual machine.

Students: Burtsev. Corresponding author: aburtsev@cs.utah.edu. Poster web page: http://www.cs.utah.edu/~aburtsev/rd2008.pdf

2 Overview

In this work we aim to run hundreds of idle or lightly-loaded virtual machines on a single node. We choose sharing and server consolidation over isolation as the primary goal.

To achieve this goal, we aggressively reclaim resources occupied by an idle VM. We detect when virtual machine becomes idle. After that we infer a minimal working set needed for a continuous operation and swap out the rest of the VM's memory. If we predict a long period of inactivity, we release the physical node entirely by migrating the virtual machine along with its local file system to a special node hosting idle VMs. Here we detect all pages, which are identical across VMs and share them in a copy-on-write manner.

Being swapped, a virtual machine remains fully operational. It processes timer events, maintains network connections, responds to requests from remote machines and local devices.

If we detect an event, which is predicted to lead to a period of active execution, we proactively start process of swapping the VM in.

3 Implementation challenges

Idleness detection: Swap out is a trade-off of whether we can benefit from the effort involved in swapping a guest operating system out and then back in, or lose on doing this expensive operation for a short period of idleness.

We have to decide when to consider a virtual machine to be idle. A good indication that VM becomes idle is the fact that it returns to the hypervisor before exhausting its scheduling quantum. Moreover, we can stress the VM further by depriving it of CPU time. For that we gradually reduce length and frequency of the VM's CPU quantum. We combine these techniques with existing approaches for detecting and predicting periods of idleness [2].

Timely response from a swapped VM: The core idea of our approach is to provide a timely response from a swapped virtual machine. Keeping the small working set

	Working Set Size	
	4KB Pages	MBs
Idle VM	445	1.8
Idle Bash shell	725	2.9
Ping response	776	3.1
Idle ssh	805	3.2
SCP out 20MB 348KB/s	856	3.5
SCP in 20MB 348KB/s	7723	31.6
Find / -name *	6098	24.9
One minute Vim editing session	1432	5.8

Table 1: Working set size of an idle Linux VM

in memory, we create an illusion of running VM. To optimise event processing in the swapped state, we try to predict the next event and swap in the corresponding working set proactively.

Our default heuristics is to prepare for processing of the most frequent events, which are incoming network packets and timer interrupts. Therefore, we keep in memory an active set corresponding to these events. If we predict that the next event is wake up of a guest task, we try to extract information about the task from the guest OS and swap in its working set.

To estimate a working set size for typical "idle" activities, we instrumented the Xen VMM to measure a working set size of a Linux VM (Figure 1). The initial experiments show that in a typical idle case, the VM's working set varies from 1.8MB to 3.2MB. This shows the possibility to host several hundreds of VMs on a single host.

Memory management: There is a general observation allowing us to significantly reduce amount of memory occupied by a single virtual machine. Virtual installations run VMs from a small set of preinstalled operating system images. As a result, many identical memory pages can be found and shared across virtual machines. Shared pages can be identified either by comparing page table hashes [4], or, more efficiently, by monitoring VM's disk I/O [1].

The Xen VMM provides two modes of managing VM's memory. Normally Xen employs advantages of paravirtualization and speeds up memory translation by directly exposing physical memory layout to a guest operating system. This means that it is impossible to implement a copy-on-write sharing transparently to the VM.

In the second mode of operation Xen maintains a shadow page table for every VM. Normally, the shadow page table is synchronised with the one used internally by the VM. However, both page tables may not be identical. The shadow page table translation comes at a performance price. Earlier approaches used the shadow page tables to implement a copy-on-write memory sharing [3]. At the moment, it is unclear if such solution is optimal.

File system sharing and migration: Similarly to the memory, file systems of all virtual machines initialized from the same system image are almost identical, and vary only with a small set of write changes. To reflect this fact, we use a "golden" file system image, from which we start all VMs.[1] The golden image remains unchanged and thus can be shared across any number of VMs. To provide the VM with a write access, we developed a fast versioning block-level storage solution. Thus, a file system of any VM is a pair of golden image and a relatively small set of write changes. The golden image is cached on all nodes. This allows us to migrate a file system by transferring only the small set of write changes.

If we want to run hundreds of VMs on a single node, we have to consider the fact that any file system uses some memory for caching its metadata information. In case of a versioning storage, the metadata is extended with information about write updates. We plan to explore how this scaling can be achieved efficiently.

Network stack virtualization: Migration of multiple virtual machines on a single node results in a folding of a complex network topology into a single physical node. This requires an adequate support from a VMM's network stack. The Xen VMM uses the Linux network stack to provide network connections across VMs.

The use of a software 802.11q VLANs allows us to avoid "revisitation" problem for nontrivial topologies on a single node. However, we have to be creative about instantiating hundreds of networks on a single host, as the Linux protocol stack is not designed to provide this support efficiently. Therefore, we work on coming up with a reasonably light-weight and efficient network stack virtualization in Linux.

V2P migration: There are cases when performance or realism of execution on a real hardware becomes essential. Naturally, this contradicts to our desire to run everything inside a VM. To meet both requirements, we explore a possibility of implementing a novel feature of converting a virtual system into physical and back on the fly, without rebooting or interrupting guest OS execution. We believe that the latest hardware virtualization support makes it possible.[2]

References

[1] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: running commodity operating systems on scalable multiprocessors. *ACM Trans. Comput. Syst.*, 15(4):412–447, 1997.

[2] Richard A. Golding, Peter Bosch II, Carl Staelin, Tim Sullivan, and John Wilkes. Idleness is not sloth. In *USENIX Winter*, pages 201–212, 1995.

[3] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fi delity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.

[4] C. Waldspurger. Memory resource management in VMware ESX server. In *OSDI*, December 2002.

[1] There is a golden image for every type of guest operating system

[2] VMWare VMM already implements this as "hot cloning"

Battery-Draining-Denial-of-Service Attack on Bluetooth Devices

Sriram Nandha Premnath, Sneha Kumar Kasera
School of Computing, University of Utah
nandha@cs.utah.edu, kasera@cs.utah.edu

Introduction

Bluetooth is a powerful wireless technology capable of uniting disparate devices. It helps in forming small networks in no time, and without using any infrastructure. The ad hoc nature, low cost, low power requirements, ability to replace cables etc., has led to its successful widespread adoption and prevalence of this technology. It has been rapidly assimilated into all sorts of consumer electronic devices like like cellphones, headsets, laptops, computer peripherals, car kits etc.

Due to the open transmission medium, wireless devices are subject to a variety of attacks. Denial of Service (DoS) is the term used to refer to a class of attacks where a system providing some service is overwhelmed by malicious requests from an attacker, rendering it unusable for the legitimate users. Battery draining attack is one form of DoS attack where the system not only becomes unusable, it also loses its battery power, greatly affecting the continued operation of the device even after the attack is completed.

Wireless devices often operate on batteries. Therefore, it becomes utmost important to conserve the battery power. Battery draining attacks are a real threat for all kinds of battery powered devices. In this work, we demonstrate such an attack in a bluetooth environment. Using a laptop equipped with a bluetooth adapter, we launch an attack to reduce the life of a MotoRazr V3 phone by as much as 97%. We then discuss an approach to mitigate the effects of this kind of an attack.

Various kinds of attacks

In his work, Krishnaswami, describes three kinds of attacks: malignant attack, benign power attack, service request attack, in the context of battery powered mobile computers [1]. Racic et al. discuss another type of attack that exploits MMS vulnerabilities to exhaust a mobile phone's battery [2]. Pirretti et al. present two kinds of attacks for sensor networks: barrage and, sleep deprivation attack [3].

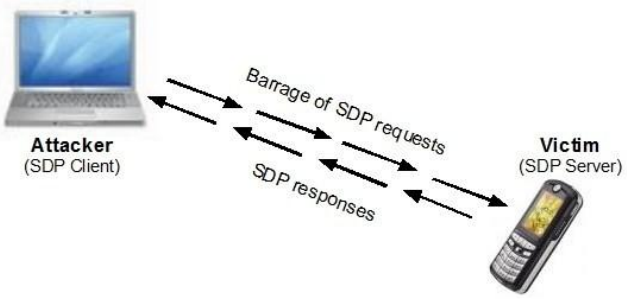


Figure 1. Schematic of our attack

Bluetooth standard describes Service Discovery Protocol (SDP) [4, 5] for finding the services offered by nearby devices. Each bluetooth device offering some service runs a SDP server. A node looking out for some service initiates the SDP client. The SDP client requests the SDP server in the neighboring nodes on information about the services offered by the device. Once the SDP client finds out the nodes that are offering the desired service, it selects one of those node and starts the service request. In our work we send a barrage of SDP requests to a victim bluetooth cellphone (Figure 1.). These repeated requests make the phone unusable by not able to receive any phone calls and the battery life is reduced by as much as 97%.

Experimentation and Results

In our experimentations, the attacker is a laptop equipped with a bluetooth adapter. The laptop was running OpenSUSE 10.3 Linux 64 bit operating system. We used the *sdptool* for demonstrating our attack. The victim is a bluetooth enabled MotoRazr V3 cellphone. The results and others details of our experiment are described in Table 1.

Our SDP barraging attack effectively cripples the victim thereby making it literally unusable, and denying the services. Compared to the stand by time of 250 hours, the victim's battery was completely drained in around 7 hours. Note that this is a passive attack, not requiring explicit action from the user of the phone. The user is completely oblivious of the attack, if he happens to be not using the device at the

time of the attack. And if at all the user tries to use the phone, it is mostly in a non-responding state.

	Experiment 1	Experiment 2
Distance between laptop & cellphone	~1 foot	~15 feet
Duration	410 minutes	435 minutes
No. of SDP server replies	6201	6415
No. of SDP client time outs	152	158
Total no. of SDP requests	6354 (= 6201 + 152 + 1)	6574 (= 6415 + 158 + 1)

Table 1. Outcome of our experiments

This attack, certainly, denies services to the phone owner. To test further, we tried to place a call to the victim node's phone number; the call was forwarded to the automated voice mail system. Some times the phone started ringing after the caller had canceled the call and not when actually the ring should have been heard!

The phone's battery was completely drained when the last SDP request was made by the attacker. On an average, about 6464 SDP requests were made in a duration of 423 minutes to bring down the phone completely. Compare this with the stand by time of the battery which is about 250 hours (15000 minutes)

We also performed a form of a slow poisoning attack where in the attacker sends requests for about a quarter of the time and sleeps for the rest of time. Roughly, this attack drained the battery in about 35 hours.

Discussion on some approaches to solve the problem

Once an attack is suspected, the victim needs to avoid communication with the attacker. This is a very challenging problem because the attacker can spoof his address, and also there is no fixed identity for devices in bluetooth – even though the devices have 48 bit hardware address, while in a piconet, they use only a 3-bit address called Active Member Address.

In our attack, the device that initiates the SDP connection is the attacker, i.e., the attacker is the master and the victim is a slave. The attacker retains the role of master until it sends SDP requests. To avoid this, we suggest that the role of the master be rotated in a round-robin fashion among all the nodes in a piconet and the role switch should occur after every pre-determined amount of time. By this way, we can reduce the damages done by the attacker.

We are also looking at developing spoofing-proof methods for identifying the attacker (based on link signatures) and, that of dropping packets from the attacker at the lowest possible layer in the bluetooth protocol stack.

References

[1] J. Krishnaswami, “Denial-of-service attacks on battery-powered mobile computers,” Master’s thesis, Virginia Polytechnic Institute and State University, 2003.

[2] Radmilo Racic, Denys Ma, Hao Chen, “Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phone’s Battery”, IEEE/CreateNet International Conference on Security and Privacy in Communication Networks (SECURECOMM '06)

[3] Matthew Pirretti, Sencun Zhu, Vijaykrishnan Narayanan, Patrick McDaniel, and Mahmut Kandemir, “The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense”, International Journal of Distributed Sensor Networks, Volume 2, Issue 3 September 2006 , pages 267 - 287

[4] Bluetooth Standards http://www.bluetooth.com/Bluetooth/Technology/Works/

[5] Jochen Schiller, Mobile Communications, Second Edition, Page 309-310.

Poster URL

http://www.cs.utah.edu/~nandha/Poster.pdf

CoGenE: A Design Automation Framework for Embedded Domain Specific Architectures

Karthik Ramani, Al Davis
School of Computing, University of Utah
Poster: http://www.cs.utah.edu/ karthikr/cogene_srpc2008.ppt

I. Introduction

The rapidly growing demand for embedded computing necessitates the creation of inexpensive low power processors targeted for battery powered systems. These processors need to be flexible enough to adapt to the ever increasing complexity and functionality of embedded applications. Examples of such applications [3] [6] are speech and face recognition, subsequently referred to as recognition. These inherently real time programs are characterized by intertwined sequential and compute-intensive parallel *kernels*. Embedded devices are required to deliver high performance for these applications in small form factors. Given the very short time-to-market constraints for embedded systems [4], the above problems make it a challenge to design programmable, high performance yet low energy devices.

Embedded devices like the iPhone support many applications and typically employ a combination of general purpose processors (GPPs) and multiple application specific integrated circuits (ASICs) [10]. While GPPs are well suited to execute sequential code, ASICs are geared towards compute-intensive kernels [4]. ASICs are expensive and time-consuming [4] to design. Moreover, their inflexible nature requires redesign when the algorithms change [11]. To overcome these limitations, commercial approaches have investigated programmable processors [2] [9] that work in different modes executing both sequential code and parallel kernels. The dynamic characteristics (supporting paper [11]) of kernels within an application domain may necessitate frequent mode changes and the resulting overheads may cause performance degradation for time critical programs. We believe that the key to solving this dilemma is to employ flexible cores that are specialized for the parallel kernels in a given application domain. Such cores are referred to as Domain Specific Architectures (DSAs). The complete heterogeneous multi-core system architecture is shown in [11]. It contains a GPP that executes the sequential code while the DSA executes the various kernels in the application domain.

Recent studies [5] [8] have investigated a DSA design methodology that employs an "ASIC-like" machine for various application domains. The approach has been successful in creating programmable DSAs that deliver one to two orders of magnitude energy-delay product improvement over commercial GPPs. However, such a design process involves significant user time and experience, in understanding the application domain, arriving at an optimal design, and in manually generating optimized code for the architecture. Given the complexity of today's applications and designs, this process is error prone, time-consuming, and may be infeasible for large search spaces. Exploring compilation techniques for code generation will significantly reduce design time and guarantee programmability of the DSA. Moreover, automatic exploration of the design space to identify optimal designs will further reduce design time. In summary, there is a need to build a flexible framework that automates the design of programmable DSAs by intelligently pruning the design space in a short time. Architectural changes impact compilation complexity and hence, the framework also needs to consider DSA flexibility while achieving superior energy-delay characteristics.

II. Research Contributions

This dissertation presents a compilation framework for automating the design of near-optimal performance-energy DSAs for three different application domains: face recognition, speech recognition, and wireless telephony. These domains are fundamentally different in their access, control, and computational characteristics [11] [8] [5] and present a diverse yet important workload [3] to test the generality of our approach. The kernels employed for recognition are generalized techniques for object recognition and can be adapted to perform other visual recognition applications. The framework consists of three major contributions: **optimized compilation, architectural simulation for performance and energy estimation, and an exploration tool for design automation of programmable DSAs**. We begin with an overview of the architectural methodology before discussing compilation and automation.

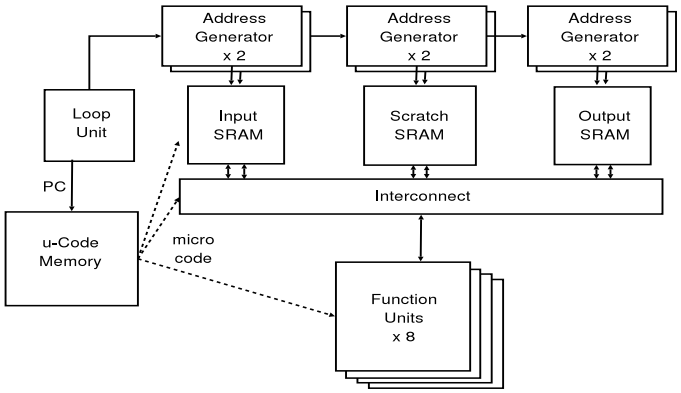


Fig. 1. DSA architectural template used in our study

The memory architecture [11] of the DSA (figure 1) consists of a hardware loop unit (HLU) and address generator units (AGU) that provide sophisticated addressing modes. This helps to increase the effective instructions committed per cycle (IPC) by performing address calculations in parallel with execution unit operations. In combination with multiple dual-buffered SRAM memories, this results in very high memory bandwidth sufficient to feed the multiple execution units. Our face recognition study [11] demonstrates that this memory system improves IPC by 4.5-10x, as compared to GPPs. The HLU also permits modulo scheduling [15] of loops whose loop counts are not known at compile time and this greatly reduces compilation complexity. The architecture is effectively a long word (VLIW) approach, but each bit in our program word directly corresponds to a binary value on a physical control wire. This very fine grained approach was inspired by the RAW project [17]. The use of programmable multiplexers (MUX) allows function units to be linked into "ASIC-like" pipelines. The output of each MUX stage and each execution unit are stored in pipelined registers. This allows value lifetime and value motion to be under program control. We show that such a programmable DSA [11] delivers approximately two orders of magnitude energy-delay product improvement over commercial GPPs. To create "ASIC-like" pipelines, the compiler generated microcode controls data steering, clock gating (including pipeline registers), function unit utilization, and single-cycle reconfiguration of the address generators associated with the SRAM ports. Thus, automatic code generation is a multi-dimensional scheduling problem.

The application suite is factored into sequential code that runs on the GPP and streaming code in C, which serves as input to our compiler framework. The Trimaran compiler (www.trimaran.org) was the starting point for the compiler development. Significant modifications were needed to transform Trimaran from a traditional cache-and-register architecture to meet the needs of our fine-grained cache-less clustered VLIW (Very Long Instruction Word) approach. The result is a compiler that is parameterized by a machine description file which speci-

fies: the number of clusters, the number and type of functional units in each cluster, the number of levels of inter- and intra-cluster interconnect, and the individual multiplexer configurations.

The program is effectively horizontal microcode which requires that all of the control points be concurrently scheduled in space and time to create efficient, highly parallel and pipelined flow patterns. To solve this problem, the **code generator employs integer linear programming (ILP) based interconnect-aware scheduling techniques** to map the kernels to the DSA. After preliminary control and data flow analysis is performed, we identify the inner most loop and load the initiation interval into the HLU. After register assignment, we perform ILP based interconnection scheduling followed by post pass scheduling to resolve conflicts. More details on compilation can be found in [11]. We have shown that interconnect-aware scheduling achieves very high functional unit utilization for such architectures. Our work in [11] [12] demonstrates the effectiveness of compilation in producing optimized code for the recognition and wireless telephony domains.

Automating the design of DSAs for constraints like minimum area, maximum performance, etc., requires that we estimate performance and energy for all design candidates. Given an architectural description file, **the compiler generates a cycle accurate simulator for performance estimation**. Our power models are based on our work in [13] [14] and we employ analytical models for all predictable structures and empirical models for complex structures like the hardware loop unit (HLU). Our area estimates are obtained from Synopsys MCL and design compiler scaled to 90 nm. Optimized compilation and architectural simulation form the basis for fast architectural design space exploration [12].

A. *CoGenE: Compilation-architectural Generation-Exploration*

Studies have investigated design automation techniques ([7], [1], [18], [16]) for application specific processors or accelerators. Some studies [7] [16] are severely limited in that they evaluate automation for one kernel phase within a multi-kernel application or focus on one subsystem within the complete processor. Our framework considers all kernels in the input application suite and optimizes all the subsystems in the DSA [12]. The approach in [18] is based on instruction set customization and this may not be amenable to extracting the unique data flow patterns in an application. The PICO design system [1] attempts to identify a cost effective combination of a VLIW GPP and an optional non-programmable accelerator for an application. The lack of flexibility in adapting to different algorithmic implementations for the same kernel is a serious drawback. Our benchmark suite employs many algorithmic implementations for the same kernel. Iterative exploration optimizes the different subsystems and automates the design of programmable DSAs.

During exploration, each design candidate is analyzed for resource addition or removal. A performance bottleneck exists if resource addition or dilation to the candidate delivers a performance improvement. Similarly, if resource removal or thinning reduces energy dissipation significantly, it indicates an energy bottleneck. Our iterative algorithm termed as "Stall Cycle Analysis" (SCA) [12], is based on the observation that total execution cycles of a program consists of cycles used for pure computation and stall cycles that occur due to the presence of bottlenecks. Bottleneck diagnosis (BD) associates stall cycles in the compilation schedule (or simulator) to bottlenecks (routability issues, insufficient parallelism, etc.) for performance or energy. During diagnosis, we investigate various solutions for each of the bottlenecks. A cost metric is assigned to each solution. The cost metric reflects the improvements in performance, energy, and code generation complexity that the solution potentially presents. In our framework, this information is fed as input to the design selection (DSel) phase. The solution with the lowest overall cost is selected to generate the next test architecture. This process is repeated in an iterative manner to arrive at near-optimal energy-performance designs for different constraints. As compared to manual designs optimized for a particular metric, SCA automates the design of DSAs for minimum energy-delay product (17% improvement for wireless telephony), minimum area (75% smaller design for face recognition), or maximum performance (38% improvement for speech recognition) [12]. In summary, the major contributions of this dissertation are:

- *Workload Studies*: To the best of our knowledge, this is the first study that investigates DSA design for a complete recognition pipeline [11]. For diversity, we employ different algorithms for recognition and also investigate the wireless telephony domain.
- *Optimized Compilation for "ASIC-like" DSAs*: For each investigated design candidate, we perform optimized compilation and quantify the impact of architectural changes on code generation. Results demonstrate the advantages of exposing sophisticated functional units to the compiler [12] and in casting domain specific functions into hardware.
- *Design Automation for embedded DSAs*: This study combines compilation and architectural simulation to explore the design space of fine grained programmable VLIW DSAs. SCA automates the design of energy efficient soft real-time embedded systems and reduces design time from man-months to hours [12].
- *Retargetable Compilation for a single DSA*: Given a face recognition DSA, our compiler can produce optimized code for speech recognition on the same machine. Our study [12] also designs a single near-optimal energy-performance DSA for all three domains. This demonstrates that our framework is flexible in adapting to changes in the application.

REFERENCES

[1] S. G. Abraham and B. R. Rau. Effi cient design space exploration in pico. In *CASES '00: Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 71–79, New York, NY, USA, 2000. ACM.

[2] G. Burns, M. Jacobs, W. Lindwer, and B. Vandewiele. Silicon hive's scalable and modular architectural template for high performance multi-core systems. In *GSPx*, Oct. 2005.

[3] P. Dubey. A platform 2015 workload model: Recognition, mining, and synthesis moves computers to the era of tera. *Intel Corporation White Paper*, Feb. 2005.

