

Supporting Quality of Service in HTTP Servers

J. Fritz Barnes *

barnes@cs.ucdavis.edu

Department of Computer Science
University of California at Davis

With the advent of the WWW [BLCL⁺94], there has been a fundamental shift in the way information is exchanged among systems connected to the Internet. Three elements [YM96] of the WWW make this possible: a uniform naming mechanism (URL) for identifying resources, a protocol (HTTP) [htt95] for transferring information, and the client server based architecture [KMR95]. A client such as a browser uses the URL of a resource to locate an HTTP server that provides the resource. The HTTP server performs the requested services (such as fetching a file or executing a program) and returns the results back to the client.

The architecture of HTTP servers has been studied in great detail and different variations of HTTP servers have been proposed. Much of the work has focused on addressing the performance limiting behaviors [PDG⁺96] of HTTP servers. The research has, thus, focused on developing techniques for eliminating performance bottlenecks arising due to the lack of sufficient CPU, disk, and network bandwidths as well as inherent limitations in the implementation techniques of HTTP servers.

While this has led to a deeper understanding of how HTTP servers operate when there are sufficient resources for various requests, not much work has been done in the case where HTTP servers are overwhelmed by the sheer volume of requests. HTTP requests can be characterized as occurring in bursts [CB96], therefore a server that handles the average workload may very well be overwhelmed occasionally by requests. The behavior of HTTP servers is quite unpredictable in such cases: they either completely bog down with pending requests resulting in unacceptable response times, or start to drop requests indiscriminately. In addition, requests for popular pages have the tendency to overwhelm the requests for other, possibly more important, pages. Further, most HTTP server implementations treat all requests uniformly. A site, thus, cannot assign priorities to different pages or control how its server

resources should be used. For instance, a site may wish to state that a set of specific pages (such as the main page or online store) be always available irrespective of the demands for other pages or that only 20% of its resources should be allocated to anonymous `ftp` requests.

Quality of Service Models for HTTP Servers

In traditional quality of service models, the emphasis has been on developing notions of service guarantees that a server can provide to its clients. For HTTP servers, we develop two views of the quality of service:

- **Client-based view:** the server guarantees specific services to its clients, such as lower bounds on throughput, or upper bounds on response times for certain requests.
- **Server-based view:** sets priorities among requests and limits server resource usage by requests.

We have concentrated on the server-based view of quality of service in our current work [PBO98].

Implementation of QoS for an HTTP Server

We have extended NCSA's `httpd` web server to provide server-based QoS. Our server first reads a specification of the QoS constraints. These constraints define categories or subsets of the document space and are used to associate an absolute or relative resource allocation with documents within the subset.

Our server models each WWW server as a pipe capable of supporting a dynamic byte stream. It determines the capacity of the pipe in terms of number of bytes transferred per second. The capacity of the pipe is dynamically calculated by monitoring the output rate of all HTTP requests at a server. Each pipe is further subdivided into smaller units, called *channels*, as shown in figure 1. A channel forms a connection between a server and a single HTTP client. It

*Joint work with Raju Pandey and Ronald Olsson



Figure 1: Pipes and channels

is the unit of allocation and resource control in the QoS Web Server.

The size of each channel (in terms of bandwidth) is dependent on how many times we subdivide a pipe. For example, if a server indicates that it has a bandwidth of 20 MB/second, the pipe size is 20 MB/second. Further, this pipe can be subdivided into 10 2 MB/second channels or 40 .5 MB/second channels. A channel with 2 MB/second capacity is different from a channel with 0.5 MB/second capacity in that it can service a request 4 times faster than the latter channel. The channel capacity has, thus, implications on response time.¹

The scheduling of HTTP requests is achieved by assigning requests to a channel. The assignment of channels is constrained by the resource allocation (i.e. if no more than 20% of the server is to be allocated to games, than no more than 20% of the channels will be assigned to game requests.)

Results

We have analyzed the performance of the QoS Web server. The QoS Web Server can specify different percentages to be served by documents. When the server is running in contention, i.e. more requests are being received than can be served in a given interval, the server upholds these percentages. The QoS Web Server provides throughput within ten percent of an unmodified version of NCSA's web server. Under contention the QoS Web Server provides better response time, see figure 2, this is because the QoS Web Server drops all requests that it cannot service after the request exceeds a limit on the queue, whereas the NCSA server continues to accept requests even if it cannot handle the requests promptly.

References

[BLCL⁺94] T. Berners-Lee, R. Calliau, A. Luotonen, H.F. Nielsen, and A. Secret. The world-wide

¹The response time does not consider the latency and transmission costs across a wide area network.

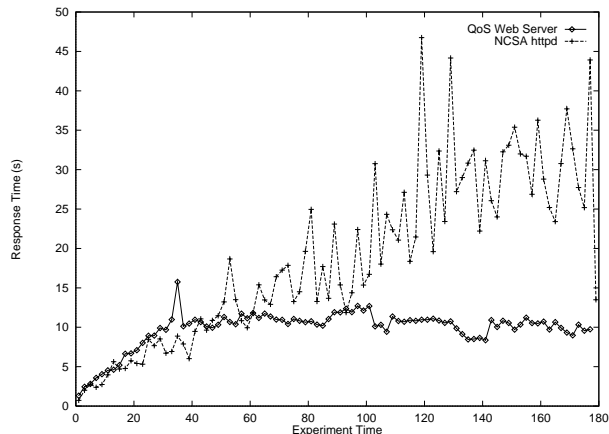


Figure 2: Comparison of response of NCSA and QoS Web servers

- web. *Communications of the ACM*, 37(8):76–82, August 1994.
- [CB96] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In *SIGMETRICS*, 1996.
- [htt95] Hypertext transfer protocol (http): A protocol for networked information. <http://www.w3.org/hypertext/WWW/Protocols>, 1995.
- [KMR95] T. T. Kwan, R.E. McGrath, and D. A. Reed. Ncsa's world wide web server: Design and performance. *IEEE Computer*, pages 68–74, November 1995.
- [PBO98] R. Pandey, J. Fritz Barnes, and R. Olsson. Supporting Quality Of Service in HTTP Servers. In *Proceedings of the Seventeenth Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 247–256, Puerto Vallarta, Mexico, June 1998. ACM.
- [PDG⁺96] F. Prefect, L. Doan, S. Gold, Th. Wicki, and W. Wilcke. Performance limiting factors in http (web) server operations. In *COMPCON*, pages 267–272. IEEE, 1996.
- [YM96] N. J. Yeager and R. E. McGrath. *Web Server Technology: The Advanced Guide for World Wide Web Information*. Morgan Kaufmann, 1996.