[4] J. A. Fisher, P. Faraboschi, and C. Young. *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Elsevier Inc., 2005.

[5] A. Ibrahim, M. Parker, and A. Davis. Energy effi cient cluster co-processors. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2004.

[6] J. Junqua. *Robust Speech Recognition in Embedded Systems and PC Applications*. Springer, 1st edition, May 2000.

[7] T. S. Karkhanis and J. E. Smith. Automated design of application specifi c superscalar processors: an analytical approach. *SIGARCH Comput. Archit. News*, 35(2):402–411, 2007.

[8] B. Mathew, A. Davis, and M. Parker. A Low Power Architecture for Embedded Perception. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '04)*, September 2004.

[9] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfi gurable matrix. In *FPL'03*, 2003.

[10] T. A. Press. Techies dissect apple's iphone. 2007.

[11] K. Ramani and A. Davis. Application driven embedded system design: A face recognition case study. In *CASES '07: Proceedings of the 2007 International conference on compilers, architectures, and synthesis for embedded Systems*, pages 103–114, 2007.

[12] K. Ramani and A. Davis. Automating the Design of Embedded Domain Specifi c Accelerators. Technical Report UUCS-08-002, University of Utah, February 2008.

[13] K. Ramani, A. Ibrahim, and D. Shimizu. PowerRed: A Flexible Modeling Framework for Power Effi ciency Exploration in GPUs. In *Proceedings of the Workshop on General Purpose Processing on GPUs, GPGPU'07*.

[14] K. Ramani, N. Muralimanohar, and R. Balasubramonian. Microarchitectural Techniques to Reduce Interconnect Power in Clustered Processors. In *Proceedings of the 5th Workshop on Complexity-Effective Design, held in conjunction with ISCA-31*, June 2004.

[15] B. R. Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, 1994.

[16] C. Silvano, G. Agosta, and G. Palermo. Effi cient architecture/compiler co-exploration using analytical models. *Design Automation for Embedded Systems*, 11(1):1–25, 2007.

[17] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, 30(9):86–93, 1997.

[18] S. Yehia, N. Clark, S. Mahlke, and K. Flautner. Exploring the design space of lut-based transparent accelerators. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 11–21, 2005.

Efficient Memory Safety for TinyOS 2.1

Yang Chen Nathan Cooprider Will Archer Eric Eide David Gay† John Regehr

University of Utah, School of Computing
{chenyang, coop, warcher, eeide, regehr}@cs.utah.edu

†Intel Research, Berkeley
david.e.gay@intel.com

1. INTRODUCTION

Reliable sensor network software is difficult to create: applications are concurrent and distributed, hardware-based memory protection is unavailable, and severe resource constraints necessitate the use of unsafe, low-level languages. Given these difficulties, errors such as null pointer dereferences, out-of-bounds array accesses, and misuse of union types are difficult to avoid.

Our work improves this situation by providing efficient memory and type safety for TinyOS 2 applications running on the Mica2, MicaZ, and TelosB platforms. Our approach to efficient, backward-compatible, safe execution for sensor network nodes is *Safe TinyOS*. Safe execution ensures that array and pointer errors are caught before they can corrupt RAM. In summary, our contributions include:

- extending the nesC language and compiler to support safety annotations;
- finding previously unknown bugs in TinyOS;
- and showing that safety can be exploited to increase the availability of sensor networks applications even when memory errors are left unfixed.

2. BACKGROUND

TinyOS. TinyOS 2 [4] is the dominant system for programming wireless sensor network devices. A TinyOS application is an assembly of components plus a small amount of runtime support. A programmer writes a few custom components and links them with components from the TinyOS library. Components are written in nesC [3], a dialect of C. The nesC compiler translates an assembly of components into a monolithic C program, which is then compiled and optimized by GCC.

Deputy. Deputy [1] is a source-to-source compiler for ensuring type and memory safety for C code. It is based on the insight that the information necessary for ensuring safety, such as array bounds, is usually already present somewhere in the program. A programmer must inform the compiler of this previously implicit information using type annotations. Code compiled by Deputy relies on a mix of compile- and run-time checks to ensure that these annotations are respected, and hence that type and memory safety are respected.

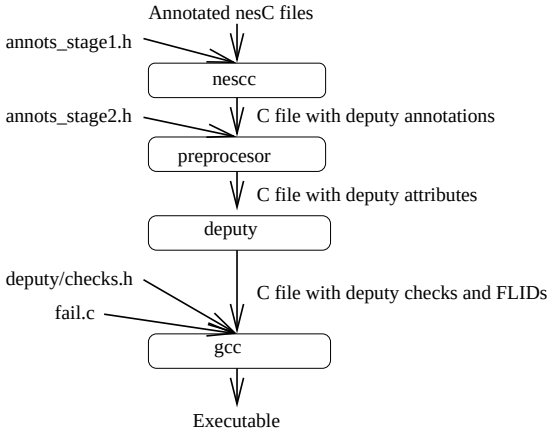


Figure 1: The Safe TinyOS toolchain

Related Work. This abstract represents our current effort to integrate our previous research [2] into the main TinyOS distribution. Condit et al. [1] present a one-paragraph summary of early experiences using Deputy with TinyOS. Our previous publications [2, 5] provide a more extensive discussion of related work.

3. PRACTICAL SAFETY FOR TINYOS

Safe TinyOS is our software platform—component set and toolchain—for sensor network applications that are fail-fast with respect to software defects that cause memory access errors. Figure 1 shows the Safe TinyOS toolchain. Safe TinyOS uses a modified nesC compiler to process annotated nesC components and then calls Deputy to perform two source-to-source transformation steps. Deputy adds safety checks as normal and then compresses bulky diagnostic information about safety violations.

Supporting Deputy annotations in nesC. In Safe TinyOS components, Deputy annotations appear in nesC source files. These source files are translated to (Deputy-annotated) C code by our modified nesC compiler. Deputy's annotations do not have any effect on nesC's compile-time checking and code generation, but they must be preserved and sometimes trans-

```
// Loop to find a free buffer
for (i = 0; i < NUM_BUFFERS; i++) {
    if (m_pool[i].msg == NULL) break;
}
// Loop post-condition: either
//    (a) i indexes a free buffer
// or (b) i is an out-of-bounds index
//        pointing one entry past the end
//        of the buffer pool

// If case (b) holds, the following access
// violates memory safety
if (m_pool[i].msg == NULL) {
    // The following store corrupts RAM in
    // the unsafe application when case (b)
    // of the post-condition holds and the
    // null check (by chance) succeeds
    m_pool[i].msg = _msg;
}
```

Figure 2: Code for bug (comments are ours)

lated by the nesC compiler.

Handling safety violations. For Safe TinyOS, we replace Deputy's verbose error-handling routines with a custom, resource-conserving system. The code that is output by Deputy contains special error-handling statements, inserted by Deputy, that use verbose strings to describe run-time safety violations. We added a pass to Deputy that replaces these verbose but constant strings in the trusted error-handling code with small integers, *fault location identifiers* (FLIDS), that represent those strings. FLIDs can be turned back to the complete strings that they represent by our tool that runs separately from the sensor network.

4. THE COST OF SAFETY

Making TinyOS applications safe incurs costs in two primary areas: source-code modifications and mote resources.

Code annotations and modifications. We are making very few changes to existing TinyOS 2 code in order to create Safe TinyOS, and the changes that we do make were generally simple and straightforward. Most changes are located in the core components of TinyOS, where they can be directly used by application developers.

Resource costs. We measured the resource costs of safety in applications that are compiled by the previous version of the Safe TinyOS toolchain. Our results showed that Safe TinyOS yields no RAM overhead and only modest overhead in code size (ROM) and CPU utilization.

5. THE BENEFITS OF SAFETY

This section describes one of several bugs that we found in TinyOS. The code shown in Figure 2 was taken out from the source code of a time-synchronization and leader-election application for TelosB motes. After processing by the nesC compiler, the original code was 13,800 non-blank, non-comment lines of C.

The developers of this application experienced unpleasant symptoms: after about 20 minutes of execution, RAM on a node would become corrupted, causing the node to drop out of its network. Our safe version of this application signaled a fault after running for about 20 minutes. The problem, shown in Figure 2, was an out-of-bounds array access in the line that tests `m_pool[i].msg` for null.

Although this bug had proven almost impossible to find, it is easy to fix: the index variable `i` must be bounds-checked before being used in pointer arithmetic. The bug is an unfortunate one because it only manifests when the send buffer overflows—an event that is expected to be quite rare. The application's authors confirmed that this bug was responsible for the RAM corruption they were seeing.

6. CONCLUSION AND FUTURE WORK

We have presented *Safe TinyOS*, our software platform for improving the dependability of sensor network applications by enforcing both type and memory safety. Safety helps developers catch bugs before a sensor network is deployed, and—equally important—it prevents memory access errors from cascading into random faulty behaviors in the field. Our effort shows that Safe TinyOS is a *practical* system for the development of reliable sensor network software.

In the near future we plan to:

- Support platforms other than Mica2, Micaz, and TelosB;
- Integrate Safe TinyOS with a stack depth analysis tool to avoid stack overflows;
- Solve the problem of unsafe accesses to pointers to dead stack frames – these are not covered by Deputy;
- and make safe execution the default for TinyOS.

Safe TinyOS home:
http://www.cs.utah.edu/ coop/safetinyos
Poster:
http://www.cs.utah.edu/ chenyang/safe_tinyos/SRPC08-safetinyos.pdf

7. REFERENCES

[1] J. Condit, M. Harren, Z. Anderson, D. Gay, and G. C. Necula. Dependent types for low-level programming. In *Proc. 16th European Symp. on Programming (ESOP)*, Braga, Portugal, Mar.–Apr. 2007.

[2] N. Cooprider, W. Archer, E. Eide, D. Gay, and J. Regehr. Efficient memory safety for TinyOS. In *Proc. of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007)*, Sydney, Australia, Nov. 2007.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc. of the Conf. on Programming Language Design and Implementation (PLDI)*, pages 1–11, San Diego, CA, June 2003.

[4] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz. T2: A second generation OS for embedded sensor networks. Technical Report TKN–05–007, Telecommunication Networks Group, Technische Universität Berlin, Nov. 2005.

[5] J. Regehr, N. Cooprider, W. Archer, and E. Eide. Efficient type and memory safety for tiny embedded systems. In *Proc. of the 3rd Workshop on Linguistic Support for Modern Operating Systems (PLOS)*, San Jose, CA, Oct. 2006.

Exploring timing based side channel attacks against 802.11i CCMP

Suman Jana
School of Computing
University of Utah
suman@cs.utah.edu

Sneha Kasera
School of Computing
University of Utah
kasera@cs.utah.edu

1. INTRODUCTION

Security has long been a weak point of IEEE 802.11 wireless networks. Wired Equivalent Privacy (WEP) had several major vulnerabilities. To ratify these, IEEE has come up with a new wireless security standard called 802.11i. It recommends all new wireless hardware to use Counter Mode with Cipher Block Chaining Message Authentication Code (CCMP).

802.11i CCMP uses Advanced Encryption Standard (AES) as its underlying encryption algorithm. Till now AES does not have any successful publicly known standard cryptanalytic attack against it. However, there are other type of attacks, which instead of attacking a cipher directly, exploits implementations of that cipher on systems which inadvertently leak data. These attacks are known as side channel attacks. Side channel attacks work on the information gained from the physical implementation of a cipher. Almost all of the real life systems provide extra information (i.e., timing information, power consumption, temperature variation, electromagnetic leaks etc.) to an attacker.

A timing based side channel attack records precise timing of data moving in or out of the CPU, or memory, on the hardware running the cipher or the cryptosystem. These information sometimes can be helpful in decreasing the complexity of standard cryptanalytic attacks thus making them more feasible. For example, in some cryptosystems it sometimes might be possible to determine how long a key is by monitoring the time taken for reading the key information.

Most of the cryptosystems which uses data dependent branch instructions are vulnerable to timing based attacks. AES implementations do not use any data dependent branch instructions. However, they use precomputed table lookups into cached memory. This causes cache collisions which in turn results in timing variations in lookups. These timing variations can be used to construct timing attacks against AES based cryptosystems.

2. OUR FOCUS & APPROACH

AES is an iterated cipher. Each round i takes a 16-byte block of input x_i and a 16-byte block of key material k_i, producing a 16-byte block of output $x_i + 1$. Each round consists of performing four algebraic operations - Byte-for-Byte substitution from a S-box, a rearrangement of bytes comprising rotating a row or column by some number of cells, and replacing one 4-byte column with another 4-byte column (MixColumn) on x_i, then taking the exclusive-or with the key material for that round k_i. Performance-sensitive software implementations of AES combine first three operations and pre-compute the values. The values are stored in large lookup tables each mapping one byte of input to four bytes of output. Variable time lookup in these tables caused by cache collisions is the source of timing attacks against AES.

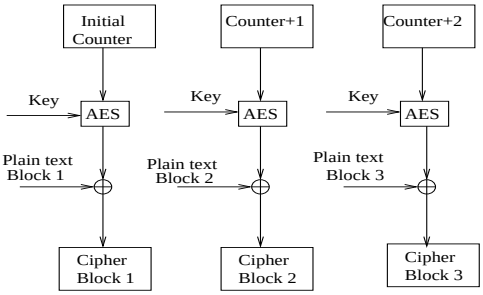


Figure 1: Counter mode using AES

Bernstein presents a timing attack against AES in [1]. He noted that the input bytes to the first round of AES encryption are the bytes $x_i^0 = p_i \oplus k_i$ and these bytes are used to index the lookup tables. This causes the entire encryption time to be affected by each of the values x_i^0. This attack takes approximately $2^{27.5}$ samples.

Bonneau *et.al.* [2] presents another cache access pattern based timing attack on AES which works by gathering timing information on AES final round and uses it to launch an attack to recover full AES key. In this work they have shown under optimal scenarios their attack requires 2^{13} time samples to fully recover AES key material.

However, All these attacks are tested on general purpose PCs. Our focus is to extend these attacks to

802.11i networks where aside from client PCs encryption/decryption is also performed at accesspoints (APs). 802.11 APs are normally small special purpose devices having significantly slower processors and less cache than general purpose PCs. These factors together can make these attacks much more effective on an AP than on a PC.

[2] and [1] have demonstrated their attacks using AES on 16 byte plaintexts. However, in real systems data blocks of much larger size need to encrypted. There are several different modes of operation (i.e., Electronic Code Book (ECB), Cipher Block Chaining (CBC) etc.) which enables AES to encrypt data blocks which are larger then the block size of AES (16 byte). 802.11i uses CBC with counter mode. In AES counter mode a counter is initialized with a start value which does not repeat. Plaintext data is broken into 16 byte blocks. For each block each successive value of the counter is encrypted using the secret key, XORed with the contents of the block to produce the encrypted data as shown in Figure 1. In 802.11 CCMP implementation a Packet Number (PN) is maintained and incremented for each processed packet. The counter value for each new packet is initialized using PN, source MAC address of the packet, flag and priority fields as shown in Figure 2.

We modify the attack presented in [1] to work against 802.11i CCMP. This work noted that the entire encryption time t can be affected by each of the values x_i^0 due to cache access patterns. So in our scheme an attacker will collect timing data for each possible values of x_i^0 on reference AP which is similar to the target AP. This data is then correlated with the data collected from the target AP to guess the value of x_i^0. As the counter value is known to the attacker, once the value of x_i^0 is known he can derive the key by performing an exclusive-or with the counter value. [2] mentioned that one of major problems with this kind of attack is that change in process load causes it to fail. However in our case (i.e., in case of APs) as opposed to PCs, the process load normally remains invariant as the processes started by the firmware are the only ones to run. This makes our attack more stable against APs using 802.11i CCMP than general purpose PCs. In CCMP an attacker can measure the time taken to encrypt a particular counter value directly by measuring encryption time of packets which are less than AES block size (128 bits). Encryption time of packets bigger than that will be equal to the total time of encrypting all the counter values used for different blocks of that packet. Though we accept that one of the potential issues this attack needs to take care of is the possibility of wireless delays outweighing the effects of cached lookups. That effect can be minimized by only considering packets with delays differing from the minimum delay by less than a certain threshold value.

Final round attack, proposed by [2], can not be effective in a remote wireless environment as it requires timing measurements of the final round of AES which will be infeasible to obtain without modifying the AP's firmware.

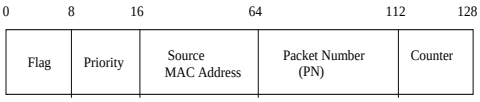


Figure 2: Counter Initialization for 802.11i CCMP

2.1 Possible Solutions We Want To Examine

The most obvious solution to this problem is to disable caching in the hardware when AES algorithm operates. However, this will cause devastating performance drop which is unacceptable under normal circumstances. Another approach to solve this problem can be to modify AES implementations to keep multiple copies of each lookup table in memory and randomly choose one of the copies of the appropriate lookup table to retrieve the value. This will increase the space overhead of AES implementations and may cause performance hit as well because of the probable loss of spatial and temporal locality. We want to investigate the exact nature of performance degradation and how does it vary with the number of copies maintained for each table.

3. FUTURE DIRECTIONS

In future, we will like to explore following ways to evaluate and improve our attack-

- Implement our attack against real-world AP and evaluate the effect of wireless delays.
- Making our attack work with less number of time samples by modifying it to exploit the structured nature of counter value as used in CCMP.
- In case of Pre Shared Key (PSK) mode of CCMP, investigating if dictionary based password guessing attacks can be used to help our attack guess the keys faster.

More graphic representation of this work can be found in `http://www.cs.utah.edu/~suman/80211iposter.ppt`.

4. REFERENCES

[1] D. Bernstein. Cache-timing Attacks on AES, April 2005 http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[2] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In *CHES*, pages 201–215, 2006.

Formal Specification of the MPI-2.0 Standard in TLA+ *

Guodong Li, Michael Delisi, Ganesh Gopalakrishnan, Robert M. Kirby
School of Computing, University of Utah
{ligd, delisi, ganesh, kirby}@cs.utah.edu

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification—Formal methods

General Terms Verification

Keywords MPI, Formal Specification, TLA+, Model Checking

1. Introduction

Parallel programs will increasingly be written using complex APIs such as MPI-2.0 [**?**], OpenMP [**?**], PThreads, etc. It is well documented that even experienced programmers misunderstand these APIs, especially when they are described in natural languages. The behavior of APIs observed through ad hoc experiments on actual platforms is not a conclusive or comprehensive description of the standard. Formal specifications, as currently written and distributed, are inaccessible to most practitioners.

In our previous work [**?**], we presented the formal specification of around 30% of the MPI-1.0 primitives (mainly for point-to-point communication) in TLA+ [**?**]. TLA+ enjoys wide usage within (at least) Microsoft and Intel by engineers (e.g., [**?**]). To make our specification accessible to practitioners, we built a C front-end in the Microsoft Visual Studio (VS) parallel debugger environment through which users can submit short (perhaps tricky) MPI programs with embedded assertions, called litmus tests. This input is turned into TLA+ code and run through the TLC model checker [**?**]. This permits practitioners to play with (and find holes in) the semantics. We also maintain cross-reference tags with the MPI reference document for traceability.

While we have demonstrated the merits of our previous work ([**?**]), this paper handles far more details including those pertaining to data transfers. In this work, we have covered much of MPI-2.0 (has over 300 API functions, as opposed to 128 for MPI-1.0). In addition, our new work provides a rich collection of tests that help validate our specifications. It also modularizes the specification, permitting reuse. The approximate sizes (without including comments and blank lines) of the major parts in the current TLA+ specification are shown in Table 1. We do not plan to model primitives whose behavior depends on the underlying operating system, The framework plus our TLA+ models can be downloaded from [**?**].

* Supported in part by NSF award CNS-0509379 and Microsoft HPC Institutes.

ACM [The preliminary version of this poster has appeared in PPoPP'08].

In [**?**], we describe an execution environment into which users can supply litmus tests and calculate their outcomes directly based on the formal semantics. The availability of a formal specification also has enabled us to create an efficient dynamic partial order reduction algorithm [**?**].

Main Module	**#primitives(#lines)**
Point to Point Communication	35(800)
Userdefined Datatype	27(500)
Group and Communicator Management	34(650)
Intracommunicator Collective Communication	16(500)
Topology	18(250)
Environment Management in MPI 1.1	10(200)
Process Management	10(250)
One sided Communication	15(550)
Intercommunicator Collective Communication	14(350)
I/O	50(1100)
Interfaces and Environments in MPI 2.0	35(800)

Table 1. Size of the Specification

TLA+ is a formal specification language based on (untyped) ZF set theory [**?**]. TLC, a model checker for TLA+, explores all reachable states in the model, looking for a state where (a) an invariant is not satisfied, (b) there are no exits (deadlocks), (c) the type invariant is violated, or (d) a user-defined TLA+ assertion is violated. When TLC detects an error, a minimal-length trace that leads to the bad state is reported (which our framework turns into a VisualStudio debugger replay of the C source).

2. Specification

We define a formal semantics for MPI primitives where execution is modeled by state transitions. The specification starts by defining advanced data structures including array, map, and ordered set to model MPI objects. For instance, MPI groups and I/O files are represented as ordered sets. A system state consists of explicit and opaque objects that may be shared among all processes or be associated with individual processes. Virtual program counters of programs/processes are also introduced to identify the current execution point. The semantics of a primitive is modeled as a $\frac{guard}{action}$ rule. When the $guard$ is satisfied, the $action$ takes effect and a transition is made.

Notation	**Meaning**
$c\ ?\ x\ :\ y$	the value of `if` c `then` x `else` y
`next`	a function returning the next label in the control flow
$\texttt{op}_l\ v$	the value returned by applying the function at l to value v
$\texttt{M}_p[x]$	variable x in the local memory of process p
$\texttt{rend}_p$	the rendezvous for the communicator to which p belongs
$\texttt{comm}_p$	the communicator to which p belongs
$\texttt{sreq}_p$	the outgoing (send) queue at process p
$\texttt{rreq}_p$	the incoming (receive) queue at process p
τ and Ψ	the status and the participants of a synchronization
$\Gamma; v$	appending value v into queue Γ

Table 2. Notations used in the semantics rules

Using the notations in Table 2, the following rules summarize the semantics of MPI primitives. A local primitive at process p only updates p's local memory. A synchronizing collective primitive is implemented by a loose synchronization protocol: in the first "init" phase, p will proceed to its next "wait" phase provided that it hasn't participated in the current synchronization (say S) and S's status is either 'e' ("entering") or 'v' ("vacant"). Note that if all expected processes have participated then S's status will advance to 'l' ("leaving"). In the "wait" phase, p is blocked if S is not leaving or p has left. The last leaving process will reset S's status to be "vacant". A similar protocol is defined for collective primitives that do not enforce synchronization.

$$\frac{\mathtt{M}_p[x]{=}v \wedge pc_p{=}l}{\mathtt{M}_p[x]{=}\mathtt{op}_l\ v \wedge pc_p{=}\mathtt{next}\ l}\ \text{local update}$$

$$\frac{\begin{array}{c}\mathtt{rend}_p = (\tau, \Psi)\ \wedge\ pc_p = l_{init}\ \wedge\ p \notin \Psi\ \wedge \\ \tau \notin \{\text{`}e\text{'}, \text{`}v\text{'}\}\ \wedge \\ \tau' \doteq (\Psi \cup \{p\} = \mathtt{comm}_p.group)\ ?\ \text{`}l\text{'}\ :\ \text{`}e\text{'}\end{array}}{\mathtt{rend}_p{=}(\tau', \Psi \cup \{p\}) \wedge pc_p{=}l_{wait}}\ \text{syn}_{init}$$

$$\frac{\begin{array}{c}\mathtt{rend}_p = (\text{`}l\text{'}, \Psi)\ \wedge\ pc_p = l_{wait}\ \wedge\ p \in \Psi\ \wedge \\ \tau' \doteq (\Psi \setminus \{p\} = \{\})\ ?\ \text{`}v\text{'}\ :\ \tau\end{array}}{\mathtt{rend}_p{=}(\tau', \Psi \setminus \{p\}) \wedge pc_p{=}\mathtt{next}\ l_{wait}}\ \text{syn}_{wait}$$

$$\frac{\mathtt{sreq}_p = \Gamma\ \wedge\ pc_p = l}{\mathtt{sreq}_p{=}\Gamma;(dst,v) \wedge pc_p{=}\mathtt{next}\ l}\ \text{isend}$$

$$\frac{\mathtt{rreq}_p = \Gamma\ \wedge\ pc_p = l}{\mathtt{M}_p[x]{=}v \wedge \mathtt{rreq}_p{=}\Gamma;(buf,src,\_) \wedge pc_p{=}\mathtt{next}\ l}\ \text{irecv}$$

$$\frac{\mathtt{rreq}_p = (buf, src, v); \Gamma\ \wedge\ pc = l}{\mathtt{M}_p[buf]{=}v \wedge \mathtt{rreq}_p{=}\Gamma \wedge pc_p{=}\mathtt{next}\ l}\ \text{wait}_{recv}$$

$$\frac{\begin{array}{c}\mathtt{sreq}_p = \Gamma_1^p; (q, v); \Gamma_2^p\ \wedge\ \mathtt{rreq}_p = \Gamma_1^q; (buf, p, \_); \Gamma_2^q\ \wedge \\ \forall (d_1, w) \in \Gamma_1^p\ :\ d_1 \neq q\ \wedge \\ \forall (buf_1, s_1, w) \in \Gamma_1^q\ :\ s_1 \neq p\end{array}}{\mathtt{sreq}_p{=}\Gamma_1^p;\Gamma_2^p \wedge \mathtt{rreq}_p{=}\Gamma_1^q;(buf,p,v);\Gamma_2^q}\ \text{transfer}$$

An asynchronous protocol is used to implement point-to-point and one-sided communication. When sending data v, process p posts a message of format $(destination, data)$ in the outgoing FIFO queue $\mathtt{sreq}_p$. In order to receive a message, p posts a request of format $(buffer, source, \_)$ in its incoming (receive) queue $\mathtt{rreq}_p$, with notation _ indicating that the data is still missing. When this "_" is filled by an incoming message, p can fetch it and updates the local memory by writing the value into the buffer. It is the rule `transfer` that models the message passing mechanism: if a message in p's outgoing queue matches a request in q's incoming queue, then the data is transferred. By matching we require: (1) the source and destination of the message and the request should match; so do their communication contexts and tags. (2) messages from the same source to the same destination should be matched in a FIFO order. A non-blocking primitive is implemented as an asynchronous operation, while a blocking operation is implemented as an asynchronous operation followed immediately by a wait operation. For example, MPI_SEND = MPI_ISEND + MPI_WAIT.

```
syn_wait(comm, p) ==
 LET i == comm.cid IN
 ∧ rendezvous[i].state = "leaving"   rend_p = (‘l‘, Ψ)
 ∧ p ∈ rendezvous[i].participants    p ∈ Ψ
 ∧ rendezvous' =
   LET members == rendezvous[i].participants \ {p} IN
      [rendezvous EXCEPT ![i] =
        [@ EXCEPT
           !.participants = members,   Ψ \ {p}
           !.state = IF members = {} THEN "vacant"
                     ELSE @   (Ψ \ {p} = {}) ? ‘v‘ : τ
        ]
      ]
 ∧ changed({rendezvous_id})
```

As an illustration, the TLA+ implementation of the syn_{wait} rule is shown above, where comments are shaded. Note that expression $s' = f\ s$ defines a transition from a state s to a new state s'; and f EXCEPT $!i = v$ defines a new map (i.e. function) with the item at i modified to v. Notation @ in $[r$ EXCEPT $!.x = @]$ stands for the (old) value of field $r.x$. Clearly this code is more readable and comprehensible than its corresponding semantics rule.

We provide comprehensive unit tests and a rich set of short litmus tests to make the specification be faithful to the English references. Generally it suffices to test Local, collective, and asynchronous primitives on one, two and three processes respectively.

Our modeling framework uses the Microsoft Phoenix compiler as a front end for simple C programs. From Phoenix intermediate representation (IR) we build a state-transition system by converting control flow graph into TLA+ relations and mapping MPI primitives to their names in TLA+. This transition system will capture completely the control skeleton of the input MPI program. Assignments are modeled by their effect on the memory. Jumps have standard transition rules modifying the values of the program counters. All processes will have the same TLA+ code. For example, the C statement "if (rank == 0) MPI_Bcast (&b, 1, MPI_INT, 0, comm1) " will be translated to the following TLA+ code, where pid and map are the process ID and the map of IR variables respectively.

$$\begin{array}{ll}\vee & \wedge\ pc[pid] = L_1 \wedge pc' = [pc\ \texttt{EXCEPT}\ ![pid] = L_2] \\ & \wedge\ mems' = [mems\ \texttt{EXCEPT}\ ![pid] = \\ & \qquad [@\ \texttt{EXCEPT}\ ![map.t_1] = (mems[pid][map.\_rank] = 0)]] \\ \vee & \wedge\ pc[pid] = L_2 \wedge mems[pid][map.t_1] \\ & \wedge\ pc' = [pc\ \texttt{EXCEPT}\ ![pid] = L_3] \\ \vee & \wedge\ pc[pid] = L_2 \wedge \neg(mems[pid][map.t_1]) \\ & \wedge\ pc' = [pc\ \texttt{EXCEPT}\ ![pid] = L_5] \\ \vee & \wedge\ pc[pid] = L_3 \wedge pc' = [pc\ \texttt{EXCEPT}\ ![pid] = L_4] \\ & \wedge\ \text{MPI\_Bcast\_init}(map.\_b, 1, \text{MPI\_INT}, 0, map.\_comm1, pid) \\ \vee & \wedge\ pc[pid] = L_4 \wedge pc' = [pc\ \texttt{EXCEPT}\ ![pid] = L_5] \\ & \wedge\ \text{MPI\_Bcast\_wait}(map.\_b, 1, \text{MPI\_INT}, 0, map.\_comm1, pid)\end{array}$$

3. Concluding Remarks

Often our formal specifications end up resembling programs written using detailed data structures, *i.e.* they are not as "declarative" as we like. We believe that this is in some sense inevitable when dealing with real world APIs. Even so, TLA+ based "programs" can be considered superior to executable models created in C: (i) the notation has a precise semantics, as opposed to C/PThreads, (ii) another specification in a programming language can provide complementary details, (iii) in our experience, there are still plenty of short but tricky MPI programs that can be executed fast in our framework. In future, we hope to write general theorems (inspired by our litmus tests), and establish them using the Isabelle theorem prover that has a tight TLA+ integration.

References

[1] The Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/.

[2] www.openmp.org.

[3] www.cs.utah.edu/formal_verification/ppopp08-mpispec/.

[4] Robert Palmer, Michael Delisi, Ganesh Gopalakrishnan and Robert M. Kirby. An Approach to Formalization and Analysis of Message Passing Libraries, Formal Methods for Industry Critical Systems (FMICS), FLoC, Berlin, 2007.

[5] Leslie Lamport, The Win32 Threads API Specification. research.microsoft.com/users/lamport/tla/threads/threads.html

[6] Leslie Lamport, research.microsoft.com/users/lamport/tla/tla.html

[7] Salman Pervez, Robert Palmer, Ganesh Gopalakrishnan, Robert M. Kirby, Rajeev Thakur, and William Gropp, Practical Model Checking Method for Verifying Correctness of MPI Programs, *EuroPVM/MPI*, 344–353, 2007.

Garbage collection in a commodity OS

Jon Rafkind

University of Utah, School of Computing
rafkind@cs.utah.edu

1. INTRODUCTION

Program reliability can be broadly separated into correctness and proper resource usage. Losing track of references to blocks of memory within a program does not affect program correctness but is a misuse of the limited memory a computing system can provide. Garbage collectors provide a means of ensuring that programs will not lose references to memory and thus provide a degree of safety for a well used resource.

Memory management in C is a difficult task to do correctly which is why tools that facilitate this work dramatically increase the reliability of a system. Garbage collectors have so far been used in the realm of applications running on top of an operating system but the operating system itself could benefit from using a garbage collector to manage its memory. No commodity operating system has yet been retrofitted with a garbage collector although many operating systems use complex allocators and reference counting to ease the complexity of memory usage. Our goal is to modify Linux, a popular open source operating system, to use a precise garbage collector to investigate the potential benefits of a garbage collector within a complex operating system.

Linux is a well tested system but is not without its share of bugs. Searching for typical problem areas on the Linux kernel mailing list, such as ”double free” and ”null pointer dereference”, yields more than three thousand hits. To fix these issues developers must manually traverse the datapaths that objects can take and see what went wrong.

We have implemented a source to source transformation for C programs based on Wick's[3] dissertation using the Cil framework that enables a C program to use a run-time precise garbage collector. The transformation can be applied to application code as well as the C code used within Linux. Unlike the Boehm[2] garbage collector, which is conservative, our collector knows exactly which words in the heap correspond to pointers. In this way our collector has the opportunity to collect all unused storage whereas conservative collectors sometimes fail to discern a large integer from a pointer.

2. BACKGROUND

Linux. Linux is an open-source and freely available operating system that has been ported to many architectures. Linux is a fully pre-emptable, multi-tasking kernel that can dynamically load modules compiled at a different time from the kernel.

Cil. Cil[1] is a package written in Ocaml that performs source to source transformations for C code. Cil can be easily extended with user created modules to perform arbitrary transformations. Our transformation is built as a module that analyzes the structure of C programs and converts them to a form that can use a precise garbage collector.

```
int foo(){
        int * x;
        ...
        return *x;
}
```

Figure 1: Original C function

```
int foo(){
        int * x;
        void * last_stack_frame = GC_last_stack_frame();
        void * simple_stack_frame[3];
        simple_stack_frame[0] = last_stack_frame;
        simple_stack_frame[1] = &x;
        GC_set_stack_frame( simple_stack_frame );
        ...
        GC_set_stack_frame( last_stack_frame );
        return *x;
}
```

Figure 2: Transformed C function

3. TRANSFORMATION

The transformation from the original C code to a form that can use a precise garbage collector is shown in Figure 2. Variables that have a pointer type and structures with pointer type fields have their address added to a shadow stack. *GC_set_stack_frame()* sets the current shadow stack that the collector will traverse during a garbage collection.

New functions are added to the source that traverse structure types and arrays. For each structure a specific *gc_tag_struct* structure is created that contain references to the **mark** and **repair** procedures a moving collector would use.

4. ISSUES

Allocators. The transformation works best on code that naively uses allocators, such as malloc, without adding in extra logic for special cases. Most applications that use facades over malloc can easily be changed to use plain malloc but the Linux kernel cannot be changed so easily. Linux has many sophisticated techniques for dealing with an allocation failure as well as uses of optimized allocators, such as *kmem_cache_alloc*, for some objects. Unfortunately the transformation cannot determine how to handle the optimized allocators so manual intervention is required to alter the original source so that the transformation can proceed as normal.

Interrupts. The Linux kernel is fully pre-emptable which allows interrupts to start executing while the kernel is executing a function. An interrupt that allocated memory while the collector was traversing pointers would cause memory corruption. It is therefore important to disable interrupts during a garbage collection until a better solution can be found.

Multiple stacks. In a single threaded user space program only one stack is ever active at a time but in the Linux kernel each user space application has a corresponding kernel stack that could be active. When a garbage collection occurs it is necessary to traverse all the kernel stacks to find all live objects. We added an extra variable to the *task* structure which defines every process in the kernel so that each process can keep track of its own stack.

5. FUTURE

A fully garbage collected kernel is a lofty goal and many technical issues still need to be worked out. Specific issues to be considered are

- Using a moving collector instead of mark/sweep
- Not disabling interrupts during a garbage collection
- Allowing binary modules to operate in the kernel

Once these issues are investigated in more depth we can more clearly evaluate the benefit of using a garbage collector within an operating system.

6. REFERENCES

[1] Berkeley. Cil. `http://manju.cs.berkeley.edu/cil/`, 2005.

[2] H.-J. Boehm. Space efficient conservative garbage collection. In *Proc. ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, pages 197–206, 2003.

[3] A. Wick. *Magpie: Precise Garbage Collection for C.* PhD thesis, University of Utah, June 2006.

ISP: A Tool for Model Checking MPI Programs *

Sarvani Vakkalanka Subodh Sharma Ganesh Gopalakrishnan Robert M. Kirby
School of Computing, University of Utah
{sarvani, svs, ganesh, kirby}@cs.utah.edu

Categories and Subject Descriptors F. Theory of computation [*F.3 Logics and Meanings of Programs*]: F.3.1 Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords MPI, Formal Verification, Model Checking, Dynamic Partial Order Reduction

1. Introduction

A significant body of parallel programs are written using MPI [1]. Bugs in MPI programs arise due to many reasons, for example: (i) during manual optimizations that turn blocking sends/receives into their non-blocking counterparts, (ii) in the context of using wild-card communications (and the resulting non-determinism), and (iii) in programs that use one-sided communication [2].

Programmers are acutely aware of the need to test parallel programs over all their interleavings. They also known that this is impossible, thus confining testing to interleavings that are guessed to be adequate – very often incorrectly, as it turns out. Model checking [4] has helped debug many concurrent systems by relying on an *exhaustive* search of all possible interleavings – albeit on a suitably abstracted system model. For example, SPIN [5] recently won ACM's prestigious software award, and has been used to verify many real-world protocols. For model checking to be successful in practice: (i) Designers must have modestly expensive techniques to create *models* of the protocols to be verified. (ii) They must be able to examine a fraction of the interleavings in a concurrent system, and be able to claim that the remaining interleavings are equivalent and hence need not be examined, using a suitable state space reduction algorithm [4, 5].

This paper presents a tool called ISP (in-situ partial order) which aspires to grow into a practical MPI model checker. In the only other active research effort on developing a model checker (called MPI_SPIN) for MPI programs [6], the following approach is taken: (i) they model MPI library functions in Promela (SPIN's modeling language), and uses SPIN for verification; (ii) they use the C extension features of SPIN to model advanced features such as non-blocking communication commands. (iii) they require users to manually express their intended MPI C program in MPI_SPIN (a macro-based extension of Promela). The drawbacks of this approach are several: users are forced to express their parallel algorithms in Promela as well as C, and maintain consistency of these representations across the program life-cycle. In addition, non-MPI library functions have to be manually broken down into Promela. ISP, on the other hand, *directly* checks MPI C programs. There is no extra work involved in handling non-MPI library functions. It employs run-time model checking methods [11] based on dynamic partial order reduction (DPOR) [12]. Our first paper on ISP [3] showed how well DPOR could potentially work on simple examples, and also confirmed all the above advantages of ISP over MPI_SPIN. In this paper, we provide an overall performance assessment of ISP. We describe our new implementation that, in addition to facilitating easy experimentation, also relies upon the formal semantics of MPI (described in [8]) more faithfully in formulating the partial order reduction algorithm. We implement more MPI primitives, including wild-card and many collectives. We present improvements possible due to a search optimization that avoids re-initializing the MPI system through an MPI_Init for each interleaving examined. Last but not least, we suggest ways to combine static analysis and ISP-based model checking.

In [9], the authors improve the coverage of MPI testing methods by forcing many more message interleavings to occur. Yet, the key property of our ISP approach is that it does not merely increase the likelihood of generating more message interleavings; it *guarantees* to cover every relevant interleaving. The work in [10] is very similar to ISP except, instead of partial order reduction, they employ iterative context bounding. They show the effectiveness of their approach on many real-world benchmarks.

2. Why Dynamic Dependence?

Partial order reduction attempts to eliminate redundant traces generated through commuting actions. Consider two C statements `a[j]++` and `a[k]--` that are concurrently enabled in two threads. These actions commute if (`j!=k`) – a fact best known at run-time (static analysis can, in general, not determine such information accurately). This *dependence information* is exploited by DPOR at run-time, thus helping to eliminate needless interleavings. What about the commuting behavior of MPI primitives such as wild-card communications? It turns out, as we show in [8], that treating them requires a DPOR approach. This is because the senders that can match a wild-card receive statement are known only at run-time. If this information is not accurately determined, then either a conservative approach has to be employed or soundness may be compromised due to an overlooked dependency.

3. How ISP Works

Currently ISP checks for deadlocks and local assertion violations (soon to be extended to check for resource leaks) in MPI programs that read all their inputs at the beginning. For every MPI function (e.g. MPI_Send), ISP provides a replacement function (a modest,

* Supported in part by NSF award CNS-0509379 and Microsoft HPC Institutes.

PPoPP'08, February 20–23, 2008, Salt Lake City, Utah, USA.
ACM 978-1-59593-960-9/08/0002.

one-time effort). When invoked, these replacement functions first consults a central scheduler through TCP sockets. If the scheduler gives permission, the replacement function invokes PMPI_Send (provided in the profiling layer of most MPI implementations, and having the same functionality as MPI_Send). This allows the scheduler to march the processes of a given MPI program according to one arbitrary interleaving, till all processes hit MPI_Finalize. ISP examines the resulting trace of actions, and records, at each of its choice points, whether a different process could have been selected, Such alternative choices are deemed necessary based on the dynamic dependence between actions in the current trace (see [8] for details). If, at choice point i, a different process $p1$ is deemed necessary to have been run, ISP (which is implemented using "stateless search" [5]) re-executes the entire MPI program till it comes to choice point i, and picks $p1$ to run. Our studies in [3] show that the DPOR algorithm in ISP can considerably reduce the number of interleavings than without DPOR.

4. Results and Assessment

We ran three simple examples: (i) the parallel trapezoidal rule computation (Trap), (ii) the control skeleton of the Monte Carlo evaluation of Pi (MC), and (iii) A byte-range locking protocol using one-sided MPI communication (BR). The byte-range protocol was actually a realistic protocol [2]. These codes as well as additional details (including ISP's code with documentation) are available from our website [1].

- For Trap, a deliberately introduced deadlock was found instantaneously. Following correction, the program was exhaustively searched over 33 total interleavings, taking 8.94 seconds. Of this time, 8.44 seconds were spent in restarting the MPI system.
- For MC, three deadlocks (which were not evident upon casual inspection) were automatically detected (detailed on our website). After correcting the code, ISP ran over 3,427 interleavings, taking 15.52 minutes, of which 15.077 minutes were spent restarting the MPI system.
- For BR, one deadlock was found after exploring 62 interleavings. After correction, the code ran over 11,000 interleavings, and the run was killed (pending further improvements to ISP before we exhaust the execution space of BR).

5. Improvements to ISP

Eliminate Restart Overhead: The restart time of the MPI system is clearly a dominant overhead. This price is being paid because as opposed to existing model checkers which maintain state hash-tables, we cannot easily maintain a hash-table of visited states including the state of the MPI program as well as the MPI run-time system. (Note: In resorting to re-execution, we are, in effect, banking on deterministic replay.) One approach eliminate restart overheads – introducing control back-edges – seems to pay-off handsomely, and works as follows. At MPI_Finalize, one can reasonably assume that the MPI run-time state is equivalent to the one just after MPI_Init, and therefore simply reset user state variables and transition each process to the label after MPI_Init (similar to the approach in CHESS). Preliminary experiments show that the execution time of MC reduces from 15.52 minutes to 63 seconds, finding the same deadlocks. For Trap, the time reduces from 8.94 seconds to 0.347 seconds.

Strength Reduction: In most MPI programs, control flows are unaffected by most (data) variables. This allows us to drop those variables (and the associated computations) not contributing to control flows or the local assertions being checked. This work is in progress.

[1] http://www.cs.utah.edu/formal_verification/ppopp08-isp/.

Barrier Insertion: We are investigating how and whether users can insert a pair of MPI_Barrier instructions, and request ISP to perform interleavings only within their confines. These barriers can, later, be moved to other parts of the MPI program. This may help users localize model checking, following their own insights into their program.

Loop Peeling: Since most MPI programs have loops from which MPI operations are invoked, it seems attractive/necessary to (through static analysis) peel out some of the iterations and have them alone be trapped and interleaved by ISP. The other iterations of the MPI program loops must still carry out these communications – however, without being trapped by ISP. We are investigating ways in which this method can be implemented without unduly complicating ISP's scheduler.

Distribute Search: ISP itself follows a highly parallelizable algorithm. The alternative interleavings may well be explored by other nodes of a large cluster. Our implementation of a dynamic partial order reduction algorithm for PThreads [7] (a different effort with no code in common) shows that with a suitably designed heuristic, linear speed-up on large clusters is a possibility.

6. Concluding Remarks

We present a practical model checking method that can detect deadlocks, local assertion violations, and many forms of resource leaks in MPI programs, without requiring users to hand-model their code as required in [5, 6].

The authors thank Salman Pervez for the initial implementation of ISP and Robert Palmer for laying down the foundations for our work.

References

[1] The Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *http://www.mpi-forum.org/*.

[2] Salman Pervez, Ganesh Gopalakrishnan, Robert M. Kirby, Rajeev Thakur, and William Gropp, Formal Verification of Programs that use MPI One-sided Communication, *EuroPVM/MPI*, 30–39, 2006.

[3] Salman Pervez, Robert Palmer, Ganesh Gopalakrishnan, Robert M. Kirby, Rajeev Thakur, and William Gropp, Practical Model Checking Method for Verifying Correctness of MPI Programs, *EuroPVM/MPI*, 344–353, 2007.

[4] E.M. Clarke, O. Grumberg, and D. Peled, Model Checking, MIT Press, 1999.

[5] G. J. Holzmann, The SPIN Model Checker, Addison-Wesley, 2004.

[6] Stephen F. Siegel, Model Checking Nonblocking MPI Programs, In VMCAI, 44–58, LNCS 4349, 2007.