Application-Level Misuse Detection in Relational DBMS

Christina Chung, Michael Gertz, Karl Levitt

University of California, Davis

{chungy|gertz|levitt}@cs.ucdavis.edu

INTRODUCTION

The security goals of a computer system can be specified explicitly by *security policies* - what actions a subject can or cannot perform on objects under specified conditions. The security policies are enforced by *security mechanisms*. Compromising the security mechanisms is *intrusion* whereas violating the security policies without compromising the security mechanisms by legitimate users is insider abuse. Misuse includes both intrusion and insider abuse.

Misuse can be handled by *signature based* and *anomaly based* approaches [CFMS95]. In signature based detection, audit logs are matched against a database of known attack patterns. In anomaly based detection, it is assumed that the set of intrusive and misuse events equals the set of anomalous events and such anomalies can be detected in the audit records. *Profiles* are derived from audit logs and policies to characterize the normal system usage by users. If the new audit records deviate from the profiles, alarm is raised.

We are interested in detecting misuse in relational database systems at the application level using techniques from both approaches. Our research is motivated by the fact that the majority of applications running on computer and network systems nowadays are information systems where the valuable data stored is vulnerable to various security threats. However, previous misuse detection systems reside at the system or network layer ([CFMS95]). Audits at this level are not suited for misuse detection at the information level because the semantics of the applications are not reflected in the low-level audit logs. Unlike these systems, our approach focuses on the *application layer* which exploits the database schema underlying the applications. That is, the relation schemas (and attributes), relationship between attributes and the integrity constraints.

SYSTEM ARCHITECTURE

System Description We use the relational model as the underlying data model for the database schema. We take into consideration both the *structure* and *semantics* of the database application. This is achieved

by defining a set of aggregate features which are abstract concepts derived from primitive features. Primitive features are features that are directly measurable by the auditing system. Examples are login/logout time of the users, access frequencies of attributes of the schema, delta values for updates on attributes of tables, or the time of occurrence of the events. Aggregate features are defined in terms of primitive features and the database schema. For example, the number of times attributes A, B are accessed, $Freq(A), Freq(B)$, are primitive features. Suppose the schema gives us a measure, $Dist(A, B)$, of how far away these attributes are in the corresponding entity-relationship diagram. We can define an aggregated feature *shortest semantic distance* between A, B as $\frac{Freq(A) - Freq(B)}{[\max(Freq(A), Freq(B))]} Dist(A, B)$.

Components The proposed detection system is located between the DBMS and the database application. It consists of the following components: (1) a data collection and processing module, (2) a security policy specification module, (3) a misuse detection module.

Data Collection and Processing Module In this module, the auditing functionality of the DBMS is used to audit the relevant primitive features. The collected audit data are then pre-processed into a format understandable by other modules of the system. This includes handling missing and incorrect data, type mapping, and representation condensing.

Security Policy Specification Module A security policy language will be developed to aid the security officers in specifying security policies and domain knowledge such as known misuse patterns. The language is designed with the following criteria in mind. (1) It should support temporal constructs since time is an important feature. It should be able to express, for instance, the point in time relative to other events, the temporal relationship between two events, or the n th occurrence of an event during an interval. (2) It should be easy to write and understand since the security officers may not be computer experts. (3) It can be efficiently implemented so that the detection system

can respond to malicious behavior in a timely manner. (4) It can be easily converted to the knowledge representation used by the misuse detection engine.

The objects of the rule-based language are (1) users or user groups, (2) tables and attributes of the database schema, and (3) events. An event states which user performs what operations on which table and attributes. The operations are operations supported by the data manipulation language, such as select, insert, update, and delete. Events can be primitive events, or composite events (which are disjunctions, conjunctions or sequences of primitive events).

Misuse Detection Module The misuse detection engine accepts audit data from the data collection module from which normal user profiles are determined. Based on the policies specified by the policy specification engine and the user profiles generated, alarm is raised if new audit records do not match the user profiles. We intend to use data mining techniques to discover the profiles for the users. The profiles based on the security policy language are logical, rule-based specifications that may contain temporal constructs, primitive and aggregated features. It is believed that the normal usage patterns form *working scopes*, i.e., clusters of regions the user usually works with. Clustering techniques ([Eve73]) can be used to discover these clusters where the similarity function can be based on the aggregated features. Deviation from the profiles is detected if new audit records fail to be classified into one of the discovered clusters. Other techniques used in knowledge discovery systems ([DF95], [FPSSU96], [PSF91]) can be borrowed to discover the usage patterns from the set of aggregated and primitive features.

FRAMEWORK

The interactions between different modules in our system are depicted as follows:

1. The system description defines a set of aggregated features from the primitive features selected by the data collection module and the given relation schemas of the database application.
2. Given a database schema for a database application, the set of users and the set of operations audited, the data collection module collects the data of the selected primitive features for events in the database system. The audit data are pre-processed (e.g., aggregated) and fed into the misuse detection module.
3. Based on the primitive and aggregated features defined, the security officers specify the domain knowledge and security policies using the security policy specification module. These are con-

verted to an internal knowledge representation understandable by the misuse detection module.

4. The misuse detection module determines the profiles for the users and user groups from the audit data. Anomalies are detected based on the profiles and the security policies specified.