[7] Yu Yang, Xiaofang Chen, Ganesh Gopalakrishnan, and Robert M. Kirby, Distributed Dynamic Partial Order Reduction based Verification of Threaded Software, In SPIN, 58–75, LNCS 4595, 2007.

[8] Robert Palmer, Ganesh Gopalakrishnan, and Robert M. Kirby, Semantics Driven Dynamic Partial-Order Reduction of MPI-based Parallel Programs, In *Parallel and Distributed Systems - Testing and Debugging PADTAD-V*, London, UK, July 2007.

[9] Richard Vuduc, Martin Schulz, Dan Quinlan, Bronis de Supinski, and Andreas Saebjornsen, Improved Distributed Memory Applications Testing By Message Perturbation, Parallel and Distributed Systems: Testing and Debugging (PADTAD - IV) 2006.

[10] Madan Musuvathi and Shaz Qadeer, Iterative context bounding for systematic testing of multithreaded programs, Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 446–455, 2007.

[11] Patrice Godefroid, Model Checking for Programming Languages using VeriSoft, In POPL, 174–186, 1997.

[12] Cormac Flanagan and Patrice Godefroid, Dynamic partial-order reduction for model checking software, In POPL, 110–121, 2005.

Leveraging Wi-PHY Measurements For Location Distinction

Junxing Zhang
School of Computing
University of Utah, Salt Lake City, USA
junxing@cs.utah.edu

Hamed Firooz
Dept. of Electrical & Computer Engineering
University of Utah, Salt Lake City, USA
firooz@ece.utah.edu

Neal Patwari
Dept. of Electrical & Computer Engineering
University of Utah, Salt Lake City, USA
npatwari@ece.utah.edu

Sneha K. Kasera
School of Computing
University of Utah, Salt Lake City, USA
kasera@cs.utah.edu

1. INTRODUCTION

The broadcast nature of wireless medium has long been viewed as a trouble maker that causes the radio channel interference, packet collision, information leakage, etc, and makes wireless networks more difficult to use than their wired peers. Our research, however, uses the complex behavior of the wireless channel to improve wireless network services. Specifically, we propose to leverage wireless physical layer measurements for location distinction.

Location distinction is the ability to detect when a device has changed its position. It is critical in many wireless network situations and applications [2, 3, 4]. In surveillance systems, video cameras, laser beams, and pressure detectors are employed to monitor any location change of valuable assets or to track the movement of a suspicious personnel. In warehouses, accelerometer based bump sensors or radio frequency identification tags are widely utilized for inventory and physical security. In sensor networks, numerous systems have been designed to sense the infrared or acoustic signals from the objects of interest for location or detection purposes. Furthermore, in wireless local area networks, location distinction can be exploited to locate wireless nodes, detect identity theft, and provide physical evidence.

Because radio channels change considerably at different locations, measurements of their characteristics are ideal for location distinction. In the following sections, we first perform a theoretical study on how to represent wireless channels and how to recover channel characteristics from sent and received signals. Then, we outline the prototype system and our research methodologies. The preliminary results are summarized both qualitatively and quantitatively in Section 4. Finally, we conclude the poster and indicate directions for future work.

2. THEORETICAL ANALYSIS

The reason that wireless channels alter greatly from location to location attribute primarily to the fact that they often consist of multiple paths. These paths are caused by reflections, diffractions, and scattering of the radio waves. As a result, when a signal is transmitted through a radio channel, the received signal contains multiple time-delayed, attenuated, and phase-shifted copies of the original signal. This process is described mathematically in the following formulas. First, a time-variant multi-path fading channel is characterized by its impulse response that can be written as:

$$h(t,\tau) = \sum_{l=1}^{L(t)} \alpha_l(t) e^{j\phi_l(t)} \delta(\tau - \tau_l(t)) \quad (1)$$

where $h(t,\tau)$ is the impulse response of the channel at time t to a signal at time $t-\tau$. It is assumed that there are $L(t)$ paths between the transmitter and receiver, and $\tau_l(t)$ is the delay of the lth path, $\alpha_l(t)$ is its gain, and $\phi_l(t)$ is its phase shift. In a short period of time, for example, for a packet duration, it is reasonable to consider the channel as a time-invariant filter with the following impulse response:

$$h(\tau) = \sum_{l=1}^{L} \alpha_l e^{j\phi_l} \delta(\tau - \tau_l) \quad (2)$$

This is the channel response represented in the time domain. Its Fourier transform $H(f)$ represents the channel response in the frequency domain.

$$H(f) = \mathfrak{F}\{h(\tau)\} \quad (3)$$

By sending s(t) through the channel, the receiver receives:

$$r(t) = s(t) * h(t) = \sum_{l=1}^{L} \alpha_l e^{j\phi_l} s(t - \tau_l) \quad (4)$$

The received signal is the convolution of the sent signal and the channel response in the time domain. In the frequency domain, it is simply the product of the transmitted signal and the channel frequency response.

$$R(f) = S(f)H(f) \quad (5)$$

In order to get the channel characteristics independent of the transmitted signal $S(f)$, the channel response must be recovered from the received signal. One of the following two methods can be used,

$$H(f) = \frac{1}{\mathcal{P}_s} S^*(f) R(f) = \frac{1}{\mathcal{P}_s} |S(f)|^2 H(f) \quad (6)$$

$$H(f) = \frac{R(f)}{S(f)} \quad (7)$$

where $\mathcal{P}_s$ is the power of the sent signal inside the band. Note that the first expression assumes that $|S(f)|^2$ is approximately constant for the given modulation scheme, and the second expression assumes that $S(f)$ is non-zero in the band of interest.

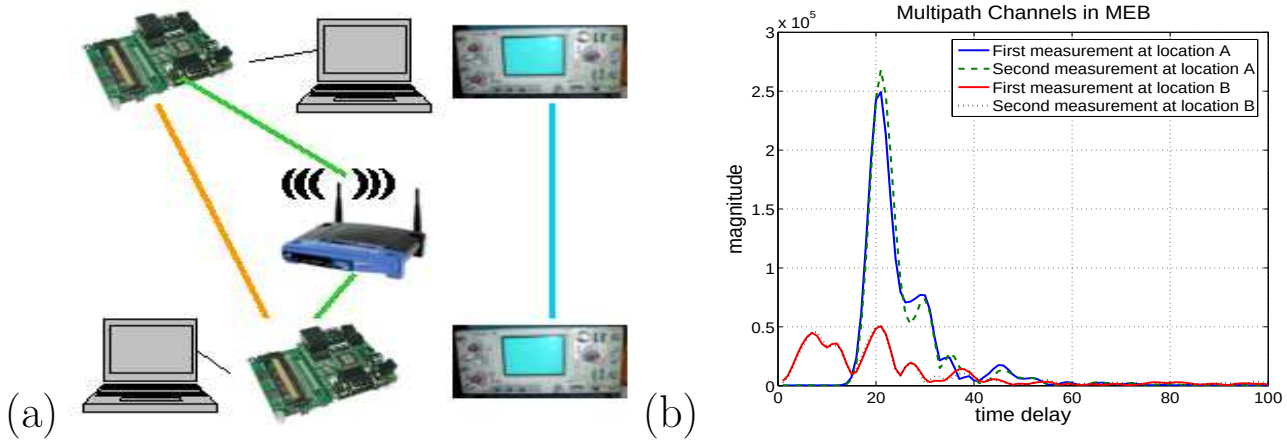


Figure 1: (a) Wi-PHY location distinction system, (b) Comparison of multipath channels.

3. EXPERIMENTATION

We implement our system with a mixture of communication systems including GNU Software Radio (GSR) systems [1, 5], a direct-sequence spread-spectrum (DS-SS) transmitter and receiver (Sigtek model ST-515), and a Linksys wireless router as depicted in Figure 1(a). DS-SS systems are good at capturing channel responses in a wide band centered at 2443 MHz. Because the DS-SS bandwidth is about four times the bandwidth of a 802.11 signal, it contains much richer channel information and provides the most sensitive measurements for location changes. On the other hand, the Linksys router and GSR systems offer the channel measurements from the most popular and sometimes over-used 802.11 band. These measurements reveals the most common performance profile for our system. In addition, the GSR systems also enable us to experiment with emerging communication technologies and customized signal processing systems such as Cognitive Radios.

In addition to the real experiments, we also use simulations for fast prototyping, testing, and verification purposes. Matlab simulink toolkit offers a good environment for the IEEE 802.11a/b/g network simulation and allows us to demonstrate how to measure wireless channels using the existing communication blocks with few added functions.

4. PRELIMINARY RESULTS

Figure 1(b) shows multipath channels measured at two locations inside the building MEB. Here the channel responses are represented with real magnitude values. It is easy to see that channel measurements at two locations are quite different (solid blue line and dashed green line versus solid red line and dotted black line) but measurements at the same location are very similar (solid blue line versus dashed green line, solid red line versus dotted black line).

Quantitatively, using a simple l_2 distance metric for comparison, our system can distinguish measurements from different locations with a detection rate of over 85% and a false alarm rate as low as 4%. If we are willing to accept a doubled false alarm rate of 8%, the system can achieve a detection rate of over 92%. We also found with better comparison methods the performance can be further improved. Therefore, measurements of physical wireless channels are indeed good indicators of locations

5. CONCLUSION AND FUTURE WORK

Channel responses can be represented in either the real domain or the complex domain. They can be described in the time domain as $h(t)$ or in the frequency domain as $H(f)$. There are many kinds of signals that can be used to measure channel responses. When it comes to comparison, a simple l_2 distance may be used, but is it the most suitable? These open questions may lead to a variety of multipath channel based solutions for location distinction. They are the driving force of our research. In pursuit of their answers, we have improved the channel measurements and metrics in the existing literature with substantial performance improvement. We also devised our own measurement and metric that makes further advancement in location distinction.

Moving forward, we plan to conduct a close study of temporal and spatial behavior of wireless channels. These behavior are instructive for contriving effective and efficient location distinction techniques. There are also a lot of applications that can benefit from location information. The availability and improvement of these applications would certainly make our wireless experience more fun and more fruitful.

A full version of this poster is available at http://www.cs.utah.edu/~junxing/poster-WiPHY4Loc.pdf.

6. REFERENCES

[1] V. Bose, M. Ismert, M. Wellborn, and J. Guttag. Virtual radios. *IEEE JSAC*, 17(4):591–602, April 1999.

[2] D. B. Faria and D. R. Cheriton. Radio-layer security: Detecting identity-based attacks in wireless networks using signalprints. In *Proc. 5th ACM Workshop on Wireless Security (WiSe'06)*, pages 43–52, Sept. 2006.

[3] Z. Li, W. Xu, R. Miller, and W. Trappe. Securing wireless systems via lower layer enforcements. In *Proc. 5th ACM Workshop on Wireless Security (WiSe'06)*, pages 33–42, Sept. 2006.

[4] N. Patwari and S. K. Kasera. Robust location distinction using temporal link signatures. In *ACM Intl. Conf. on Mobile Computing Networking (Mobicom'07)*, Sept. 2007.

[5] D. L. Tennenhouse and V. G. Bose. A software-oriented approach to wireless signal processing. In *Mobile Computing and Networking*, pages 37–47, 1995.

Leveraging 3D Technology for Improved Reliability

Niti Madan and Rajeev Balasubramonian
Appeared in the 40th Annual Symposium on Microarchitecture (MICRO), December 2007
{niti, rajeev}@cs.utah.edu

1. INTRODUCTION

The probability of erroneous computation increases as we employ smaller device dimensions and lower voltages. Firstly, bit values can be easily corrupted by the charge deposited by high-energy particles [4]. This phenomenon is referred to as a *transient fault* and it results in *soft errors* that corrupt program outputs, but do not render the device permanently faulty. Secondly, parameter variation can cause uncertainties in various device dimensions (such as gate length/width, wire height/width/spacing), that can influence the delay characteristics of circuits [2]. Dynamic conditions such as temperature, supply voltage noise, and cross-coupling effects can also influence the delay of a circuit. Because of these variations, timing constraints are occasionally not met and an incorrect result is latched into the pipeline stage at the end of that clock cycle. We refer to these errors as *dynamic timing errors.* Thirdly, processor components gradually wear out and are rendered permanently faulty, leading to *hard errors.*

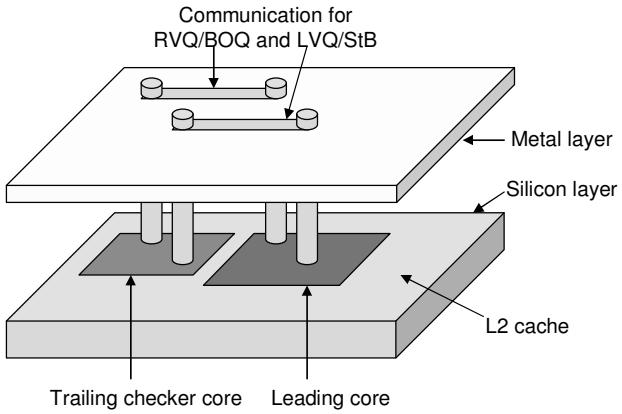
Various solutions exist to handle each of the above errors. In the first part of this work, we outline a design that represents a complexity-effective solution to the problem of comprehensive error detection and recovery. This solution builds upon a number of prior proposals in the area of reliable microarchitecture (such as [1, 3, 5]). In this design, a redundant checker core (a.k.a. trailing core) verifies the computation of a primary core (a.k.a. leading core) and the checker core employs in-order execution and conservative timing margins. This processor model is then used as an evaluation platform for the second half of the work. The 3D proposal discussed in the second half represents the primary contribution of this work.

3D technology is emerging as an intriguing prospect for the future (see [9] for a good overview). This technology enables the vertical stacking of dies with low-latency communication links between dies as shown in Figure 1(b). 3D stacking offers three primary advantages that make it a compelling research direction:

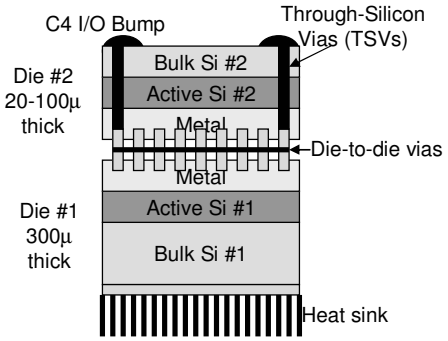
- Stacking of heterogeneous dies: A concrete application of this approach is the 3D stacking of DRAM chips upon large-scale CMPs. Inter-die vias can take advantage of the entire die surface area to implement a high bandwidth link to DRAM, thereby addressing a key bottleneck in CMPs that incorporate nearly a hundred cores [7, 8].
- "Snap-on" analysis engines: Chips employed by application developers can be fitted with additional stacked dies that contain units to monitor hardware activity and aid in debugging [6]. Chips employed by application users will not incorporate such functionality, thereby lowering the cost for these systems.
- Improvement in CPU performance/power: The components of a CPU (cores/cache banks, pipeline stages, individual circuits) can be implemented across multiple dies. By lowering the penalties imposed by long wires, performance and power improvements are possible.

All of the above advantages of 3D can be exploited if we implement the checker core on a die placed above the CPU die as shown in Figure 1(c): (i) The communication of results between cores is very efficient in terms of delay and power as short inter-die wires can be used. By isolating the checker core to a separate die, the layout and wiring of the main CPU die can be simplified. (ii) CPU dies and checker dies can be independently fabricated and chip manufacturers can offer products with a varying number of checker dies. (iii) The checker cores can be implemented in a process technology that is more resilient to soft errors and dynamic timing errors.

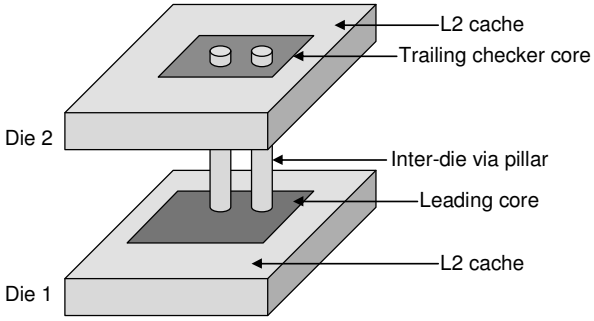
The one salient disadvantage of 3D technology is that it increases on-chip power density and temperature. Temperature increases can have a significant impact on lifetime reliability, soft error rates, leakage power dissipation, dynamic timing error rates, and performance. We consider the above effects and attempt to alleviate them with older process, deeper pipelines, and three-dimensional L2 cache organizations. We show that with the most pessimistic assumptions, the overhead of the second die can be as high as either a 7 °C temperature increase or a 8% performance loss. However, with the use of an older process, this overhead can be reduced to a 3 °C temperature increase or a 4% performance loss, while also providing higher error resilience.



(a) 2D-2a baseline processor with checker core



(b) Example 3D stack with face-to-face (F2F) bonding (based on figure in [2])



(c) 3D-2a processor with checker core on stacked die (metal layers not shown)

Figure 1: Baseline (a) and proposed (c) processor models and example F2F 3D stack (b). Figure not drawn to scale.

2. CONTRIBUTIONS

The major contributions of this work are as follows:

- We quantify the impact of a 3D checker core on power, temperature, performance, and interconnect overhead.
- We argue for the use of an older process technology for the checker die to improve its error resilience and quantify the impact of this choice on power, temperature, and performance.
- We show that a high-ILP checker can operate at low frequencies, allowing much more timing slack in each pipeline stage and greater error resilience.
- We argue against the use of deep pipelines for the checker core.

3. REFERENCES

[1] T. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of MICRO-32*, November 1999.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings of DAC*, June 2003.

[3] N. Madan and R. Balasubramonian. Power Efficient Approaches to Redundant Multithreading. *IEEE Transactions on Parallel and Distributed Systems (Special issue on CMP architectures)*, Vol.18, No.8, August 2007.

[4] S. Mukherjee, J. Emer, and S. Reinhardt. The Soft Error Problem: An Architectural Perspective. In *Proceedings of HPCA-11 (Industrial Session)*, February 2005.

[5] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed Design and Implementation of Redundant Multithreading Alternatives. In *Proceedings of ISCA-29*, May 2002.

[6] S. Mysore, B. Agrawal, N. Srivastava, S. Lin, K. Banerjee, and T. Sherwood. Introspective 3D Chips. In *Proceedings of ASPLOS-XII*, October 2006.

[7] J. Rattner. Predicting the Future, 2005. Keynote at Intel Developer Forum (article at http://www.anandtech.com /tradeshows/showdoc.aspx?i=2367&p=3).

[8] Samsung Electronics Corporation. Samsung Electronics Develops World's First Eight-Die Multi-Chip Package for Multimedia Cell Phones, 2005. (Press release from `http://www.samsung.com`).

[9] Y. Xie, G. Loh, B. Black, and K. Bernstein. Design Space Exploration for 3D Architectures. *ACM Journal of Emerging Technologies in Computing Systems*, 2(2):65–103, April 2006.

Network Simplicity for Latency Insensitive Cores

Daniel Gebhardt, JunBok You, W. Scott Lee, Kenneth S. Stevens
University of Utah, Salt Lake City, UT 84112, U.S.A.
gebhardt@cs.utah.edu, {jyou,wlee,kstevens}@ece.utah.edu

Abstract

In this paper we examine a latency insensitive network composed of very fast and simple circuits that connects SoC cores that are also latency insensitive, de-synchronized, or asynchronous. These types of cores provide native flow control that is compatible with this network, thus reducing adapter overhead and buffering needs by applying backpressure directly to the sending core. We show that under realistic traffic patterns our sample network meets performance requirements and uses less power compared to a similar design. This concept of a simplified network, along with latency insensitive cores lends itself well to meeting the needs of low-power interconnect components in future design processes.

Traffic patterns of some system-on-chips (SoCs) are known at design time. In these cases, a custom generated network topology and physical placement of components yields improved performance and power than a regular-pattern network [1].

In many embedded applications, absolute performance is not the most important trait to optimize. Power, energy, or cost are often more important so long as the system meets the minimum performance.

Latency insensitive (LI), or *elastic* design, is an area of research typically orthogonal to NoC protocols. LI protocols combine asynchronous design concepts with a standard clocked environment, and provide tolerance to variation in communication channel latency. When applied to a core's design, LI can yield power savings and other advantages [2], and may soon become a more common design style. We have developed a LI protocol (pSELF) to allow operation with both synchronous and asynchronous circuits [3].

Our network circuits can interface directly with cores that operate with a phased elastic protocol [3]. With this scheme, the core and network interfaces of a network adapter (NA) follow compatible protocols. This simplification eliminates the extra size and power requirements of interfacing with a standardized core protocol. One study indicates an OCP NA implementation can add up to 50% latency over a native interface [4].

The network fabric is implemented using two components: a phase elastic half buffer (pEHB) and a binary-tree router. A pEHB contains a data latch, and associated gates and control latches implementing the phase synchronous elastic flow (pSELF) protocol. pEHBs may be added along long links in order to pipeline them for higher clock rates. A router consists of three bi-directional ports, control logic, and output buffers. It simultaneously routes a packet on each input port to one of its two possible output ports, and arbitrates if there is output contention. Packets contain source-routing bits and are, in this case, a single flit long.

We first evaluated this network topology and protocol using a custom simulator and a 16-core binary tree network to measure throughput under traffic distributions of uniform random and Gaussian. The uniform random traffic simulates highly global communication, and because of this network's low bisection bandwidth, the throughput saturates at only 0.22 packets/cycle per core. The Gaussian traffic simulates more localized communication when a core will more frequently communicate to topologically close cores than to distant ones. We set σ in the distribution such that the nearest one-quarter of the network nodes fall within two σ from a given node. The maximum throughput in this case is much better at 0.45 packets/cycle. With this Gaussian traffic at a 30% offered load, on average one out of five packet send attempts is blocked at the sending core by congestion, compared with two out of 5 attempts for uniform traffic. These results confirm that this network would be a poor choice for connecting many cores needing high bandwidth but communicating in an unknown pattern. However, it also indicates reasonable performance for more localized traffic, while allowing efficient global communication that occurs infrequently.

We also evaluated our network on two realistic traffic models. These have been used in previous work to evaluate application-specific NoC generation techniques [5]. The expected traffic is represented as a *communication trace graph* (CTG) showing the bandwidth requirement between various cores in the SoC. The first, a Multi-Window Displayer (MWD), has mostly one or two connections per core, all with relatively similar (and low) bandwidth requirements. The second model is a MPEG4 decoder (MPEG4). This CTG is quite different, and requires a number of high bandwidth connections to a few shared memories. We construct a tree topology by pairing cores, or groups of cores and their routers, that have the highest edge-weight between them in the core graph [6]. This topology generation is important to minimize network hops on paths needing a high bandwidth (or low latency, as the need may be). The generated topology for the MPEG4 design is shown in Figure 1. We simulated these topologies using the CTG information to determine if our network could support such applications successfully. In both cases, even for the more "difficult" MPEG4 application, the topology and network maintained the average asked bandwidth for various message sizes. Assuming our network is using 4-byte wide flits, and running at 1.3 GHz (in 130nm technology), the link bandwidth is 5.2 GB/s, fast enough to provide headroom for instantaneous bandwidth needs on the combined average 1.8 GB/s path to the sdram in the MPEG4 application.