FUTURE WORK AND CONCLUSION

An extension of the system would be a response module that can react to malicious activities detected. This includes generating a visual presentation to aid the security officers in identifying source of anomalies and to carry out counter-measures against malicious activities. The system can be adapted to a distributed environment such as a federated database (an integration of heterogeneous databases) over a network. Aggregation of information across individual DBMS can lead to potential improvement in performance. Issues that need to be addressed include resolving structural and semantic conflicts among security policies of individual DBMS and aggregation of data from multiple heterogeneous systems.

We plan to conduct a detailed study of misuse detection in relational database systems using the anomaly based approach. A prototype of the misuse detection module has been implemented and preliminary results show our approach is feasible. Our focus at the application layer is novel and we expect to show improvement in performance over existing misuse detection systems.

References

- [CFMS95] Silvana Castano, Maria Grazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. Addison-Wesley, 1995.
- [DF95] Karsten M. Decker and Sergio Focardi. Technology overview: A report on data mining. Technical Report CSCR TR-95-02, Swiss Scientific Computing Center, 1995.
- [Eve73] Brian Everitt. *Cluster Analysis*. John Wiley & Sons - New York, 1973.
- [FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. *Advances In Knowledge Discovery And Data Mining*. AAAI Press/The MIT Press, 1996.
- [PSF91] Gregory Piatetsky-Shapiro and J. William Frawley. *Knowledge Discovery In*

Databases. AAI Press / The MIT Press, 1991.

Concurrent Language Support for Rapidly Prototyping Interoperable Applications*

Eugene F. Fodor (fodor@cs.ucdavis.edu, <http://avalon.cs.ucdavis.edu/>)

Ronald A. Olsson (olsson@cs.ucdavis.edu)

Department of Computer Science, University of California, Davis 95616

Introduction

Developers of distributed software systems need better ways of providing communication between different, remote programs (e.g., file servers or databases) in a simple, efficient way. Unfortunately, existing mechanisms that address this problem are complicated, and so system designers continue to seek novel, simpler methods to express such communication. This paper presents an alternative approach to this problem. Synchronizing Interoperable Resources (SIR) extends the communication mechanisms in the SR programming language.

This work will benefit distributed system researchers and developers as SIR's expressive communication facilitates prototype development.

Related Work

Many systems contain mechanisms for performing distributed communication. DCOM [BK98], Java RMI[Mic98], and CORBA [Gro98] allow remote methods to be invoked through remote objects. RPC (Remote Procedure Call) was first investigated by [BN84], which subsequently lead to a significant body of research. ILU [JSLJ97] uses an intermediate language to provide a language neutral environment for distributed systems.

SR Background

The expressive power of SR's communication mechanisms helps the programmer create complex communication patterns with relative ease [AO93]. For example, a file server written in SR can service "open" invocations only when file descriptors are available. Furthermore, it can schedule invocation servicing by priority. The following code fragment accomplishes this easily through SR's **in** statement. Both scheduling and synchronization expressions (**st** and **by**, respectively) constrain the execution of each arm of the **in** statement.

```
process fileservice()
do true ->
  in rmt_open(fname, pri) st fdcnt > 0 by pri ->
    rfs_service(fname, OPEN)
```

*This work is partially supported by the United States National Security Agency University Research Program.

```
    fdcnt--
[] rmt_close(fname, pri) by pri ->
  rfs_service(fname, CLOSE)
  fdcnt++
...
ni
od
end process
```

The priority (**pri**) could be mapped to an array of host names, thus allowing preference for specific hosts.

Design

At a high level, our model consists of communication between three programs: the remote service requester (RSR), the remote service host (RSH), and the registry program (RP). The RSR is the client, the RSH is the server, and the RP negotiates the connection between the other two. More details appear in the Implementation section.

To add inter-program communication to the SR programming language, a registration mechanism is necessary. To accomplish this, the following built-in functions were added to the language:

register(regname, oper) Registers the operation **oper** with the RP on the local host under the name **regname**.

unregister(regname) Removes the registry entry from the RP.

bind(regname, host, opcap) Contacts the RP on the specified host. Assigns the remote operation to the capability (i.e., pointer to operation) **opcap**.

unbind(opcap) Dis-associates **opcap** from the remote operation.

This design implies that type checking must be performed at run time, since the parameters **opcap** and **oper** of the functions **bind** and **register**, respectively, may have any signature and are bound at run time. Our initial design has two limitations: operations may only be registered on the local machine, and the registration name must be unique for each operation. These problems can be solved, respectively, by adding another parameter to the register function and by using path names like Java RMI does.

Example

SIR communication allows for rapid prototyping. Consider, for example, building a server for some online database. The clients and server both create remote queues using SR's send and receive mechanisms to establish a complex dialog. Each client first registers its queue to the RP and then binds the known server queue ("knownqueue") to its capability (**capqueue**). In turn, the server binds to the unique name transferred by the client and replies on the corresponding capability (**remqueue**) after performing some data processing. The following SIR code fragment shows this described interaction.

Client Code (multiple, different clients):

```
#Each client requests a unique name from its local RP.
uname(uid)
register(uid, myqueue)
do not done->
  bind("knownqueue", "knownhost", capqueue)
  #myhost() returns local host name.
  send capqueue(uid, myhost(), data)
  #Receive processed data.
  receive myqueue(data)
od
unbind(capqueue)
unregister(uid)
```

Server Code (single server):

```
register ("knownqueue", queue)
do not done ->
  receive queue(name, host, data)
  #Invoke process_data as separate thread.
  send process_data(name, host, data)
od
unregister("knownqueue")

proc process_data(name, host, data)
  var remqueue:cap (datatype)
  bind(name, host, remqueue)
  #Process data (not shown).
  #Send data after processing.
  send remqueue(data)
  unbind(remqueue)
end
```

Implementation

The SIR implementation extends the standard SR compiler and run-time system (RTS). The compiler-generated code encodes the signatures of capabilities and operations into RTS calls. These RTS calls provide dynamic type checking by comparing the encoded signatures. When an operation is registered, this encoding is sent as part of the registry entry to the RP. An RSR initiates the RTS comparison when it attempts to bind to an operation. The encoding is a simple text representation of the signature.