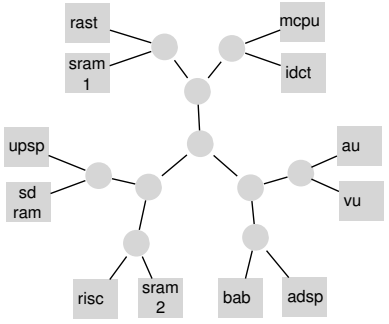


Figure 1. Automatically generated topology for MPEG-4 traffic simulation.

We evaluated the power consumption of our pSELF-based routers against a Xpipes Lite [5] 4x4 router. We used Synopsys Design Compiler and Power Compiler to estimate our router's power at the technology of our fabricated test-chip (0.5 micron), and scaled the values down for a 130 nm process using constant-field scaling theory. The power of a Xpipes 4x4 router with 32-bit flit widths and four output buffer stages is given as 55.5 mW when running at its maximum of 1 GHz. We constructed a functionally similar "4x4" router from two of our binary-tree routers and pEHBs. We estimated the FIFO buffers' power of the Xpipes router, and added that to our 4x4 router construct for fair comparison. The power for this arrangement running at 1.3 GHz is 28.9 mW, 48% less than that of the Xpipes router.

It is likely that additional internal network buffering, virtual channels, and other features could show performance improvements for certain applications. However, a simplified network concept can still offer the performance required for many designs while giving a significant power advantage. A network compatible with elastic protocols yields very efficient circuits and there is no need for extra flow-control logic in the core's network adapter, nor extra *NACK* control messages. If a core is *desynchronized* [7] there is an additional opportunity for greater power savings, and pSELF is directly compatible with the handshaking used in the desynchronization process, requiring only a minimal conversion circuit.

This text will be published in the IEEE Proceedings on NOCS. Newcastle, UK. April 2008.

References

[1] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. of ICCAD*, 2006, pp. 355–362.

[2] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. of DAC*, Jul. 2006, pp. 657–662.

[3] J. You, Y. Xu, H. Han, and K. S. Stevens, "Performance Evaluation of Elastic GALS Interfaces and Network Fabric," in *Workshop on FMGALS*. Elsevier ENTCS, May 2007.

[4] L. Ost, A. Mello, J. Palma, F. G. Moraes, and N. Calazans, "Maia: a framework for networks on chip generation and verification," in *ASP-DAC*, 2005, pp. 49–52.

[5] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. De Micheli, "XPipes Lite: a Synthesis Oriented Design Library for Networks on Chips," in *DATE*, vol. 2, 2005, pp. 1188–1193.

[6] D. Gebhardt and K. S. Stevens, "Elastic flow in an application specific network-on-chip," in *Workshop on FMGALS*. Elsevier ENTCS, May 2007.

[7] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. on CAD*, vol. 25, no. 10, pp. 1904–1921, 2006.

Path Based Network Modelling and Emulation

Pramod Sanaga Jonathon Duerig Robert Ricci Jay Lepreau
School of Computing, University of Utah
{pramod, duerig, ricci, lepreau}@cs.utah.edu

Click here for the poster

1. DESCRIPTION

Network emulation works by forwarding packets from an application under test through a set of queues that approximate the behavior of real router queues. By adjusting the parameters of these queues, an experimenter can control the emulated capacity of the link, delay packets, and introduce packet loss. Popular emulators include DummyNet [2], ModelNet [3], NISTNet [1], and Emulab (which uses Dummynet) [4]. These emulators focus primarily on *link emulation*, meaning that they concentrate on realistic emulation of individual links and queues. In order to emulate a realistic Internet path with a link emulator, one must use multiple instances of a link emulator to create a router-level topology. A model with this level of detail, however, is often not available or desirable.

Our goal is to model Internet paths as a whole rather than specific links in a router-level topology. There are three potential approaches to modeling and emulating an Internet path.

One approach is to model the Internet as a mesh of distinct paths each of which has an independent link emulator. The simplicity of the model, however, means that it cannot model crucial aspects of the Internet such as reactivity, and bottlenecks shared between paths.

Another approach is to inspect the Internet in depth to obtain a complete picture of the router-level topology which underlies it, complete with annotated capacity and measures of background traffic. While much work has been done in this area, obtaining a fully annotated topology is still a difficult problem.

We have created a system which follows a third approach. Our emulation model takes as input a number of characteristics which describe the end-to-end behavior of paths, and then *abstracts* these paths as much as possible. The parameters of our model include both an extended model of the reactivity of background traffic on the network, and relationships between paths.

We have identified four fundamental principles for modeling Internet paths. Our contribution consists of identifying these principles, exploring them in depth, and implementing them in a path emulator so that they can be empirically examined and improved.

Our four principles for modeling path emulation consist of:

- *Model capacity and available bandwidth separately.* Existing link emulators model links with limited capacity. We show why this isn't always sufficient to create a path emulation with a particular target bandwidth. Then we provide the mathematical basis for deciding how much capacity and how much cross-traffic are necessary to have the desired effect.
- *Pick appropriate queue sizes.* Much work has been done in choosing "good" values for queue sizes in real routers, but the issues that apply to emulation are somewhat different. We define concrete mathematical limits for queue sizes in emulation and simulation.
- *Use an abstracted model of the reactivity of background flows.* It is very difficult to discover the characteristics of background traffic directly. We show that a reactivity model can be created which concentrates only on the effect that reactivity has on foreground flows.
- *Model shared bottlenecks.* We model the relationships between paths that share bottleneck links by defining equivalence classes, which place multiple paths in common bottleneck queues.

*

2. EVALUATION

2.1 Bidirectional throughput microbenchmark

We start two TCP flows simultaneously on the path, one in each direction. The results with the link emulator and the path emulator are presented in Table 1 and Table 2. The *Achieved Tput* columns of each row present the achieved throughput.

*This material is based upon work supported by the National Science Foundation under Grant Nos. 0335296, 0205702, and 0338785. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Test Type	ABW (Kbits/sec) Forward	Reverse	Base Delay (ms)	Queue size (KB)	Achieved Tput (Kbits/sec) Forward	Reverse	Error (%) Forward	Reverse
Measured	2251	2202	64	73	2070	2043	8.0	7.2
	4061	2838	29	73	2774	2599	31.7	8.4
	6436	2579	12	73	3176	2358	50.6	8.5
	25892	17207	4	73	20608	15058	20.4	12.5
Synthetic	8000	8000	45	73	6228	6207	22.0	22.4
	12000	12000	30	73	9419	9398	21.5	21.6
	10000	3000	30	73	3349	2705	66.5	9.8

Table 1: Link emulator bidirectional throughput

Test Type	ABW (Kbits/sec) Forward	Reverse	Base Delay (ms)	Queue size (KB)	Achieved Tput (Kbits/sec) Forward	Reverse	Error (%) Forward	Reverse
Measured	2251	2202	64	957	2202	2163	2.1	1.8
	4061	2838	29	957	3822	2706	5.8	4.6
	6436	2579	12	844	6169	2448	4.1	5.0
	25892	17207	4	197	23237	15644	10.2	9.1
Synthetic	8000	8000	45	237	7493	7420	6.3	7.2
	12000	12000	30	158	11220	11208	6.5	6.6
	10000	3000	30	265	9150	2690	8.5	10.3
Bad Q-size	25892	17207	4	390	21012	15916	18.8	7.5
	10000	3000	30	488	7641	2768	23.6	7.7

Table 2: Path emulator bidirectional throughput

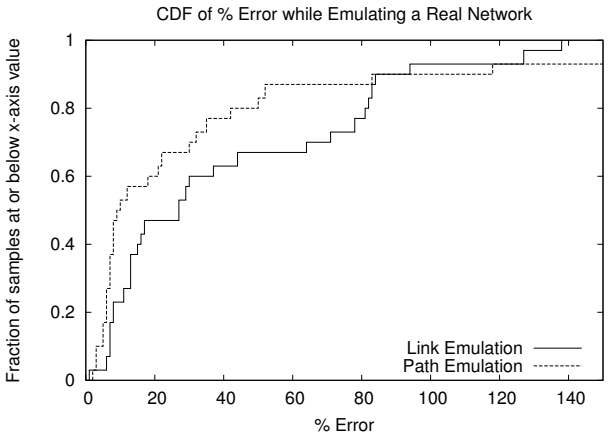


Figure 1: A CDF showing percentage of error compared to PlanetLab of an emulation based on real measurements.

While link emulation performs adequately in the first test, it is clear from the error margins in the remaining tests that it is not sufficient when emulating Internet path measurements, resulting in 50–60% lower throughput compared to the target available bandwidth. Our path emulation model makes the most difference on asymmetric paths, always being within roughly 10% of the target bandwidth.

2.2 Emulating a Real Network from End-to-end Measurements

We show how we can successfully model a network derived from real measurements. We ran bottleneck detection code on a set of six PlanetLab nodes, and gathered bandwidth, delay, and reactivity information for the paths between these nodes, and used this data as the parameters for our path emulator.

We also compare to the results obtained by using this data as the parameters to a link emulator.

Figure 1 shows that approximately 80% of the emulated paths were within 40% of the bandwidth achieved on Emulab. On the link emulator only 60% of the paths were within 40%.

3. REFERENCES

[1] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(3):111–126, 2003.

[2] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review*, 27(1):31–41, Jan. 1997.

[3] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 271–284, Boston, MA, Dec. 2002.

[4] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002.

ProtoGENI: A Network for Next-Generation Internet Research

Robert Ricci, Jay Lepreau, and the Flux Research Group
<ricci@cs.utah.edu>, <lepreau@cs.utah.edu>, http://www.flux.utah.edu

The Flux Research Group in the School of Computing is developing *ProtoGENI*, the first NSF-funded network to be a prototype of NSF's *Global Environment for Network Innovation (GENI)*. ProtoGENI will be a highly distributed network with PCs, programmable networking devices, wireless equipment, and sensor-network nodes spread across 100+ sites in the USA. Many of these sites will be connected with dedicated bandwidth, allowing ProtoGENI to be a first-of-its-kind *network embedding service* for cutting-edge research. With initial availability by mid-2008, ProtoGENI will be a platform for designing and testing radical new ideas in large-scale networks and distributed systems—including ideas that will drive the complete GENI facility and, eventually, the next-generation Internet.

New Networks From the Ground Up..

Despite the Internet's rampant success, industrial and academic experts believe that the *evolutionary* road for the Internet may be reaching an end. For both technical and economic reasons, it may no longer be possible to augment the existing Internet architecture to meet its always-widening requirements. New and *revolutionary* research is needed, if the challenges faced by today's Internet—including security, mobility, and reliability—are to be solved in future networks.

Building the "next-generation Internet" requires research that may affect all levels of the Internet's implementation, from the lowest levels of hardware up to application-level protocols. This research, in turn, requires new experimental infrastructure. Current testbeds for network research are suitable for a wide range of experiments and education, but they can be limiting because they are "stuck" in the architecture of the existing Internet. Researchers need to be able to escape these constraints so they can modify or reimplement any or all parts of a network.

The NSF GENI project was created to provide this new type of research infrastructure. It seeks to design and build a new experimental facility, called the *Global Environment for Network Innovation*, that will support experiments with radical new network designs at large scales. GENI will be a testbed capable of running multiple research activities at one time—essential for the deployment and evaluation of long-running experiments. In addition to being open to researchers, GENI will also be open to "real people": i.e., the users of today's Internet. These users will be able access the services that researchers make available to the public via GENI—whatever those future services might be!

ProtoGENI..

Prof. Jay Lepreau and the Flux Group are building the first prototype facility based on the GENI architecture. Early, usable, prototypes are essential to the success of GENI. Because the goal of the facility is to enable experimentation that can't be done today, it is important to get experience with what researchers need from this network, and how they use it, as crucial input to the design of the final GENI facility. The flux group's effort, , called *ProtoGENI*, will be a large and highly distributed testbed for radical network research. It will provide experimenters with a wealth of resources—both dedicated and shared—and almost complete top-to-bottom control over those resources.

The work required to build ProtoGENI falls into two main categories: deploying hardware at sites around the United States, and creating software to control that hardware and enable experimentation on it.

Hardware..

The wide-area architecture of the hardware to be deployed for the ProtoGENI network is shown in Figure 1. The equipment at each wide-area site will consist of two high-powered PCs, each with a NetFPGA card. These cards—developed by Prof. Nick McKeown and his team at Stanford University—allow programmers to implement their own high-speed and hardware-assisted network protocols down to OSI layer 2. Each NetFPGA card and each PC will have multiple Ethernet ports, all connected to a switch. At Internet2 sites, the switch will be connected to bandwidth that is dedicated to ProtoGENI: Utah is currently working with Internet2 to allocate this resource. At non-Internet2 sites, the

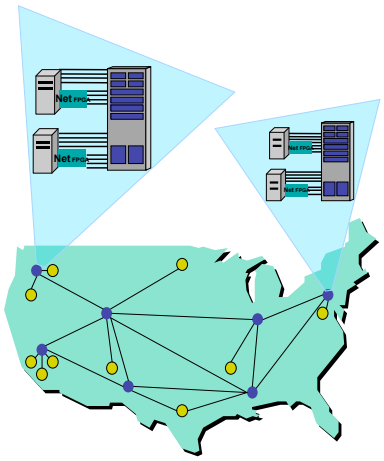


Figure 1: ProtoGENI's wide-area architecture. The ProtoGENI network will be distributed across dozens of sites around the USA. The wide-area "backbone" will consist of PCs and NetFPGA devices at Internet2 sites, which will be connected to each other via dedicated Internet2 bandwidth. Similar devices will be located at up to one hundred additional sites; these can connect to ProtoGENI via IPv4 tunnels.

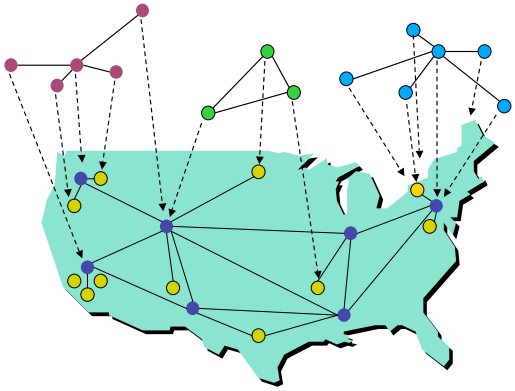


Figure 2: ProtoGENI's network embedding service. Experimenters' virtual networks, shown at the top, are emulated within ProtoGENI's physical network, shown at the bottom.

switch will connect to a (non-dedicated) uplink.

ProtoGENI will also provide access to a diverse collection of more local networks. These include two separate, multi-node wireless clusters at CMU—developed by Profs. Peter Steenkiste and David Andersen—and two similar networks at Utah.

The physical infrastructure is designed so that ProtoGENI can provide experimenters with complete networks that are "embedded" within the ProtoGENI testbed, as illustrated in Figure 2. ProtoGENI's management software will map researchers' logical networks of devices and links network onto physical resources in the testbed. ProtoGENI itself will control and configure network switches at each site, thus allowing ProtoGENI to emulate the researcher's logical network within the physical ProtoGENI network.

Software.

Much of ProtoGENI's testbed management software will be based on parts of two existing testbeds: PlanetLab and Emulab. PlanetLab was mentioned above. *Emulab* is the Flux Group's current network testbed that manages hundreds of clustered (wired) PCs, a variety of wireless devices including PCs and sensor-network nodes, and virtual devices such as VMs and network simulators. As of January 2007, Emulab had over 1,700 users from more than 250 institutions around the globe, and these users ran over 15,000 experiments in the preceding twelve months.

For ProtoGENI, Emulab's software will be adapted to the GENI architecture, including implementing the GENI abstractions, APIs, and resource models. We will also extend it to support the configuration and control of new types of hardware, such as layer 2 configuration and the NetFPGA cards. ProtoGENI will incorporate large parts of PlanetLab's software; by reusing Emulab and PlanetLab software, ProtoGENI will avoid many of the traps and pitfalls that would come with developing a new testbed from scratch. The Flux Group's years of experience with testbed software will speed the work for ProtoGENI.

The Future..

Like Utah's existing Emulab, the ProtoGENI testbed will be a resource for the network research community around the world. By mid-2008—within the first year of the project—the goal is to have a functional system in the field that includes the hardware and software needed to create and manage experiments. ProtoGENI and its initial use will be a proving ground for the GENI architecture: invaluable to the development and eventual deployment of the complete GENI.

Although much has already been done to define its architecture, GENI is very much a work in progress. The full-scale architecture is now being evolved by a broad panel of experts in the field of networking, including two from the Flux Group. Jay Lepreau and Robert Ricci, a research staff member, are members of the GENI Narrow Waist Working Group. This panel is designing the architectural components of GENI as a whole, and laying the groundwork for GENI's evolution.

For More Information..

The home page for NSF's GENI project is at http://geni.net/. Utah's ProtoGENI site is at http://protogeni.net/, and Utah's Emulab site is at http://emulab.net/. If you are doing research in networking or distributed systems, we encourage you to use Emulab today—and soon, *ProtoGENI!*

This material is based upon work supported by the National Science Foundation under Grant No. 0723248. A version of this text will appear in the Utah Teapot.

Quantifying Multi-Programmed Workload Sensitivity to Non-Uniform Cache Access Latencies

Kshitij Sudan and John Carter

School of Computing, University of Utah
{kshitij,retrac}@cs.utah.edu

***Abstract*— Extracting optimal performance from a heterogeneous application mix running on a multi-core processor is a considerable challenge due to the varied memory requirements of simultaneously executing applications. A cache management policy for such a mix of applications is a critical design problem considering larger on-chip caches with every VLSI technology generation and worsening memory access latency. Due to the variation in cache requirements of different applications, aggregate on-chip capacity can be underutilized if the management policy is inefficient. To design effective cache management policies, it is critical to first understand the cache access patterns of simultaneously executing applications. In this work we determine the last level cache access pattern of multi-programmed applications and observe the effect they have on each others performance when they are sharing a large Level-2 data cache which allows dynamic allocation of cache capacity and has non-uniform access latencies to different cache banks. We show that although multi-programmed workloads are sensitive to cache capacity, there is little impact of latency on the performance on these applications.**

I. INTRODUCTION

Given the current trend towards chip multiprocessors (CMPs) for the next leap in computing performance, many architectures have explored the design space of possible optimizations to use the resources on a CMP in the most effective manner. Among the several possible optimizations, those made to shared resources are the most crucial ones for better performance because sharing resources allows for better utilization of available on-chip resources without further increasing the cost of the microprocessor.

Sharing last-level data cache has shown the biggest cost-performance improvement, primarily because of widening gap between the microprocessor's data demands from the main memory and memory's inability to scale with each new microprocessor generation in terms of data delivery capacity. The memory bandwidth requirements increases with every new microprocessor generation because larger transistor budgets give newer microprocessors increased ability to compute. This increased computation ability puts pressure on the memory to deliver more instructions in the same amount of time as compared to previous generations, and, DRAM technology is unable to keep pace with this due to technology constraints. This problem is usually termed as "memory wall" [7] and is know to be a bottleneck.

The memory wall problem is exacerbated by CMPs because, in principle, memory now has to service more compute cores unlike the previous generation processors, where there was only one processing element on a chip. To mitigate the impact of memory wall in uni-processors, on-chip cache memories were introduced such that the most recently used data could be accessed at relatively lower access times as compared to main memory. In CMPs however, hiding long memory access latencies gets harder because caches are not large enough to successfully accommodate the working set of all the cores. Aside from this issue, another orthogonal issue which hurts performance in CMPs is the increased and differential latencies to cache banks.

With every new generation of VLSI technology access latency to cache banks increases due to increasing wire delays. As new process technologies are introduced, smaller feature sizes increase the amount of time it takes for the signal on the wire to be propagated on-chip. At the same time, with smaller feature sizes, it's possible to fabricate more cache capacity on-chip. These two factors have led to differential access latency caches and the architectures are referred to as NUCA: Non-Uniform Cache Architecture [5].

With all these hardware perturbations affecting the way microprocessors are built and architected, the problem of extracting optimal performance from a heterogeneous application mix executing on a CMP is complicated by varied memory requirements of executing applications. A cache management policy for such an application mix is thus a critical design problem. The need to have an optimal cache management policy which can reduce wastage of resources and at the same time promote sharing of data at least latency among executing applications is therefore pressing and there has been a lot of prior work in this respect [4], [1], [3], [8].

In this work we focus on determining the cache behavior of simultaneously executing multi-programmed applications. This first phase of the work can thus be classified as a workload characterization study wherein we try to determine how a heterogeneous mix of multi-programmed applications behaves. With this workload characterization we plan to observe the effect these applications have on each others performance when they are sharing a large Level-2 NUCA cache, which allows dynamic allocation of cache capacity. It should be noted here that in this study we focus only on multi-programmed application mix and do not consider any form of data sharing among applications. This study is a sensitivity study where cache capacity and access latencies to banks are varied to observe how application performance is impacted.

The rest of this abstract is organized as follows - we present our methodology in Section 2 and present a brief summary of results in Section 3. We finally conclude in Section 4 with pointers to future work.

II. METHODOLOGY

We modified the SimpleScalar [2] architectural simulator to model two important aspects of our study:

1) Modeling variable cache capacity
2) Modeling differential access latency

To model variable capacity, we allocated capacity to applications at the granularity of a cache way. We also incorporated in the cache access functions of the simulator the ability to reflect differential access latency to a particular cache bank(we aggregate a variable number of ways and call them banks). This latter modification implied that any new capacity allocated for the application could potentially have a different latency than the existing capacity. Thus for example, with a 32KB base capacity initially allocated to an application having access latency as 18 cycles, with these modifications to the simulator we were able to allocate another 16KB (and other arbitrary capacities) at 28 cycle (and other arbitrary latencies) latency. This helped us study the sensitivity of the application to both cache capacity and access latency. We also initially used CACTI 6.0 [6] to determine the access latencies for various cache capacities.

We used an inference based approach to model a CMP with shared L2 since we used multi-threaded benchmarks, and simulations were run as stand alone binaries without the operating system. Also, there was no sharing of data between the applications. We then ran each application independently (because SimpleScalar is a uni processor simulator) with proper scaling factors for cache capacity. To model cache sharing, we partitioned the cache statically and allocated appropriate capacity for that run. These set of benchmarks are single threaded and for this study we are only looking at the impact of cache size and access latency on an applications performance.

III. RESULTS

We ran a number of experiments for 11 benchmark applications from the SPEC suite to measure their sensitivity to capacity and latency. We chose these applications based on their high level-2 data cache miss rates and varied cache capacity first without altering access latency. That set of experiments gave us the baseline sensitivity of these benchmarks to capacity. We next varied the access latency also along with capacity and observed the performance by measuring IPC and cache-miss rates. We also experimented with the granularity of capacity allocation and varied the access latencies at which these capacities (with different granularity of allocation than previous experiments) were available to the application. This gave us an insight into how sensitive application performance is to granularity of additional capacity.

After compiling the results for these experiment runs, we observed that for these set of applications, there were marked differences in application performance when we varied capacity. For some applications, there was up to 200% improvement in IPC when cache capacity was varied from 32KB to 12MB.

For sensitivity studies pertaining to access latencies, we observed that most applications were insular to a large extent to latencies of access to newly allocated banks. The overall performance for the applications, as captured by the IPC, did indeed drop, but the reduction was not substantial. The reason why these applications were insensitive to access latency could best be explained by the fact that main memory accesses, being an order of magnitude larger, dominate performance and an aggressive Out-of-Order processor can seemingly hide/tolerate the marginal increase in cache access successfully.

IV. CONCLUSIONS AND FUTURE WORK

A. Conclusions

As mentioned above, the experiments show marginal impact on performance of multi-programmed mix of workloads due to variable access latency of cache banks. We however re-iterate that we did not account for sharing of data of any kind in this work and assumed that the each core independently executes its application in isolation.

B. Future Work

The next stage in our work is to determine the cache behavior of multi-threaded application which share data between multiple threads executing on different cores of the CMPs. We also intend to modify our methodology by using a full system simulator to capture the behavior of applications in presence of an Operating System. Our current methodology uses SimpleScalar, which is a syscall emulator, cycle-accurate simulator.

REFERENCES

[1] B. Beckmann, M. Marty, and D. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of MICRO-39*, December 2006.

[2] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[3] J. Chang and G. Sohi. Co-Operative Caching for Chip Multiprocessors. In *Proceedings of ISCA-33*, June 2006.

[4] Z. Chishti, M. Powell, and T. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *Proceedings of ISCA-32*, June 2005.

[5] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *Proceedings of ICS-19*, June 2005.

[6] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14, Washington, DC, USA, 2007. IEEE Computer Society.

[7] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, 1995.

[8] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proceedings of ISCA-32*, June 2005.

Reduced Packet Probing (RPP) Multirate Adaptation For Multihop Ad Hoc Wireless Networks

Jun Cheol Park
University of Utah
jcpark@cs.utah.edu

Sneha Kumar Kasera
University of Utah
kasera@cs.utah.edu

1. INTRODUCTION

Multirate support in the IEEE 802.11 PHY [1] allows a wireless network interface card to select different data rates[1] at the physical layer so that it can dynamically adapt to diverse channel conditions. Such multirate adaptation capability is critical in improving the performance of IEEE 802.11-based multihop ad hoc wireless networks. There is a fundamental trade-off between a selected data rate and its associated packet delivery ratio. In general, as the selected data rate decreases, the channel-error and the corresponding packet loss decreases. This is because at lower data rates a more robust modulation scheme is used. This modulation scheme has a longer medium occupancy time (at a reduced bandwidth) to resist interferences.

The key idea of multirate adaptation schemes is to find a data rate that can maximize the product of the selected data rate and its associated packet delivery ratio. However, developing an effective multirate adaptation scheme in multihop ad hoc wireless networks is challenging. This is because accurate assessment of the instantaneous channel conditions in multihop ad hoc wireless networks with multiple collision domain is more difficult than on a single IEEE 802.11 collision domain with multiple contending nodes in an infrastructure mode. The interaction between channel-error loss and collision in multihop ad hoc wireless networks causes an interesting problem, "packet loss anomaly" (that we call). In this anomaly, when a data rate decreases, the channel-error loss decreases whereas the collision loss increases.

To address the packet loss anomaly problem, in this extended abstract we propose an efficient multirate adaptation scheme, *Reduced Packet Probing* (RPP), that enables a sender node to effectively approximate channel-error loss in the presence of collisions without requiring any special information from the physical layer (e.g., SNR or physical capture) or incurring any significant overhead such as the exchange of RTS (Request To Send)/CTS (Clear To Send).

The rest of this abstract is organized as follows. In the next section, we motivate our work to demonstrate the limitations of the existing work. Section 3 describes the packet loss anomaly problem. We present our RPP multirate adaptation scheme in Section 4. Finally, we evaluate our preliminary implementation of RPP in Section 5.

[1]We use the term of *data rate* for "IEEE 802.11 PHY transmission rate".

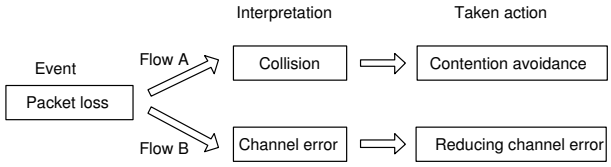


Figure 1: Packet loss Actions: Contention Avoidance or Channel-error reduction.

2. MOTIVATION

In recent years, a number of multirate adaptation schemes (ARF [2], AARF [3], SampleRate [4], HRC [5], CARA, [6], RRAA [7], etc) have been proposed. In these schemes, *a packet loss on a wireless link* is used to estimate channel conditions. A packet loss event is considered to be an indication of the data rate at which the packet is transmitted being too high for the given instantaneous channel condition, and used to downgrade the rate of transmission. However, in multihop ad hoc wireless networks, not all packet losses are due to poor channel conditions. Packets can also be lost due to collisions when packet transmissions overlap. The two different causes of packet loss in multihop ad hoc wireless networks are shown in Fig. 1. When a packet is lost due to collisions (Flow A) while channel conditions on a wireless link are good, a node must not downgrade its data rate. Rather, it must take contention avoidance actions to prevent future collisions. In contrast, when a packet is lost due to channel-error (Flow B), a data rate downgrade becomes necessary to cut down packet loss and prevent wastage of the channel capacity. We find that the existing multirate adaptation schemes do not effectively differentiate the causes of packet loss in multihop ad hoc wireless networks, and hence perform poorly.

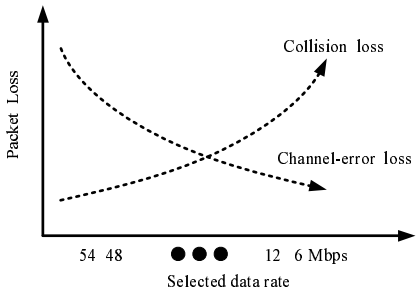


Figure 2: The problem of *packet loss anomaly* between channel-error loss and collision loss.

3. PACKET LOSS ANOMALY

Currently, wireless nodes have no effective way of distinguishing between packet loss due to channel errors from those due to collisions. So multirate algorithms reduce the data rate even when they observe loss due to collisions. Interestingly, the reduction in the data rate increases the frame transmission time and thus increases the medium occupancy time. This increase in the medium occupancy time actually increases the probability of collision loss.

The relationship between channel-error loss and collision loss as a function of a selected data rate is shown in Fig 2. When a data rate decreases, the channel-error loss decreases whereas the collision loss increases. This anomaly between channel-error and collision makes decision on suitable data rates in multihop ad hoc wireless networks more challenging.

4. RPP SCHEME

In order to address the packet loss anomaly problem in multirate adaptation schemes for multihop ad hoc wireless networks, we propose a new multirate adaptation scheme, *Reduced Packet Probing* (RPP), that effectively approximate channel-error loss in the presence of collisions.

In our RPP scheme, a sender node periodically invokes "a probing phase." In the probing phase, the sender node examines from the highest data rate to the lowest data rate to find the suitable data rates. While downgrading the data rate, it reduces the packet size to be transmitted so that the actual transmission time of the reduced packet at the lowered data rate is same as the one of the packet at the previously higher data rate. The rationale behind this approach is the observation that the collision loss mainly depends on the actual medium occupancy time regardless of the selected data rate. By keeping the same medium occupancy time for different data rates (i.e., keeping the same collision loss), any changes in the packet loss at the sender node can be inferred as changes in the channel-error loss. Thus, our approach enables a sender node to find suitable data rates without any help from its receiver node by performing an accurate analysis of channel-error loss in the presence of collisions.

5. EVALUATION

We have implemented our preliminary scheme in the Emulab wireless testbed [8]. We deploy the Roofnet [9] software on the Emulab wireless nodes. Roofnet implements the Dynamic Source Routing (DSR) [10] using the Click toolkit [11]. Each Emulab wireless node has Netgear WAG311 802.11a/b/g cards that are based on the Atheros chipset [12] (AR5212). We use our own modified version of the Madwifi-0.9.3 [13] IEEE 802.11 driver on Linux-2.6.20.6 to make it compatible with Click.

Our preliminary implementation shows a 20%∼50% improvement in throughput on a 3-hop ad hoc path compared to the existing multirate adaptation scheme, SampleRate [4].

6. REFERENCES

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, ANSI/IEEE Std., 1999.
[2] A. Kamerman and L. Monteban, "WaveLAN-II: a High-Performance Wireless LAN for the Unlicensed Band," in *Bell Lab Tech Journal*, 1997.
[3] M. Lacage, M. H. Manshaei, and T. Turletti, "IEEE 802.11 rate adaptation: A practical approach," in *ACM MSWiM*, 2004.
[4] J. C. Bicket, "Bit-rate Selection in Wireless Networks," Master Thesis, MIT, 2005.
[5] Ivaylo Haratcherev, et al., "Hybrid Rate Control for IEEE 802.11," in *ACM MobiWac*, 2004.
[6] Jongseok Kim, et al., "CARA: Collision-Aware Rate Adaptation for IEEE 802.11 WLANs," in *IEEE INFOCOM*, 2006.
[7] Starsky H.Y. Wong, et al., "Robust Rate Adaptation for 802.11 Wireless Networks," in *ACM MOBICOM*, 2006.
[8] "Emulab," http://www.emulab.net.
[9] John Bicket, et al., "Architecture and Evaluation of an Unplanned 802.11b Mesh Network," in *ACM MOBICOM*, 2005.
[10] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *ACM SIGCOMM*, 1996.
[11] "Click," http://www.read.cs.ucla.edu/click/.
[12] "Atheros communications," http://www.atheros.com/.
[13] "Madwifi: Multiband Atheros Driver for WiFi," http://madwifi.org/.

Salable, Reliable, Power-Efficient Communication for Hardware Transactional Memory

Seth Pugsley, Manu Awasthi, Niti Madan, Naveen Muralimanohar, Rajeev Balasubramonian
School of Computing, University of Utah

1 Introduction

Transactional Memory (TM) [3] is being viewed as a promising approach to simplify the task of parallel programming. In a TM system, critical sections are encapsulated within transactions and either the software or hardware provides the illusion that the transaction executes atomically and in isolation.Many recent papers have argued that the implementation of transactional semantics in hardware is feasible. Most of these studies have considered small-scale multiprocessor systems (fewer than 16 processors) and have shown that *hardware transactional memory (HTM)* imposes tolerable overheads in terms of performance, power, and area. However, it is expected that the number of cores on a chip will scale with Moore's Law. Further, transactional parallel programs will also be executed on multi-processors composed of many multi-core chips. If HTM is to be widely adopted for parallel programming, it is necessary that the implementation scale beyond hundreds of cores.

Implementations of transactional memory have been classified as either Lazy or Eager based on conflict detection and version control mechanisms they employ. This helps defines three classes of plausible implementations based on which combination of policies are used. In this work we focus on Lazy conflict detection and Lazy conflict management based policies, based on Stanford-TCC model [1].

In a recent paper, Chafi et al. [1] attempt to provide scalable parallel commits in a large-scale multiprocessor system based on lazy conflict detection can lazy version management. Numerous processors (possibly many multi-cores) are connected with a scalable grid network that allows message re-ordering with distributed shared-memory with a directory-based cache coherence protocol. The problem with allowing multiple parallel transaction commits is that a subset of these transactions may conflict with each other. The discovery of these conflicts mid-way through the commit process must be handled elegantly. As a solution, Chafi et al. propose the Scalable-TCC algorithm that is invoked by a transaction when it is ready to commit. In summary, the algorithm first employs a centralized agent to impart an ordering on the transactions. Two transactions can proceed with some of the steps of the commit algorithm in parallel as long as their read-set and write-set directories are distinct. The commits are serialized based on the tokens the transaction recieve from the centralized vendor, which could be a bottleneck with tens or hundreds of cores. Also, the number of messages exchanged scales linearly with number of nodes, which increases the interconnect bandwidth requirement and the associated power overheads.

We propose novel algorithms that significantly reduce delay, are free of deadlocks/livelocks, do not employ a centralized agent, do not produce new starvation scenarios, and significantly reduce the number of network messages (and associated power), relative to the Scalable-TCC implementation. These algorithms are more scalable because the message requirement is not a function of the number of nodes in the system. We design a basic simple algorithm that requires very few network messages and then propose a few performance optimizations. Reduction in number of messages helps reduce the power footprint of the system. Similarities of proposed algorithms to existing token coherence protocols provide proof of correctness of the proposals.

2 Scalable Commit Algorithms

We propose commit algorithms that avoid a centralized resource and have message requirements that do not directly scale up with the number of nodes. In doing this, we preserve most of the Scalable-TCC architecture except the algorithm that determines when a transaction can make its writes permanent. We begin with a conceptually simple algorithm and then add a modest amount of complexity to accelerate its execution time. Similar to the work by Chafi et al. [1], we assume a distributed shared memory system with a directory-based cache coherence protocol. To keep the discussion simple, we assume that the number of processors equals the number of directories.

2.1 Sequential Commit

We introduce an *"Occupied"* bit in each directory that indicates that a transaction dealing with this directory is in the middle of its commit phase. In this first algorithm, a transaction sequentially proceeds to "occupy" every directory in its read- and write-set in ascending numerical order (Step 1) (the transaction must wait for an acknowledgment from a directory before proceeding to occupy the next directory). A directory is not allowed to be occupied by multiple transactions, so another transaction that wishes to access one of the above directories will have to wait for the first transaction to commit. After Step 1, the first transaction knows it will no longer be forced to abort by another transaction and it proceeds with sending information about its write-set to the corresponding directories (Step 2); these cache lines will be marked as Owned in the directory and invalidations are sent to other sharers of these lines. After the directory receives all ACKs for its invalidations, it re-sets its Occupied bit. As part of Step 2, the transaction

also sends Occupancy Release messages to directories in its read-set.

If a transaction attempts to occupy a directory that is already occupied, the request is buffered at the directory. If the buffer is full, a NACK is sent back and the transaction is forced to re-try its request. In our experiments, we observe that re-tries are uncommon for reasonable buffer sizes. The buffered request will eventually be handled when the earlier transaction commits. There is no possibility of a deadlock because transactions occupy directories in numerically ascending order and there can be no cycle of dependences. Assume transaction A is waiting for transaction B at directory i. Since transaction B has already occupied directory i, it can only stall when attempting to occupy directory j, where $j > i$. Thus, a stalled transaction can only be waiting for a transaction that is stalled at a higher numbered directory, eliminating the possibility for a cycle of resource dependences. This algorithm imposes an ordering on conflicting transactions without the use of a centralized agent: the transaction to first occupy the smallest-numbered directory that is in the read/write-sets of both transactions, will end up committing first.

2.2 Optimizations to the Basic Algorithm

2.2.1 SEQ-PRO

The simple SEQ algorithm does not make a distinction between directories in the read and write sets. Each directory must be occupied sequentially, with no commit parallelism at a given directory. However, if two transactions only have reads to a given directory, there is no possibility of a conflict at this directory. These two transactions can simultaneously occupy the directory and safely proceed with the rest of their commit process. In other words, we can allow multiple transactions to simultaneously occupy a directory as long as none of these transactions write to this directory. Note that this optimization is also deadlock-free as all transactions proceed to occupy directories in a sequential order.

2.2.2 SEQ-TS

While the SEQ algorithm helps reduce the total number of required messages, the delay may be higher than that of the scalable-TCC algorithm because each of the directories must be occupied in sequential order. To remove the dependence on this sequential process, we propose the following timestamp-based commit algorithm (SEQ-TS). Transactions attempt to occupy directories in parallel and a transaction is allowed to steal a directory from another transaction if it is deemed to have higher priority. Priority is determined by age: every occupancy request is accompanied by the timestamp of when the transaction began its commit process. For such an algorithm to work, we need every core to participate in a distributed clock algorithm [2] so they have an approximate notion of relative time

3 Results and Conclusion

In our evaluations, we focus on synthetically generated commit requests. The N nodes in the system (each node has a processor and directory) are organized as a grid. We agumented the simulator to model cache coherence and network delays. We believe that synthetic workloads provide more insight and a more direct comparison between the commit algorithms. The synthetic workloads allow us to easily change parameters (locality, transaction size, commit set size, number of nodes) and understand the factors influencing behavior. This is especially relevant because most available benchmark applications have not been designed or tuned to run on 64-256 nodes and the landscape of transactional workloads of the future is not clear. For the mid-locality case and 64 nodes, SEQ, SEQ-PRO, and SEQ-TS reduce the commit delay by 46%, 70%, and 78%, respectively. If the transaction length is only 200 cycles, Scalable-TCC requires as many as 245 cycles for commit (a huge overhead). Hence, SEQ-TS's 78% improvement in commit delay translates to an impressive 43% improvement in overall performance. As transaction lengths are increased, the commit delays contribute less to overall performance (Amdahl's Law and fewer simultaneously competing transactions). Also, irrespective of transaction length, the SEQ algorithm reduces the message requirements by 48X for the 256-node cases.

References

[1] H. Chafi , J. Casper, B. Carlstrom, A. McDonald, C. Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A Scalable Non-Blocking Approach to Transactional Memory. In *Proceedings of HPCA-13*, February 2007.

[2] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21:558–565, 1978.

[3] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan & Claypool, 2006.

Scaling Formal Methods Towards Hierarchical Protocols in Shared Memory Processors*

Xiaofang Chen[1], Ganesh Gopalakrishnan[1], Ching-Tsun Chou[2], Steven M. German[3] and Michael Delisi[1]
[1]School of Computing, University of Utah
[2]Intel Corporation, Santa Clara
[3]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598

Introduction

Due to the high performance requirements and manufacturing realities, cache coherence protocols used in future shared memory processors will be extremely complex, and they will be *hierarchically* organized. To verify such complex protocols, a common approach will be verify either the high (specification) level, or the low (RTL implementation) level.

Protocol specifications are usually verified by modeling them in guard/action languages such as Murphi or TLA+, and by model checking them using explicit state enumeration tools. With hierarchical cache coherence protocols, the verification has to handle the complexity of several coherence protocols running concurrently.

Moreover, it is insufficient to merely verify high level models of protocols. The semantic gap between specifications and their hardware implementations is non-trivial. For example, one atomic step in the specification is typically realized through a *transaction* in the implementation, which is a multi-cycle activity consisting of one or more concurrent steps in each clock cycle. Also, a sequence of high level atomic steps may actually be realized through multiple transactions whose time intervals may overlap. Therefore, techniques to bridge the gap between protocol specifications and implementations are essential.

In our work, we are motivated to solve both the problems. To verify a hierarchical protocol specification, we propose a compositional approach by decomposing it into a set of abstract protocols. By verifying the abstract protocols, the original hierarchical protocol is guaranteed to be correct with respect to its safety properties. On the other hand, to bridge the gap between protocol specifications and implementations, we propose a formal theory and a methodology using transaction based modeling and verification. We also propose a compositional approach to reducing the verification complexity, and a tool support to mechanize the process.

A key feature of our approach is that it requires only existing model checking tools for supporting the assume guarantee method at the high level, and requires only existing HDL model checking frameworks for supporting the assume guarantee approach at the implementation level.

*This work was supported in part by SRC Contract 2005-TJ-1318.

Verifying Hierarchical Protocols in the High Level

For the verification of hierarchical cache coherence protocols, currently there is no publicly available hierarchical protocols with reasonable complexity. So we first develop three 2-level coherence protocols for multicore systems[1].

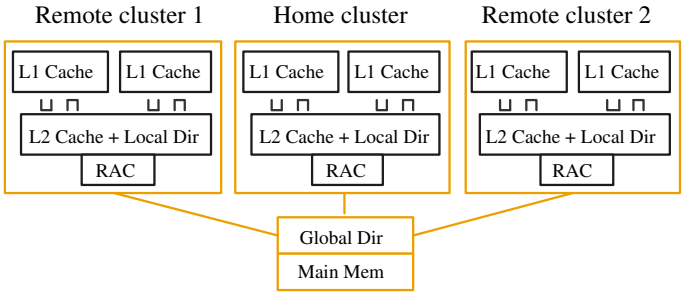


Figure 1: A 2-level multicore coherence protocol.

Figure 1 shows the first protocol. It employs directory based MESI protocols in both levels: the intra-cluster and the inter-cluster levels. Three NUMA clusters are modeled for one address, one home cluster and two remote clusters of the same design. The intra-cluster protocol is used to maintain coherence inside a cluster, and the inter-cluster is used among the clusters. Other features include: (i) explicit writeback and silent drop are modeled, (ii) messages can arrive out of order, and (ii) the L1 and L2 caches in a cluster are modeled as inclusive. The protocol modeled in Murphi has about 2500 LOC, and it has similar complexity with those used in practice. The other two protocols are similar, except that the second one uses the MSI protocols with the non-inclusive cache hierarchy, and the third employs snooping protocols inside clusters.

Given a hierarchical protocol, we first decompose it into a set of abstract protocols, as shown in Figure 2. Each abstract protocol (`Abs #i`) is obtained by projecting out different set of global variables, and correspondingly overapproximating the protocol transitions. We then verify each abstract protocol individually. If a genuine bug is found, we fix it in the original protocol and iterate the process. Otherwise when a spurious bug is found, we fix it by constraining

[1]Slides URL: http://www.cs.utah.edu/~xiachen/socPoster.ppt

the overly approximated rule; at the same time, a new verification obligation is added to an abstract protocol to ensure the constraining is sound. The process is iterated until all the `Abs #i`'s are correct [3]. Figure 3 shows the abstract protocols for the protocol in Figure 1 (in fact we have four abstract protocols but two are identical due to the symmetry).

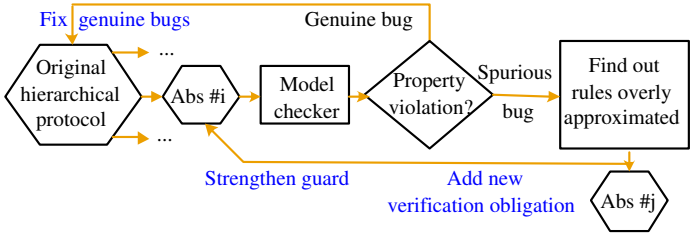


Figure 2: The workflow of the high level protocol verification approach.

Moreover, for the non-inclusive and the snooping multicore protocols, we propose to use history variables in an assume guarantee manner, together with the compositional approach described above. For all the three protocols, the experiments show that our approach can reduce more than 95% of the explicit state space of the original protocol [2].