The bind function provides a direct connection between the RSR and RSH. The RP establishes this connection by forwarding the network address of the

RSR to the RSH. The RSH then contacts the RSR, which unblocks the RSR from the bind. The RSH then sends back an acknowledgment to the RP to complete the request and thus frees it to establish other connections. If the RP times-out or receives an error from the RSH, then the RP sends an error back to the RSR. Upon invocation, an invocation block is generated at the client side and is sent over the connection to the RSH. Unbind simply disconnects and cleans up any data structures as necessary.

We have built a working version of SIR's compiler and runtime system with dynamic type-checking support. A distributed, HTTP server written in SR is being modified to function as a testing ground for SIR. We are also looking at the performance of binds, registers, and invocations with respect to other systems as well as doing qualitative comparisons of communication mechanisms. Additionally, we have begun designing a distributed file server written in SIR.

Conclusion

Currently, we have only investigated SIR to SIR communication, but intend to extend this work to allow any language to utilize SIR operations. In particular, we are exploring methods for supporting the full scope of communications mechanisms between different languages. Also, we will look at SIR performance against other interoperable packages. We are also looking at design issues for supporting partial, distributed termination, indirect binding, and implicit binding.

References

- [AO93] Gregory Andrews and Ronald Olsson. *The SR Programming Language: Concurrency in Practice*. Benjamin/Cummings, Redwood City, CA, 1993.
- [BK98] Nat Brown and Charlie Kindel. *Distributed Component Object Model Protocol (DCOM/1.0)*, January 1998. Microsoft.
- [BN84] Andrew Birrell and Bruce Nelson. Implementing remote procedure calls. *ACM Trans. Computer Systems*, 2(1):39-59, February 1984.
- [Gro98] Object Management Group. *The Common Object Request Broker: Architecture and Specification.*, Feb 1998.
- [JSLJ97] Bill Janssen, Mike Spreitzer, Dan Larner, and Chris Jacobi. *ILU 2.0alpha12 Reference Manual*, November 1997. XEROX PARC.
- [Mic98] Sun Microsystems. Java remote method invocation specification. July 1998.

Providing Fine-Grained Access Control For Mobile Programs Through Binary Editing *

Brant Hashii Raju Pandey
Parallel and Distributed Computing Laboratory
Computer Science Department
University of California, Davis, CA 95616
{hashii, pandey}@cs.ucdavis.edu

1 Overview

With the advent of the WWW, there is considerable interest in developing a runtime infrastructure for mobile code. In the mobile programming model [CHK95, Tho97, CPV97], programs, called mobile programs in this paper, have the ability to compute at a host, stop their execution, migrate to another host, and continue their execution. Mobile programs are appealing because they support efficient utilization of network resources and extensibility of information servers. Also, the model is ideally suited for extensible distributed system structures such as the Internet.

However, since they cross administrative domains during their executions, they have the ability to access a site's protected resources. In this paper, we present an approach that allows a site to control access to its conceptual resources – resources which are explicitly called and have well-defined interfaces. Examples of conceptual resources include file systems, window managers, and database servers. In this approach, a site enumerates a set of constraints on accesses to its resources. A set of tools enforce

these constraints on a mobile program by integrating the code for checking access constraints into the Java classes that form both the mobile program and the resources. Executions of the resulting mobile program satisfy all access constraints imposed by the site, thereby protecting its resources.

This approach depends only on the ability to recognize accesses to resources. Any language that provides identifiable interfaces to resources can be used.

2 Mechanism

Our mechanism consists of two parts: a policy language for specifying access constraints and a tool for enforcing the constraints.

Our policy language is a simple declarative language of the form, *deny(C1.M1 – > C2.M2) when B*. This states that method *M1* of class *C1* cannot invoke method *M2* of class *C2* when *B* is true. The when clause can be omitted, in which case *B* defaults to true, indicating that access should always be denied. Likewise, *M1* and/or *M2* can also be omitted, implying that access should be denied to any method of the given class. Further, the subject *C1.M1* can be omitted, resulting in no class accessing *C2.M2*. The boolean expression *B* can be based on the parameters of the methods or the state of the system. The format of the boolean expression is similar to Java boolean expressions. Elements of the boolean expression can include method invocations, fields of the object and parameters of the method. An example is *#N.X op V*, where *N* specifies either the subject object *C1* or the target object *C2*. *X* specifies the parameter of the method. *op* is a comparison operator and *V* is some

*This work is supported by the Defense Advanced Research Project Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0221. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Project Agency (DARPA), Rome Laboratory, or the U.S. Government.

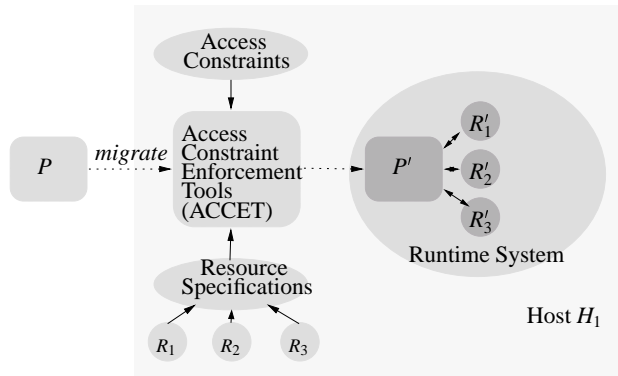


Figure 1: Access control enforcement

value. These simple boolean constraints should be enough for most access constraints. However, in order to provide a more extensible system, the policy language also has an *add* command for adding a security class object which can be used for implementing more general procedural access constraints.

Access constraints are inherited by a class's subclass. They cannot be overridden, although they can be strengthened. For example, suppose R_2 is a subclass of R_1 and we have the two constraints: $deny(P \rightarrow R_1) \text{ when } B_1$ and $deny(P \rightarrow R_2) \text{ when } B_2$. Then the actual constraint for P accessing R_2 is $deny(P \rightarrow R_2) \text{ when } B_1 \text{ or } B_2$.