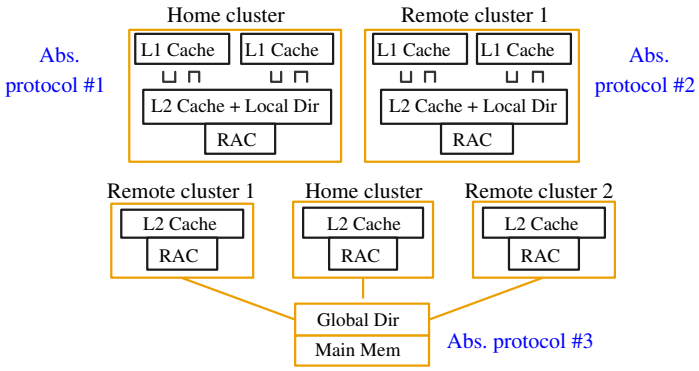


Figure 3: The abstract protocols of Figure 1.

Verifying Protocol Implementations Meet Specifications

To check the consistency between a protocol specification and an RTL implementation, we propose to use transactions to relate one step in the specification, to multiple step executions in the implementation. Now we define the refinement as follows. For every concurrent execution E_I of the implementation, there exists an interleaving execution E_H of the specification, such that every state s_I of E_I corresponds to a state s_H of E_H. Moreover, for every *joint* variable v in both models, its value must be equivalent at s_I and s_H, when v is inactive at s_I. By *inactive*, we mean that for all the transactions that can write to v, they have either finished executing or have not started at s_I.

With the above definition [1], we check refinement as follows. We employ Murphi to model the protocol specifications, and *Hardware Murphi* to model implementations. Hardware Murphi was proposed by German and Janssen from IBM, and they also implemented a translator called *Muv*, from Hardware Murphi into synthesizable VHDL. We extended Muv including the language so that it is not just a translator but also can generate assertions for refinement check. Now given a specification in Murphi, an RTL implementation in Hardware Murphi, and the joint variables in both models, Muv can automatically generate a VHDL model including all the assertions for refinement check. As said earlier, any existing verification tool for VHDL or Verilog can support this methodology. Figure 4 shows this process.

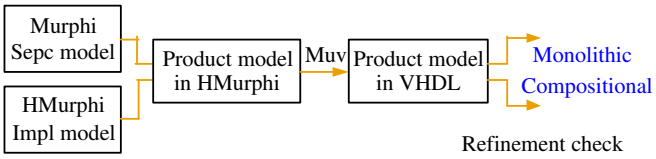


Figure 4: The workflow of refinement check.

Moreover, we propose a modular refinement verification approach by developing abstraction and assume guarantee reasoning principles that allow an implementation transaction realizing a single specification step to be situated in sufficiently general environments. This is partly done by constructing abstract models for each transaction, where reads of a global state variable can be selectively changed into reads of a free input variable (or an input signal). Thus, any property which holds in the abstract model must also hold in the original model.

We have successfully applied the refinement check to a 2-stage pipelined stack example and a driving coherence protocol with realistic features. For the coherence protocol, three subtle bugs were found in the RTL implementation. These bugs are easy to miss by writing assertions alone, while our approach can automatically generate such assertions. Also, we show that when the datapath of a cache line is 10-bit, our modular approach can reduce the runtime of the refinement check from over a day to about 30 minutes, using SixthSense a state-of-art verification tool from IBM.

In conclusion, we propose practical formal verification techniques for one of the most critical subsystems of future multicore processors - coherece protocol engines. For future work, we plan to apply our methods to other subsystems of multicore processors, and also to verify the modification of existing protocols to incorporate power management, built-in self test, and post-silicon verification support.

1. REFERENCES

[1] X. Chen, S. M. German, and G. Gopalakrishnan. Transaction based modeling and verification of hardware protocols. In *FMCAD*, 2007.

[2] X. Chen, Y. Yang, M. Delisi, G. Gopalakrishnan, and C.-T. Chou. Hierarchical cache coherence protocol verification one level at a time through assume guarantee. In *HLDVT*, 2007.

[3] X. Chen, Y. Yang, G. Gopalakrishnan, and C.-T. Chou. Reducing verification complexity of a multicore coherence protocol using assume/guarantee. In *FMCAD*, 2006.

Stateful Runtime Model Checking of Multithreaded C Programs *

Yu Yang Xiaofang Chen Ganesh Gopalakrishnan Robert M. Kirby
School of Computing, University of Utah, Salt Lake City UT 84108 U.S.A.
{yuyang,xiachen,ganesh,kirby}@cs.utah.edu

ABSTRACT

In applying runtime model checking methods to realistic multithreaded programs, we find that stateless search methods are ineffective in practice, even with dynamic partial order reduction (DPOR [1]) enabled. This happens especially when critical sections are nested inside loops. Existing DPOR methods can fail on even short examples containing this pattern. To solve this problem, this paper makes two related contributions. The first contribution is a novel and conservative light-weight method for recording states that helps avoid redundant searches. The second contribution is a stateful dynamic partial order reduction algorithm(SDPOR) and its efficient implementation. Our stateful runtime model checking approach combines the light-weight state recording method with SDPOR, and strikes a good balance between state recording overheads, on one hand, and the elimination of redundant searches, on the other hand. Our experiments confirm the effectiveness of our approach on several multithreaded benchmarks in C, including some non-trivial practical programs.

1. INTRODUCTION

Despite all the advances in developing new concurrency abstractions, explicit thread programming using thread libraries remains one of the most practical ways of realizing concurrent programs that take advantage of multiple cores, and help improve the overall system responsiveness and performance. Many high level concurrency abstractions (*e.g.*, software transaction memories) also require the use of threads for implementation. Unfortunately, programmers are all too familiar with the risks of thread programs [2], including the ease with which deadlocks and local assertion violations can be introduced. In this paper, we focus on the efficient checking of a given multithreaded program for safety violations over *all possible interleavings*.

Runtime model checking is a promising method for bug detection. As model building, extraction, and model maintenance are expensive [3, 4], we believe in the importance of developing efficient runtime checking methods. However, even with specific inputs, the number of interleavings due to internal scheduling decisions can grow astronomically. Stateless search methods (e.g., Verisoft [3]) may help mitigate the problems, however often causing state explosion due to redundant searches.

Much of the state explosion in this context can be attributed to not maintaining visited states. For example, for the program shown in Figure 1, one needs to examine $200!/(100!100!) > 2^{100}$ different interleavings even with stateless search and dynamic partial order reduction. In multithreaded programs, a common pattern is that the critical sections are nested inside loops. This makes the problem worse for stateless search. Unfortunately, it is often expensive or impossible to maintain visited states.

It is clear that different executions of a program may share common states. For example, Figure 2 shows two executions (over one iteration of each loop) that lead to the same state. So the question really is: how to cheaply, and soundly, detect revisitations?

It is not that straightforward to capture the states of a multithreaded C program at runtime. For example, if we take CMC's approach [5], we need to capture the state of the kernel space plus the user space. If we want to follow Java PathFinder's [6] approach, we will have to build a virtual machine that can handle C programs. Both approaches are very involved.

1.1 Our Contributions

- We propose a novel local-state delta maintenance scheme. Instead of capturing the entire state, we conservatively assign a unique id to each local state of a thread such that there is a many-to-one function from IDs to local states. When affordable, we capture the changes δ between consecutive local states (otherwise we pretend that the changes are arbitrary, denoted by $\delta_\perp$). We then use δ, the state of the global objects, and the ID of the *preceding* local state, to test whether the same local state has been revisited. The points where the δ capture is deemed affordable is currently identified by hand, but we propose program analysis to automate this.

*Supported in part by NSF award CNS00509379, Microsoft HPC Institute Program, and SRC Contract 2005-TJ-1318.

```
const int MAX = 100;
int d = 0;
mutex x;

  thread 1:                                thread 2:
L0: for (int i = 0; i < MAX; i++){   M0:   for (int j = 0; j < MAX; j++){
L1:    lock(x);                      M1:      lock(x);
L2:    d=d+5;                        M2:      d += j;
       assert(d % 5 != 4);                    assert(d % 5 != 2);
L3:    unlock(x);                    M3:      unlock(x);
L4:  }                               M4:   }
```

Figure 1: A simple example

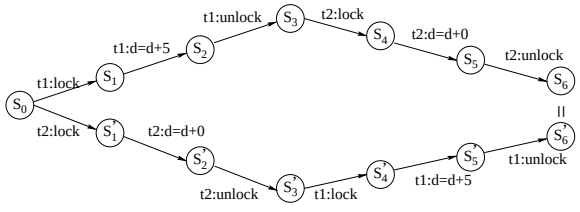


Figure 2: Two alternative interleavings of the program shown in Figure 1 reach the same state.

- We present a stateful dynamic partial order reduction (SDPOR), which combines the stateful search with dynamic partial order reduction. In such an implementation, an obvious soundness bug is to not update the backtracking set along a new path that revisiting a state. This is solved through a novel transition dependency graph.

- We have implemented SDPOR within our runtime model checker `Inspect` [7], and evaluated our approach on a set of multithreaded C benchmarks. The experiments show that with SDPOR, `Inspect` has noticeable improvements in search efficiency.

Our poster is available at
`http://www.cs.utah.edu/~yuyang/poster08.ppt`.

2. IMPLEMENTATION

We implemented a stateful runtime model checker based on our stateless runtime model checker `Inspect` [7, 8] . `Inspect` is designed in a client/server style. The server side is a centralized monitor which controls the execution of the program under test. At the client side, first the program under test is instrumented with code that is used to intercept the visible operations. Then the instrumented code is linked with a stub library, which include stubs to intercept thread library routines, into a binary executable. Thereafter, the monitor starts systematically explore alternative interleavings of the program under specific inputs by concretely executing the program undertest.

The intrumentation part can be done automatically. In [8], we describe how automated instrumentation is done for stateless runtime checking. Within the context of stateful runtime checking, the additional code that we need to instrument in the program under test for runtime montoring is: (i) A flag to tell the monitor whether the sequence of invisible operations between two consecutive visible operations are global objects dependent or not. We can compute this automatically with static analysis on the data dependency of invisible operations. (ii) The changes of local states in a global objects indepedent local transition. This can also be automated with a conservative heuristic on the changed that we can capture with respect to the local states of threads.

3. REFERENCES

[1] Flanagan, C., Godefroid, P.: Dynamic Partial-order Reduction for Model Checking Software. In Palsberg, J., Abadi, M., eds.: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM (2005) 110–121

[2] Lee, E.A.: The problem with threads. Volume 39., Los Alamitos, CA, USA, IEEE Computer Society Press (2006) 33–42

[3] Godefroid, P.: Model Checking for Programming Languages using Verisoft. In: POPL. (1997) 174–186

[4] Musuvathi, M., Qadeer, S.: Iterative context bounding for systematic testing of multithreaded programs. In Ferrante, J., McKinley, K.S., eds.: PLDI, ACM (2007) 446–455

[5] Musuvathi, M., Park, D.Y.W., Chou, A., Engler, D.R., Dill, D.L.: CMC: A Pragmatic Approach to Model Checking Real Code. In: OSDI. (2002)

[6] Visser, W., Havelund, K., Brat, G.P., Park, S.: Model checking programs. In: ASE. (2000) 3–12

[7] http://www.cs.utah.edu/~yuyang/inspect

[8] http://www.cs.utah.edu/~yuyang/inspect/inspect-intro.pdf

Towards A Virtual Teaching Assistant to Answer Questions Asked by Students in Introductory Computer Science

Cecily Heiner
School of Computing, University of Utah
3190 MEB, 50 S. Central Campus Drive, Salt Lake City, Utah 84112 USA
cecily@cs.utah.edu

ABSTRACT

Introductory Computer Science students ask many questions as they work on assignments, but finding answers independently can be an arduous process because the student must articulate the question, search for an answer that may or may not be in the course material, and determine how to apply the answer to the assigned exercise. This paper proposes that information from student source code, compiler output, and educational context can improve partially automated student question answering above a non-trivial baseline consisting of the natural language from the question that the student asks. The paper describes the system architecture and the class-specific inputs necessary to deploy such a system in a Computer Science 1 class. I hypothesize that many student generated questions in Computer Science 1 can be automatically answered.

Keywords

Educational technology, question answering, introductory Computer Science, information retrieval for novices

1. INTRODUCTION

Because novices are notoriously inept at articulating their reasoning, many intelligent tutoring systems use one of two approaches to dealing with the questions that students ask. Several tutoring systems have completely bypassed the problem of poorly articulated student language by restraining the size of the information space that the system covers (e.g. single physics problem[1], a sentence to be read aloud[2], or algebra symbolization[3]), dividing each problem into a set of skills, and creating interventions or help messages for each skill. Alternatively, some systems have focused specifically on helping students to articulate their reasoning because generating natural language may help students develop deep learning. However, these systems are generally trying to elicit specific pieces of natural language from the student instead of providing answers to student questions. Research on producing automated responses to student generated questions is remarkably sparse throughout the literature.

Although research on automated responses to student generated questions is rare, there are a few relevant papers. The most relevant paper reports on a wizard-of-oz study with the Cognitive Tutor to answer student questions; a short paragraph in that paper describes a preliminary pilot study that provided automated video clips as answers to questions, similar to a synthetic interview[4]. Another paper suggests that the majority of questions asked by students about a programming assignment are often clustered around one or two topics[5]. Finally, the How May I Help You system[6] semi-automatically classified and resolved help requests from users in a call system. A semi-autonomous model is particularly well-suited for a question answering system because the majority of the questions fall into a few major categories that can be easily automated, and the small number of questions for which an automated response is not practical can be handled by humans.

One group of students that may be especially well suited to using a semi-automated question answering tool is found in introductory Computer Science. Students in introductory Computer Science classes have had no prior exposure to programming, so they have many questions that are poorly articulated. Students depend on teaching assistants to provide help and feedback. Unfortunately, teaching assistants for introductory Computer Science classes are a scarce and expensive resource, and the demand for their attention often exceeds their supply of time. These demands come from students who are in the lab as well as students working remotely who send questions via e-mail. The questions that students submit via e-mail often concern a problem that occurs when compiling or running a program, and students usually neglect to provide sufficient context such as their source code and compiler errors. Furthermore, regardless of whether students ask questions in the lab or remotely, it is difficult to track what the student's question was and whether or not it has been answered.

2. A VIRTUAL TEACHING ASSISTANT FOR INTRODUCTORY COMPUTER SCIENCE

To address the problems outlined in the introduction, I propose an intelligent question answering system called a Virtual Teaching Assistant (VTA). This VTA will mediate the question answering process between the students and the teaching staff. Interaction with the VTA will work as follows. The student will type his or her question into the student software client of the VTA system. Typing the question will force students to give some thought about the question they want to ask, and hopefully facilitate deeper learning. Questions and student source code submitted to the VTA system will appear in a teaching staff's software client of the VTA system, ordered by time of submission. A member of the teaching staff will answer the question with a text or URL response. Then, the VTA system will log the answer to a database along with the question. The answer will be displayed in the interface of the student's software client and/or e-mailed to the student.

When possible, the VTA system will short-circuit the process by providing automated answers to common questions that similar to other questions already in the system. If an automated answer is provided, the student will be prompted to indicate if the answer

was adequate or if they still need help from a human TA. For example, consider the following three questions:

- How do I animate the smoke? (Q1)
- What does this compiler error mean? (Q2)
- How do I make the smoke move? (Q3)

When the system receives Q1, it has not seen any other questions, so it does not have a matching response, and it will pass Q1 to a human who will type in a response A1. When the system receives Q2, the system will compare Q2 to Q1 using a vector similarity algorithm (e.g. cosine similarity), determine that they are not similar, and pass Q2 to a human who will type in a response A2. When the system receives Q3, the system will compare Q3 to Q1, determine that there is a potential match and try to provide an automated answer A1 without human intervention. The student who asked Q3 will then indicate if A1 was a satisfactory answer.

3. A BRIEF OUTLINE OF THE EXPERIMENT

The thesis of this work is that information from student source code, compiler output, and educational data mining can improve partially automated student question answering above a non-trivial baseline consisting of the natural language in the questions that students ask. Happily, this idea can be explored while allowing all students to use a fully functional version of the VTA system by running the experiment as a post-hoc ablation study offline. To run the experiment, the VTA system will consider each question in the order in which it was entered into the system. The system will then decide if it matches a previously entered question or if it represents a new question category. This classification is called the evaluation classification. The evaluation classification will be compared to the original classification of the question logged in the system when the student asked the question. This original classification was either made by a human TA or the complete VTA system; if the original classification was made by the complete VTA system, then it was verified by the student asking the question. If the evaluation classification and the original classification match, then the system will receive credit for classifying that question correctly. If they do not match, the system will receive no credit for classifying that question. The score for each system will be the percentage of questions classified correctly.

The experiment will use the VTA system with natural language as the only input source as a non-trivial baseline. The experiment will then consider four additional conditions. In the first condition, just the natural language input and the abstract form of student code will be used. In the second condition, just the natural language and the compiler output will be used. In the third condition, just the natural language and the additional information from the educational context will be used. A fourth condition will incorporate all of the input. I hypothesize that the baseline condition of the natural language alone will be able to classify approximately half of the questions, but that the additional information will allow at least an additional fourth of the questions to be correctly classified.

4. CONCLUSIONS AND FUTURE WORK

This paper introduces a Virtual Teaching Assistant for Introductory Computer Science classes. The Virtual Teaching Assistant will provide automated answers to common questions and a tracking mechanism for all questions. This tracking mechanism will allow the teaching staff to monitor the kinds of difficulties that students are having and tailor instruction appropriately. The paper describes an architecture for a generic automated question answering system and then describes the domain-specific modules and class-specific inputs necessary for the system to function for an Introductory Computer Science classes. The paper then describes an ablation experiment to determine if the additional information provided by the student input, compiler output, and educational context can improve question classification above the non-trivial baseline consisting of student-generated natural language questions alone.

The modular architecture of the VTA system described in the paper will provide a foundation for extensive additional research and instructional functionality beyond what is described in this paper. For example, a metacognitive component could be developed using much of the existing framework to anticipate when students might ask a question and automatically launch the VTA system. An instructional evaluation component could be developed to compare different answers or interventions for a particular kind of question. Finally, future experiments may elucidate how automated answers affect programming efficiency, student retention, and long-term learning gains.

5. ACKNOWLEDGEMENTS

Thanks to my advisor Joe Zachary, classmates in CS 7010, participants in the 2006 SIGCSE Doctoral Consortium, and Chad Lane for feedback on early versions of this work. Additional thanks to many anonymous reviewers and other conference participants who have contributed comments on various versions of this work.

6. REFERENCES

1. Gertner, A.S. and K. VanLehn. *Andes: A Coached Problem Solving Environment for Physics*. in *Intelligent Tutoring Systems*. 2000. Montreal, Canada: Springer.
2. Mostow, J. and G. Aist, *Evaluating tutors that listen: An overview of Project LISTEN*, in *Smart Machines in Education*, K. Forbus and P. Feltovich, Editors. 2001, MIT/AAAI Press: Menlo Park, CA. p. 169-234.
3. Heffernan, N., *Intelligent Tutoring Systems have Forgotten the Tutor: Adding a Cognitive Model of Human Tutors*, in *School of Computer Science*. 2001, Carnegie Mellon University: Pittsburgh, Pennsylvania. p. 187.
4. Anthony, L., A.T. Corbett, A.Z. Wagner, S.M. Stevens, and K.R. Koedinger. *Student Question-Asking Patterns in an Intelligent Algebra Tutor*. in *Intelligent Tutoring Systems*. 2004. Maceio, Brazil: Springer.
5. Robins, A., P. Haden, and S. Garner. *Problem Distributions in a CS1 Course*. in *Australian Computing Education Conference*. 2005. Hobart, Tasmania: Conferences in Research in Practice in Information Technology
6. Walker, M., I. Langkilde, J. Wright, A. Gorin, D. Litman, and *Learning to Predict Problematic Situations in a Spoken Dialogue System: Experiments with How May I Help You?* in *North American Chapter of the Association of Computational Linguistics*. 2000. Seattle, WA: Morgan Kaufmann.

Versions of this work have been published in the Young Researcher's track of Artificial Intelligence in Education, the Doctoral Consortium of SIGCSE, and the Colorado Celebration of Women in Computing.

Utilizing World Knowledge for Coreference Resolution

Nathan Gilbert ngilbert@cs.utah.edu
Ellen Riloff riloff@cs.utah.edu

The work described in this document concerns the problem of coreference resolution, which is inherent in many Natural Language Processing tasks. Current methods for resolving coreferent entities in text rely on "knowledge poor" features, i.e. lexical or syntactic information. This work is an attempt to gain new advances in coreference resolution by exploring richer knowledge features, such as information from the web or other "real-world" sources.

1. THE PROBLEM

Often in Natural Language Processing (NLP) tasks we are presented with the problem of determining if a given phrase refers to another phrase that has previously occurred in the text. These "referring" or "pointing-back" phrases are called anaphora and what they refer to are labeled antecedents. If an anaphor and its antecedent refer to the same entity in the "real-world" then they are considered coreferential.

For an example consider the following sentence:

David Beckham is *the LA Galaxy midfielder.*

The phrase *the LA Galaxy midfielder* is an anaphor with antecedent *David Beckham*, since both refer to the same entity in the "real-world" they are considered coreferential. The act of determining that both phrases are coreferential is known as *coreference resolution.*

Improving automatic coreference resolution would have a major impact on other areas of NLP research, such as document summarization and information extraction. For instance, knowing that two phrases are coreferent can help decrease redundancy in summarization tasks and potentially increase the effectiveness of extraction patterns for information extraction.

2. PREVIOUS WORK

The first Machine Learning approach to coreference resolution that achieved results surpassing heuristic (rule-based) systems was described in [3]. Soon et al. developed 11 features they hoped could adequately capture coreference and tested their system on the Message Understanding Conference (MUC) 6 and 7 data sets. For every possibly coreferent noun phrase in a document (denoted "markables" by the authors), a feature vector would be created for it and each markable before it in the document. After all feature vectors were created, a decision tree classifier was used to identify coreferent noun phrases.

The Soon et al. system scored an F-measure of 63% on MUC 6 and achieved an F-measure of 61% on MUC 7. While these scores are in the middle of the pack for systems that actually participated in the MUC conferences, it was a great achievement for learning-based coreference systems.

Ng & Cardie [1] took the Soon et al. work further by greatly enriching the feature set, increasing their numbers from 11 to the mid-fifties. This time, not only did this system perform better than any other learning-based coreference approach, it also did better than any of the participants in the original MUC conferences, achieving a F-measure score of 63.4 on MUC 6 and 61.6 on MUC 7. There have several other novel systems introduced in the past several years, but these are the two by which others are judged.

3. RECONCILE

At the University of Utah we have developed our own fully automatic coreference resolution system, named Reconcile, which posts state of the art numbers on both MUC data sets. The approach taken in this project is similar to that of Cardie & Ng [1] described above. We have developed a feature set heavily influence by Cardie & Ng but differ in the modularity and range experiments that are possible with Reconcile.