The editing tool takes the access constraint statements and inserts code into the Java classfiles in order to make sure that the constrained method is only invoked when B is not true. If B is true, then a security exception is thrown. The location where this code is added depends on the type of the policy statement. For statements of the form $deny(C1.M1 \rightarrow C2.M2) \text{ when } B$, the constraint check is added to the code body of method $M1$ of class $C1$ before each method invocation of $C2.M2$. For statements of the form $deny(- \rightarrow C2.M2) \text{ when } B$, it is more efficient to include the constraint check within the body of the code for $C2.M2$. Thus, we edit both incoming mobile programs and the interfaces for local resources.

3 Performance

We performed experiments on a 266 MHz Pentium II running Red Hat Linux 5.0. We compared this ap-

proach with the Java's security manager [AG96]. In Java, one sets a security policy by creating a security manager class and setting it as the system's security manager. Then, each protected resource makes a call to the system to get the security manager and another to the security manager to check if access is allowed.

We incur a small one time overhead of about 0.08 seconds for reading a small policy file and modifying a small class file. However, execution time is superior to Java's security manager approach. Our approach inlines security checking code and avoids the overhead of the method invocations.

4 Summary

We have proposed a method where access constraints are specified independent of the runtime system. These constraints are then implemented by adding code directly into the mobile program and a host's resources. The results show that both the time and space overheads of this approach are moderate. Further, it performs better than Java's runtime system in certain cases.

References

- [AG96] K. Arnold and J. Gosling. *The Java Programming Language*. Addison Wesley, 1996.
- [CHK95] D. Chess, C. Harrison, and A. Kershenbaum. *Mobile Agents: Are They a Good Idea?* Technical report, IBM T.J. Watson Research Center, 1995.
- [CPV97] A. Carzaniga, G. P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *ICSE '97. Proceedings of the 1997 international conference on Software engineering*, pages 22–32, Boston, MA, May 1997.
- [Tho97] T. Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, 29(3):213–239, September 1997.

Low-Power Design of Page-Based Intelligent Memory

Justin Hensley, Aneet Chopra, Mark Oskin, Timothy Sherwood, Aamir Farooqui and Frederic T. Chong
University of California at Davis

Abstract

Advances in DRAM technology have led many researchers to integrate computational logic on DRAM chips to improve performance and reduce power dissipated across chip boundaries. The density, packaging, and storage characteristics of these intelligent memory chips, however, present new challenges in power management.

We introduce *Active Pages*, a promising architecture for intelligent memory based upon pages of data and simple functions associated with that data [OCS98]. We evaluate the power consumption of three design alternatives for supporting Active-Page functions in DRAM: reconfigurable logic, a simple processing element, and a hybrid combination of reconfigurable logic and a processing element. Additionally, we discuss operating system techniques to manage power consumption by limiting the number of Active Pages computing simultaneously on a chip.

1 Introduction

As processor performance grows, both memory bandwidth and power consumption become serious issues. At the same time, commodity DRAM densities are increasing dramatically. Many researchers have proposed integrating computation with DRAM to increase memory bandwidth and decrease power consumption. The operating and packaging characteristics of DRAM chips, however, present new constraints on the power consumption of these systems.

To use Active Pages, computation for an application must be divided, or *partitioned*, between processor and memory. For example, we use Active-Page functions to gather operands for a sparse-matrix multiply and pass those operands on to the processor for multiplication. To perform such a computation, the matrix data and gathering functions must first be loaded into a memory system that supports Active Pages. The processor then, through a series of memory-mapped writes, starts the gather functions in the memory system. As the operands are gathered, the processor reads them from user-defined output areas in each page, multiplies them, and writes the results back to the array data structures in memory. Simulation results show up to a 1000X speedup on such applications running on systems that use Active-Page memory over conventional memory.

This paper focuses upon ongoing work which evaluates the power consumption of Active-Page implementations. First, we describe three alternative designs based upon reconfigurable logic, small processors, and a combination of the two. Then we present a some brief power estimates for these designs. Finally, we conclude with some future directions for this work.

Acknowledgments: Thanks to Venkatesh Akella for his helpful comments. This work is supported in part by an NSF CAREER award to Fred Chong, by Altera, and by grants from the UC Davis Academic Senate. More info at <http://arch.cs.ucdavis.edu/RAD>

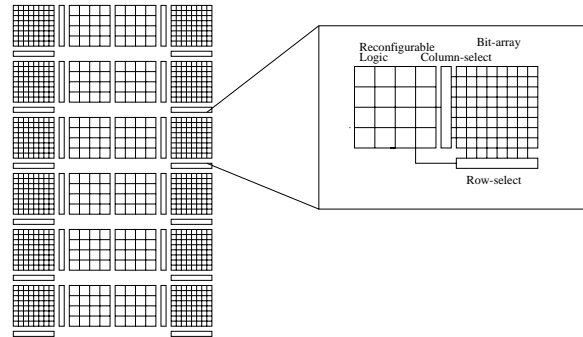


Figure 1: The RADram System

2 Design Alternatives

In this section we briefly present three architectures for Active Page memory systems. These architectures consist of RADram, or reconfigurable logic in DRAM [OCS98], an architecture consisting of only a small RISC or MISC [Jon98] like processing element near each page of data, and a hybrid architecture which provides a small processing element with reconfigurable logic processing capability.

Currently, within all of our designs we adopt a *processor-mediated* approach to inter-page communication which assumes infrequent communication. When an Active-Page function reaches a memory reference that can not be satisfied by its local page, it blocks and raises a processor interrupt. The processor satisfies the request by reading and writing to the appropriate pages. The processor-mediated approach reduces power by not requiring extensive inter-page communication networks, and global clocking.

2.1 Reconfigurable Logic

Our initial implementation of Active Pages focused upon the integration of DRAM and reconfigurable logic. For gigabit DRAMs, a reasonable sub-array size to minimize power and latency is 512Kbytes [I⁺97]. The RADram (Reconfigurable Architecture DRAM) system, shown in Figure 1, associates 256 LEs (Logic Elements, a standard block of logic in FPGAs which is based upon a 4-element Look Up Table or 4-LUT) to each of these sub-arrays. This allows efficient support for Active-Page sizes which are multiples of 512Kbytes.

Each LE requires approximately 1k transistors of area on a logic chip. The Semiconductor Industry Association (SIA) roadmap [Sem97] projects mass production of 1-gigabit DRAM chips by the year 2001. If we devote half of the area of such a chip to logic, we expect the DRAM process to support approximately 32M transistors, which is enough to provide 256 LEs to each 512Kbytes sub-array of the remaining 0.5-gigabits of memory on the chip. DeHon [DeH96] gives several estimates of FPGA area, and existing prototype merged DRAM/reconfigurable logic designs demonstrate this to be a realistic design partitioning [M⁺97].

2.2 Processing Elements

An alternative architecture for Active Page implementations is that of a small RISC or MISC [Jon98] type processing element used to execute user level process functions in memory. Several low-power processing cores currently exist and Active Page designs can leverage the existing and future work in this area. However, a processing engine designed for Active Pages must satisfy two constraints: power and size. While high-performance low-power designs exist, such as the StrongARM [San96], their size makes them prohibitive for use in Active Page memory. Due to these constraints, it will necessary to design a processor core specifically for this application. A modified MIPS instruction set machine is in the process of being modeled. This processor will support most of the traditional MIPS features, except that it will not have any coprocessors or any of the associated instructions, and will not it have any support for exception handling.

2.3 Hybrids

An alternative architecture from RADram or a small processing element, is a hybrid, which mixes a RISC-like processor with reconfigurable logic. Such an architecture is presented in [HW97]. Here we extend this notion by simplifying the processing element core, and shrinking the reconfigurable logic array. The hybrid architecture would consist of a simplified processing element described above in Section 2.2 and a smaller reconfigurable logic array than that presented for RADram in Section 2.1. The configurable logic array would be on the order of 128 LEs in size.

3 Power Consumption

Our estimates show that our initial RADram design using reconfigurable logic has the lowest power consumption of our three alternatives. A further advantage for RADram is increased chip yield due to fault-tolerant designs. The complexity and number of Active-Page functions, however, is limited by the amount of reconfigurable logic available for each page. The processing element and hybrid designs have more computational power. Furthermore, we expect to refine both of the latter designs to use less power. Refinements such as clock-gating, and asynchronous processor designs [FGT⁺97] will help reduce the power consumed by processor, and hybrid processor / reconfigurable logic processor element designs.

4 Future Work

Our previous work [OCS98] has shown that Active Pages are a promising architecture for intelligent memory with substantial performance benefits. Our current work focuses upon the power consumption of Active Page implementations. Estimates show that several design alternatives exist, but Active Page DRAMs will benefit from advances in low-cost packaging technologies. Future work will pursue both architectural and software approaches to reducing power consumption. This work will include detailed chip-level synthesized VHDL models, of both processors, and FPGAs. Furthermore, we will explore the software techniques discussed in this paper using cycle-by-cycle simulation of the processor and memory subsystem.

An increasing amount of research is being done in the area of asynchronous logic, due to the fact that clock skew and distribution are serious issues that a logic designer must face.

Clock speed has an effect on both the area and power consumed by a design. As the clock speed increases, more complex structures are needed to distribute the clock accurately throughout the chip. The increased clock speed has two disadvantages. First, the larger clock drivers consume more power. Second, overall power dissipation increases linearly with clock frequency.

Recent developments have made it possible to use asynchronous logic in more complex logic designs. The AMULET2e processor [FGT⁺97] is an asynchronous ARM[Fur96] 32-bit processor that consumes 30% of the power and takes 54% of the area of the ARM810[Fur96]. An asynchronous RISC core provides a low-power means of implementing an Active Page processing element.

Another possibility is the use of an asynchronous FPGA. Commercial FPGAs are currently targeted to synchronous logic and do not lend themselves to the implementation of asynchronous logic[HBBE94][HBBE92]. Various new FPGA designs have been developed solely for the purpose of implementing self-timed logic[HBBE94][MA98]. Work done at the University of Washington has shown that it is possible to implement real circuits in an asynchronous FPGA[BEHB]. Asynchronous FPGAs may prove to be a viable architecture for Active Page memory system processing elements.

References

- [BEHB] G. Borriello, C. Ebeling, S. Hauck, and S. Burns. The triptych fpga architecture. Univ. of Washington, Seattle WA 98195.
- [DeH96] André DeHon. *Reconfigurable Architectures for General-Purpose Computing*. PhD thesis, MIT, 1996.
- [FGT⁺97] S.B. Furber, J.D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. Amulet2e: An asynchronous embedded controller. In *Async '97*, 1997.
- [Fur96] S.B. Furber. *ARM System Architecture*. Addison Wesley Longman, 1996.
- [HBBE92] S. Hauck, G. Borriello, S. Burns, and C. Ebeling. Montage: An fpga for synchronous and asynchronous circuits. In *2nd International Workshop on Field-Programmable Gate Arrays*. FPL, August 1992.
- [HBBE94] S. Hauck, S. Burns, G. Borriello, and C. Ebeling. An fpga for implementing asynchronous circuits. *IEEE Design and Test of Computers*, 11(3), Fall 1994.
- [HW97] J. Hauser and J. Wawrzynek. Garp – a MIPS processor with a reconfigurable coprocessor. In *Symp on FPGAs for Custom Computing Machines*, Napa Valley, CA, April 1997.
- [I⁺97] K. Itoh et al. Limitations and challenges of multigigabit DRAM chip design. *IEEE Journal of Solid-State Circuits*, 32(5):624–634, 1997.
- [Jon98] D. Jones. The ultimate risc. *ACM Computer Architecture News*, 16(3):48–55, 1998.
- [M⁺97] Masato Motomura et al. An embedded DRAM-FPGA chip with instantaneous logic reconfiguration. In *1997 Symposium on VLSI Circuits*, 1997.
- [MA98] K. Maheswaran and V. Akella. Pga-stc: Programmable gate array for implementing self-timed circuits. *International Journal of Electronics*, 84(3), 1998.
- [OCS98] Mark Oskin, Frederic T. Chong, and Timothy Sherwood. Active pages: A computation model for intelligent memory. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA '98)*, Barcelona, Spain, 1998. To Appear.
- [San96] S. Santhanam. Strongarm sa110: A 160mhz 32b 0.5w cmos arm processor. *Hot Chips*, VIII:119–130, 1996.
- [Sem97] Semiconductor Industry Association. The national technology roadmap for semiconductors. <http://www.sematech.org/public/roadmap/>, 1997.