Reconcile is a supervised learning coreference resolution system that processes texts from the ground up. In short, it extracts noun phrases on its own and then trains and tests a model based off of these extractions. Annotated data is required for training a model. Various classifiers are available for use in Reconcile, all results in this document were obtained by a pruned decision tree.

Current results on the MUC6 and 7 data sets are in the table below, for this type of system are state of the art.

	Recall	Precision	F-measure
MUC6	67.7%	68.0%	68.0%
MUC7	57.0%	72.6%	64.4%

The feature set of Reconcile and other similar systems mainly consists of "knowledge-poor" features. Essentially, these methods rely on information that has low semantic value. Examples of this type of feature would be string matching or whether or not the antecedent and anaphor are both in quotes. Surprisingly, these simple features prove capable of achieving state of the art scores. Recent research has indicated that knowledge-poor features may have hit a performance upper-bound [2]. If this is the case, the time is right to start exploring more knowledge rich information sources.

4. INCORPORATING WORLD KNOWLEDGE

The goal of this research is to improve upon the automatic coreference resolution task by introducing more knowledge rich features into the learning framework. Currently, the majority of this knowledge is coming from the Web in the form of co-occurrence statistics between anaphor and antecedent.

Our first hypothesis is that coreference resolution can be improved by analyzing the results of web search queries containing an anaphor and a possible antecedent. For instance, given the anaphor "Bloomberg" and possible antecedents "table" and "NYC mayor" one would expect to find that there are more results returned for "Bloomberg AND NYC mayor" than the alternative. The hardest part of using this knowledge if determining the best way to represent it to the classifier. Experiments have shown that raw hit counts are too noisy to be incorporated effectively.

Several methods for representing this knowledge have been attempted thus far. As previously mentioned, adding raw hit counts as a feature does not induce any significant change in system performance. The pointwise mutual information between anaphor and antecedent has proven more successful in representing hit counts. Pointwise mutual information (PMI) is defined

$$PMI(x, y) = \log \frac{p(x, y)}{p(x)p(y)}$$

where in this new setting the probabilities $p(x, y)$, $p(x)$ and $p(y)$ are replaced by hit counts of the antecedent and anaphor. This feature representation has so far improved recall 2-3 points above the previous baselines.

Another representation that has proven useful is conditional probability $P(A|B)$, where we want to determine the probability of seeing anaphor A given that we have seen antecedent B. This is approximated by $P(A|B) = \frac{count(A,B)}{count(B)}$.

The previous features described are still fairly simple, are achieving only modest gains above current systems. Our current work is focusing on inducing a partitioning over an anaphor's possible antecedents. For instance, by creating a feature that contains the difference between anaphor hit count and antecedent hit, a 1-2 point improvement can be gained for both recall and precision. We hypothesis that this feature is crudely modeling each possible antecedents "popularity" compared to the anaphors.

On going work on this hypothesis is leading us to induce clusterings for an anaphor's set of possible antecedents and make features based off various characteristics of these clusters.

5. FUTURE WORK

The full vision of this work goes beyond using hit counts and simple web queries to gain access to "world-knowledge" for coreference resolution. There are many avenues for obtaining world knowledge from the web, hits counts being a small fraction of them. Other areas of interesting involve structured data such as lists and records that are found on the web. Trusted websites such as dictionaries and encyclopedias also provide a manner in which to mine the web for meaning.

6. CONCLUSION

Coreference resolution continues to be an active area of NLP research. Any improvements to resolution performance are valuable because of the importance of coreference knowledge in other areas of NLP research. We have developed a state of the art coreference resolution system and have been experimenting with incorporating real world knowledge with growing success. This knowledge is currently coming in the form of antecedent and anaphora hit counts from popular search engines, but will continue to more fine grained knowledge sources as the research proceeds.

7. REFERENCES

[1] Ng, V., and Cardie, C. Improving machine learning approaches to coreference resolution. *Proceedings of the 40th Annual Meeting of the ACL* (2002).

[2] Palomar, M. An algorithm for anaphora resolution in spanish texts. *Computational Linguistics* (2001).

[3] Soon, W. M., Ng, H. T., and Lim, D. C. Y. A machine learning approach to coreference resolution of noun phrases. *Proceedings of the ACL* (2001).

Poster available at:
http://www.cs.utah.edu/~ngilbert/coref/posterSession2008.pdf

Visualization of Statistical Measures of Uncertainty

Kristin Potter*

School of Computing, University of Utah

Abstract

One of the important problems facing visualization is the lack of uncertainty information within the visualization context [2]. This work investigates the problem starting from a graphical data analysis standpoint. By using descriptive statistics to investigate uncertainty, the goal of this work is to clearly present the quantifiable aspects of uncertainty, rather than the *unknowns* of uncertainty.

Keywords: Uncertainty, Descriptive Statistics, The Box Plot

1 Introduction

While visualization effectively presents large amounts of information in a comprehensible manner, most visualizations typically lack indications of *uncertainty*. Uncertainty is a term used to express descriptive characteristics of data such as deviation, error and level of confidence. These measures of data quality are often displayed as graphs and charts alongside visualizations, but not often included within visualizations themselves. This additional information is crucial in the comprehension of information and decision making, the absence of which can lead to misrepresentations and false conclusions.

2 Problem Statement

Most visualization techniques that incorporate uncertainty information treat uncertainty like an unknown or fuzzy quantity; a survey of such techniques can be found in [3]. These methods employ the syntax of the word uncertainty to create the interpretation of *uncertainty* or *unknown* to indicate areas in a visualization with less confidence, greater error, or high variation. Blurring or fuzzing a visualization, while accurately expressing the lowered confidence one should have in that data, does not lead to a more informative decision making tool, but instead obfuscates the information that lead to the measure of uncertainty. Such a solution to the problem of adding qualitative information to visualization does not elucidate on the quantitative measures leading to the uncertain classification, and thus is missing some important information.

The goal of this work is to identify and visualize the measures typically thrown under the umbrella of uncertainty. Uncertainty, as the scientific visualization field titles it refers to quality of a measured value and can include measures of confidence, error, and deviation. Statistically, uncertainty is harder to identify, since there are many measures that can add to the qualification of data. Using measures that are statistically meaningful to express the uncertainty in a data set exposes insights to the data that may not have previously been obvious. Adding such quantities to visualizations will improve the effectiveness of visualizations by providing a more complete description of the data, and create better tools for decision making and analysis.

3 Approach

Exploratory data analysis, as coined by John Tukey, uses graphical techniques to summarize and convey interesting characteristics of a data set to facilitate not only an understanding of the given data, but for further investigation and hypothesis testing. These tested graphical methods, such as the Box Plot, Histogram, and Scatter Plot, are easily identifiable signatures of a data distribution, and their simplicity allows for quick recognition of important features, comparison of data sets, and can be substituted for the actual display of data, specifically when data sets are too large to efficiently plot.

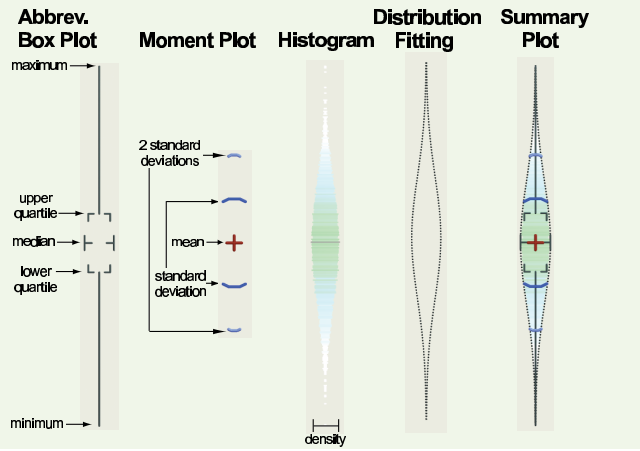


Figure 1: The Summary Plot on a Gaussian distribution.

This work takes inspiration from the visual devices used in exploratory data analysis and extends their application to uncertainty visualization. The statistical measures often used to describe uncertainty are similar to measures conveyed in graphical devices such as the Histogram and Box Plot. The first step of this research investigated the creation of the *Summary Plot* which combines the Box Plot, Histogram, a plot of the central moments (i.e. mean, standard deviation), and distribution fitting. The success of this work has lead to the hypothesis that similar visual techniques can be used in higher dimensional data, by using visual metaphors to indicate a variety of statistical measures as an overlay to traditional visualization techniques.

One problem faced when supplying additional information in a visualization is visual clutter and information overload. This can quickly become a problem in this approach because of the variety of available visual information, and the desire to allow traditional visualization techniques to be incorporated. The solution to this problem is two-fold, the first being a concerted attempt to reduce the visual impact of each visual device, and the second to provide an interactive user interface to provide the viewer with the exact needed information. Such a user interface provides not only the ability to remove unwanted visuals, but also query specific data values, data attributes such as number of observations, and statistical measures.

4 The Summary Plot for 1D Categorical Data

The Summary Plot (Figure 1 is a visual device designed to describe the distribution of a data set including interesting statistical measures as well as tools for distribution fitting and the display of fea-

*e-mail: kpotter@cs.utah.edu

ture statistics on canonical distributions. The Summary Plot incorporates the Box Plot, Histogram, Moment Plot, and distribution fitting into a unified display. Each piece of the Summary Plot is designed to work cooperatively with all other elements, and the addition of a user interface provides the inclusion of as much or as little of the summary elements as desired.

The first piece of the Summary Plot is the *Abbreviated Box Plot*, a refinement of the traditional Box Plot which minimizes the visual impact and maintains the data signature. In this implementation, the minimum and maximum values (including extreme outliers), upper and lower quartiles, and mean are represented. The box surrounding the inner quartile range is reduced to its corners, and the line representing the mean is extended past the edge of the box, to allow the mean to be easily seen, even when the quartiles are spatially close together. While the traditional Box Plot would not be considered visually cluttered, this reduction emphasizes the values expressed by the plot and leaves room for additional visual information.

The histogram is a technique which manageably describes the density of the distribution by binning the data observations. The histogram is added to the Summary Plot as a collection of quadrilaterals, their height reflecting the bin size, and their width indicating density. In addition, density is encoded through each color channel independently, the red channel being normalized log density, the green channel, square root of normalized log density, and the blue channel, normalized linear density. Each of the color channels give a distinct insight to the density and combine to the histogram color seen in Figure 1. Presenting the density of a distribution along with the Box Plot can provide a more clear summary of the data, for example masses of data falling outside the inner quartile range, modality, and the skew, or heaviness, of the data on one side of the mean.

Central moments can be used to reinforce and expand on the statistics of the Box Plot and histogram, and include measures such as mean, standard deviation, skew, and kurtosis or peakiness. The Moment Plot uses glyphs to indicate the location of the mean, and the values of the higher order moments away from the mean. Each glyph is designed to visually express the meaning of that moment, for instance the mean is a small cross that aligns with the median when they are equal (i.e. a normal distribution), and standard deviation is conveyed as brackets falling on either side of the mean. In addition, similar brackets are placed 2 standard deviations away from the mean, giving a sense of the outliers of the data. While the use of some of the higher-order central moments are problematic in that their calculations can be unstable, they do provide some insight to the distribution of the data. Work currently in progress is to determine the most stable methods of calculating the central moments, as well as investigating other descriptive statistics.

Often, understanding the characteristics of a particular data set is less interesting than determining the canonical distribution that best fits the data. This is due to the fact that the feature characteristics of the canonical distributions (such as Normal, Poisson, and Uniform distributions) are well known. The final element of the Summary Plot is a Distribution Fit Plot which is either the best fitting distribution in a library of common distributions, or a distribution chosen by a user. This fit distribution is displayed symmetrically as a dotted line showing the density of the distribution along the axis. Through the user interface mentioned above, the user can also get information about the parameters of the fit distribution, as well as closeness-of-fit statistics.

The resulting Summary Plot provides a simple overview of the data, and techniques for further data exploration and exact statistical measures. The successfulness of this plot will be used as inspiration for higher dimensional data sets.

5 Expanding this work to higher dimensional data

The main challenges to be addressed when expanding the Summary Plot to higher dimensional data sets are finding a visual presentation that is simple and integrates into the data display, and finding visual metaphors such as those in the Moment Plot that are meaningful in 2 and 3 dimensions. The approach that will be taken to accomplish this will be to use transparent surfaces, contours and silhouettes to indicate ranges as well as specific values within the data display. A user interface will be important in allowing a variety of statistics to be used, and to ensure that the display of these measures does detract from existing visualization methods.

A first attempt at such a visualization can be seen in Figure 2. In this figure, uncertainty information of a data simulation of the electric potentials of the heart are shown as transparent surfaces. The surface representing the mean uses the direct mean values, since they have a spatial unit. The position of each of the other surfaces is relative to the mean surface. The values of each of the moments can be seen in the grayscale images at the bottom. While this visualization is not a full proof of concept for the proposed work, it does demonstrate the use of transparent surfaces to express the spatial locations of statistical measures commonly used to describe uncertainty.

The work completed to date includes the development of the Summary Plot and its user interface. Work on the two-dimensional data set of the electric potentials is in progress, and exploration of three-dimensional data sets will begin late fall.

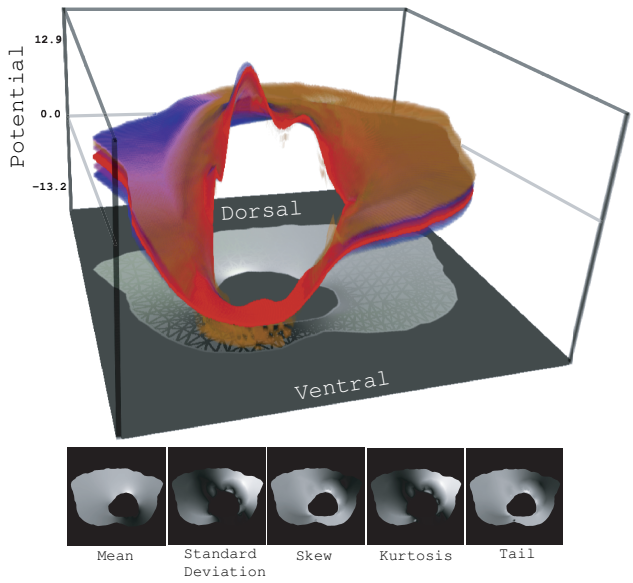


Figure 2: Uncertainty Information for Electrophysiological Potential Models of the Heart. Data provided by S. Geneser and R.M. Kirby [1]

Acknowledgements

*This work was supported in part by ARO(DAAD19-01-1-0013), NSF 03-12479 and ARO grant W911NF-05-1-0395.

References

[1] S. E. Geneser, R. M. Kirby, and F. B. Sachse. Sensitivity analysis of cardiac electrophysiological models using polynomial chaos. In *Proceedings of the 27th Annual IEEE EMBS*, 2005.

[2] C. R. Johnson and A. R. Sanderson. A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications*, 23(5):6–10, 2003.

[3] A. Pang, C. Wittenbrink, and S. Lodha. Approaches to uncertainty visualization. *The Visual Computer*, 13(8):370–390, Nov 1997.

Understanding Large On-chip Caches With CACTI 6.0 *

Naveen Muralimanohar, Rajeev Balasubramonian

1. Introduction

With the proliferation of Chip Multi-Processors (CMPs), computer architecture research today is less focused on the optimization of a single core. Instead, much of the focus has moved towards (i) hardware support for thread creation and (ii) efficient communication between threads. With all inter-thread communication happening through the cache hierarchy, the design of an optimal cache organization is crucial to the success of many-core processors. On-chip caches in modern processors typically dictate more than 50% of the die area [4]. In addition to the performance impact, the organization of these large caches also play a critical role in deciding power and thermal characteristics of modern high-performance processors. The coming decade will likely see many innovations to the multi-core cache hierarchy: policies for data placement and migration, logical and physical cache reconfiguration, optimizations to the on-chip network fabric, and so on.

A large multi-megabyte cache, shared by multiple threads, will likely be partitioned into many banks, connected by an interconnect fabric. The cache is referred to as a non-uniform cache architecture (NUCA) if the latency for an access is a function of the variable distance traversed on the fabric. Our analysis [5, 6] shows that the contribution of the fabric to overall large cache access time and power is 30-60%, and every additional cycle in average L2 cache access time can worsen performance by about 1%. Most NUCA evaluations to date have made simple assumptions when modeling the parameters of the cache: it is typically assumed that the cache is connected by a grid network such that every network hop consumes one cycle. By modeling network parameters in detail and by carrying out a comprehensive design space exploration, we show that an optimal NUCA cache organization has remarkably different layout, performance, and power than that assumed in prior studies.

Most cache evaluations today rely on the CACTI cache access modeling tool [7]. It has been cited by over 500 papers and is an integral part of other high-level power models [1] and processor simulators [2]. While CACTI accurately models small and medium-sized caches, it has many inadequacies when modeling multi-megabyte caches. Future research evaluations can be strengthened by the creation of a tool that accurately models the properties of large caches and provides the flexibility to model new ideas and trade-offs. In this work, we describe the creation of such a tool (CACTI, version 6.0).

1.1. Key Contributions

Earlier versions of CACTI (versions 1 through 5) assumed a Uniform Cache Access (UCA) model in which a large cache is partitioned into a number of banks and connected by an H-tree network that offers uniform latency for every sub-array. The traditional UCA model has a major scalability problem since it limits the access time of a cache to the access time of its slowest sub-bank. For future large L2 or L3 caches, the disparity in access delay between the fastest and slowest sub-banks can be as high as 47 cycles [3]. Hence, having a single uniform access latency to the entire cache will result in a significant slowdown in performance. Besides, the current version of the tool is restricted to use a single network model and a single wire type (global wires with optimal repeater placement). Given that inter-bank network contributes 30-60% of large cache delay and power, the search area of the tool can be greatly enhanced by considering different wire choices for inter-bank network.

The salient enhancements in CACTI 6.0 are:

- Incorporation of many different wire models for the inter-bank network: local/intermediate/global wires, repeater sizing/spacing for optimal delay or power, low-swing differential wires.
- Incorporation of models for router components (buffers, crossbar, arbiter).
- Introduction of grid topologies for NUCA and a shared bus architecture for UCA with low-swing wires.
- An algorithm for design space exploration that models different grid layouts and estimates average bank and network latency. The design space exploration also considers different wire and router types.

*This work is selected to appear in IEEE Special Issue on Top-picks Jan/Feb 2008. This is one of ten papers recognized as year's most significant research publications based on novelty and industry relevance.

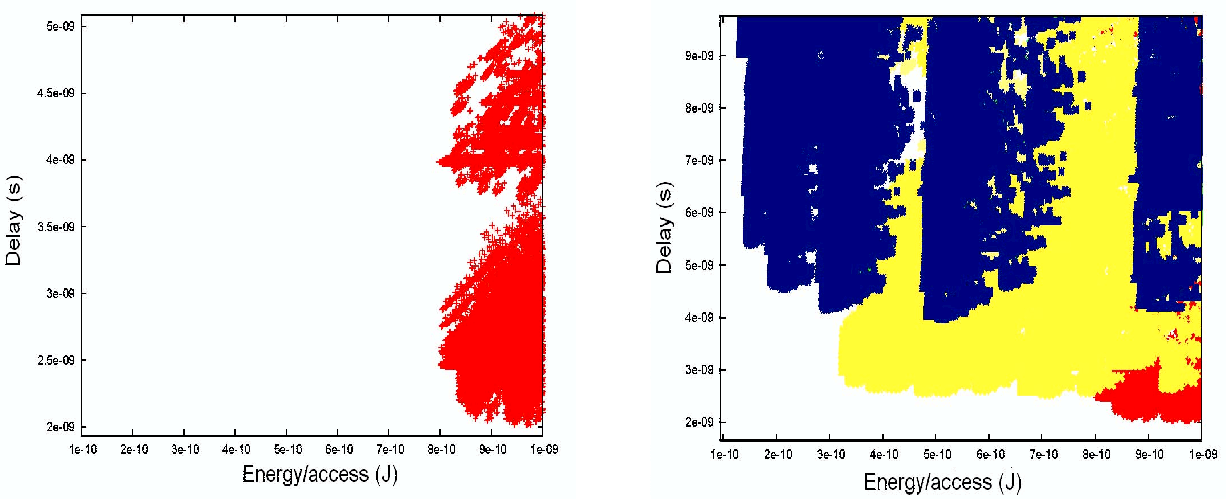


Figure 1. **The figure on the left shows the energy and delay for cache organizations considered by CACTI 5.0. The figure on the right represents the design space exploration in CACTI 6.0. The points in red (bottom region) employ full-swing global wires and are the same points considered in CACTI 5.0, the points in yellow (middle region) employ power-optimized global wires with a 30% delay penalty, and the points in blue (top region) employ low-swing differential wiring.**

- The introduction of empirical network contention models to estimate the impact of network configuration, bank cycle time, and workload on average cache access delay.
- A validation analysis of all new circuit models: low-swing differential wires, distributed RC model for wordlines and bitlines within cache banks (router components have been validated elsewhere).
- An improved interface that enables trade-off analysis for latency, power, and area.

1.2. Tool Impact

One of the key goal of this project is to strengthen current methodologies to model scalable NUCA caches and impact future research on cache hierarchy. In this front, the new updated CACTI tool that does design space exploration of scalable NUCA caches, output cache models that performs 114% better compared to prior NUCA configurations. Due to prudent allocation of network resources, this performance improvement is also accompanied by 50% reduction in cache access power. The second goal is to significantly enhance the search space of the tool and gain insight on power/performance trade-offs in large caches. Figure 1 best summarizes the comprehensive nature of the tool's design space exploration. The red (lower) points in the right graph depict the delay and energy for each cache organization considered by CACTI 5.0 (also shown separately in the left graph). The yellow and blue (middle and upper) points in the right graph represent additional cache organizations now considered by CACTI 6.0. These additional designs are made possible by considering global wires with power-optimal repeater placement (yellow points) and low-swing differential wires (blue points). Such organizations allow the architect to pursue delay-energy trade-offs that were not possible with prior versions of the tool. For example, it is possible to select a cache organization that reduces power by a factor of three, while incurring a 25% delay penalty. This feature is in addition to the NUCA modeling capabilities of CACTI 6.0. The models of the tool's circuit components have been validated to be within 12% of detailed Spice simulations.

Poster Link:
`www.cs.utah.edu/~naveen/poster08.ppt`

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of ISCA-27*, pages 83–94, June 2000.

[2] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[3] C. Kim, D. Burger, and S. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proceedings of ASPLOS-X*, October 2002.

[4] C. McNairy and R. Bhatia. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro*, 25(2), March/April 2005.

[5] N. Muralimanohar and R. Balasubramonian. Interconnect Design Considerations for Large NUCA Caches. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA-34)*, June 2007.

[6] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches With CACTI 6.0. In *Proceedings of MICRO-40*, 2007.

[7] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0. Technical Report HPL-2007-167, HP Laboratories, 2007.