DOVES ABSTRACT

Keith C. Herold

PURPOSE

There are a number of resources available to individuals who are involved in computer security, both in publications and on the Internet. However, many of these resources are difficult to use, primarily because of a lack of organization and sheer volume of material. The DOVES (Database Of Vulnerabilities, Exploits, and Signatures) project is an attempt to organize and reduce information on bugs, vulnerabilities, and exploits to a more accessible format.

Besides the information resource DOVES represents, the database will be used to facilitate the taxonomy of vulnerabilities and exploits while also providing an exploit 'signature' which details what a session of a particular attack would look like. For example, a user may have noticed that her system has been repeatedly attacked through buffer overflows. A search for "buffer overflows" within the Vulnerabilities area might reveal a that her OS is especially vulnerable to a particular class of buffer overflow exploits (login overflows, for example). Links lead her to the Exploits area detailing the various exploits useful against her system. Finally, a Signatures link reveals that her logging utilities exactly match a pattern of an exploit known as 'foobar'. Additionally, her Vulnerabilities search might have yielded a fix particular to that class of overflow which allows her to simultaneously be able to plug not only the leak she looked for, but any other similar leaks she does not know about. Best of all, instead of having to search multiple sources for information, she is able to find all the relevant information in one central, standard source.

FORMAT

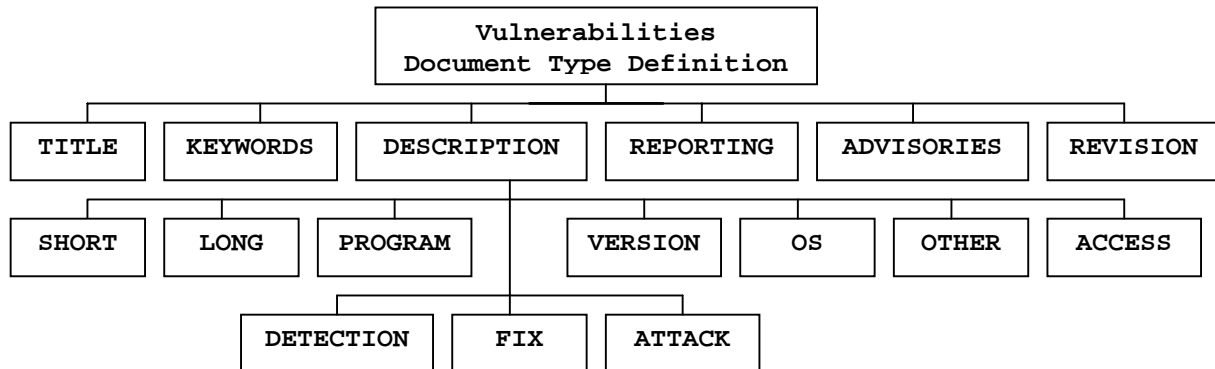
Five undergraduate students under the direction of Professor Matt Bishop are distilling security reports into an SGML format defined by Professor Bishop. The entries are obtained from a variety of sources including the BUGTRAQ mailing list and Digests, as well as older documents local to the project. Once entered into the SGML format, an in-house developed database engine called OliveBranch is used to manage and search the database based on a variety of search options including keywords, title, and program affected. Additionally, once a user has located the document(s) she is looking for, a format interpreter called JADE can be applied on-the-fly to the SGML document to convert it to any one of several well-known and widely available document formats, including rich text (*.rtf), HTML (*.html), and Postscript (*.ps).

The database (including engine) will be made available through several venues including CD-ROM and the World-Wide Web. A key concern of the project is that UC Davis not become a clearing house for would-be hackers but remain a resource for responsible individuals to have access to large amounts of information to help secure their sites against attacks or bugs. We address this issue by separating the entries on vulnerabilities from the associated attacks and their signatures. In essence there are three databases: one containing vulnerabilities, one exploits, and one signatures. Only authorized personnel such as bona-fide researchers or contributing commercial interests, will be able to access all portions of DOVES. Access will be granted on an as yet undetermined basis.

ENTRY DESIGN

Since the intent of DOVES is to provide readily accessible information, it is critical that DOVES support a variety of file formats. This criterion led to the selection of SGML (Standard Generalized Markup Language) as a base format for DOVES.

SGML is a standard method for ensuring that each document written conforms to an accepted structure by the use of a Document Type Definition (DTD). Essentially, a DTD contains information about what sections must be in a document, as well as the order of the sections and the document's hierarchy. In other words, SGML is concerned about the structure of a document and not its content.



Elements of a Vulnerabilities Entry

The selection of SGML as the base format has two strengths: every document produced and verified against the DTD has the same fields which is convenient for parsing into the database, and the use of an interpreter like JADE allows the contents of the file to be exported into other file formats allowing the use of special formatting like bold and italics.

OLIVEBRANCH DESIGN

Like the SGML format, it is important that accessing the information be as widely supported as possible. Therefore the engine implementation is written entirely in Java. The Java philosophy is one of platform independence and allows us to reap the benefits of the 'write once, read anywhere' concept.

OLIVEBRANCH is confirmed to run on the production releases of JRE 1.1.6 (Java Runtime Environment) for Windows NT and 95, as well as Solaris 2.6. These are the only two official ports of the JRE; however, *any* Java Virtual Machine that is fully compliant with Sun's Java Specs as well as the Java Foundation Classes 1.0.3 will run OliveBranch. Every effort has been made to eliminate any platform dependence code.

The engine itself currently allows multiple searches in the form of logical AND operations upon filename, keywords, title, OS, program, and reporting date, as well as editing of existing files or the creation of new entries. All output is guaranteed to conform to the appropriate DTD. Additionally, all areas of the database run within their own thread, so it is possible to conduct multiple searches upon one or all areas simultaneously.

REFERENCES

1. Bob DuCharme. 1998. *SGML CD*. Prentice Hall PTR, New York.
2. Bruce Eckel. 1998. *Thinking in Java*. Prentice Hall PTR, New York.
3. James Clark. *JADE*. <http://www.jclark.com/jade/>
4. Sun Microsystems. <http://java.sun.com/>

Security Policy Specification Using a Graphical Approach

James A. Hoagland*

hoagland@cs.ucdavis.edu

Department of Computer Science

University of California at Davis

Security forms a core component of many systems. Traditionally, what is meant by security has been formally described for certain interpretations of confidentiality, integrity, but less precisely for availability. However, what is realistically meant by security for a particular system¹ varies from system to system, and possibly depending on the interaction of the system with its environment. The military would likely have a different definition of security than a bank, a university, or a home user would. They each have their own needs that should be reflected when securing their systems. In addition, one can describe security at various system levels. A security policy is a description of the security goals for a system and how a system should behave in order to meet these goals. An example policy is that the request for a purchase and its approval must be from different users, each authorized for the particular operation.

Since the security goals for a system effect, among other things, how the security mechanisms on the system are configured, it is important for the security policy to be stated clearly and precisely. Additional benefits accrue from the precise formulation of a policy that can be directly used to configure the security mechanisms or can be used to formally reason about the effect of the policy. A common security need is to restrict access to the resources on a system. This is reflected in a constraint security policy². These constraints describe, in general terms, restrictions on how the system should behave. We focus on constraint security policies in this abstract.

It is additionally important to be able to state the policy for the appropriate system, where the semantic information that the policy is based upon is available. The approach presented in this abstract allows policies to be stated for a variety of systems. In particular, it can be applied to any system that can be modeled by our system model. In our model, a system contains one or more objects that interact

through events such as accesses and creation methods. Objects possess attributes whose values may vary with time. Events have fixed parameters and values and occur at a particular time. Operating systems and programs are systems that meet this model.

We have developed a language and approach to facilitate specification of access control policies. The language, called the Language for Security Constraints on Objects (LaSCO), represents a policy as an annotated directed graph, called a policy graph. The nodes of the graph denote objects and the edges denote events among the objects. Policy nodes and policy edges are each annotated with a pair of predicates: the domain predicate and the requirement predicate. Each predicate is a boolean condition on object attribute values and event parameter values. When applied to a particular object or event, a predicate evaluates to true if it matches or false if it doesn't. Predicates may use Prolog-style logical variables to interrelate attribute and parameter values for different objects and events.

The domain of a policy graph is defined by the nodes and edges of the policy graph and their domain predicates. The domain matches a portion of a system execution if each of the edge predicates match a event and each of the node predicates match a object. In addition, the choice of objects for nodes must be consistent: the object that the node matches must be the originating object of the events that matched the outgoing edges of the node and must be the destination object of the events that matched the incoming edges. This describes the situation under which the policy applies. The requirement of the policy graph (which describes the constraint for the program whenever the domain matches) is described by the requirement predicates. The requirement matches if each of the requirement predicates match the object or event that their node or edge matched when the domain matched. If the requirement does not match, then the policy has been violated. Certain details have been skimmed over here; full details of the syntax and semantics of LaSCO can be found in [HPL98].

There are several reasons we find LaSCO appealing. The policy can be specified separately from a program. The language is not tied to any particular enforcement mechanism and the policies so stated may be used in different ways. For example, policies may be compiled into a program, used by

*This work is sponsored in part by ARPA contract DOD/DABT63-93-C-0045 and by the Intel Research Council.

¹We use "system" generally here, to include any computational entity. In particular we can model a program execution, a file system, an operating system, a workgroup, and a network of hosts.

²Other types of security policies are ones that describe how trust is managed, describe how to choose the security options in setting up a network connection, and ones that describe how to respond to a security violation.

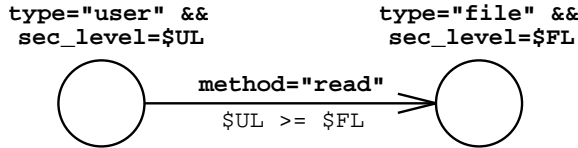


Figure 1: Policy graph for the basic security property

a reference monitor [And72] or wrapper to mediate access to system objects, or to check an audit trail off-line. We can state policies that apply whenever a certain pattern of accesses is encountered. This can be seen as a template for a general policy rather than the set of configuration details that an access control matrix describes. The language can describe policies that span several events over a period of time and can describe policies that depend on conditions other than the state at the time of check. The language is visual which might make it easier for humans to cope with, a serious issue since policy specification will be human intensive. At the same time, the language has a formal basis. Having a formal basis and a formal semantics promotes our ability to reason about policies and about system with policies employed.

Figure 1 presents a simple LaSCO policy graph depicting the simple security policy of Bell-LaPadula [BL73]. The policy specifies that if a user is reading a file, the security level of the user must be at least as great as that on the file. In this graph, the left node denotes a user object, whereas the right node denotes a file object. The edge between the nodes denotes an access relationship as specified by a read event. The domain matches when the event “read” occurs between a user and a file. The requirement specifies that the level of user should be at least as high as that of the file. This is specified by annotating the graph with expression $\$UL \geq \FL . Note that $\$UL$ and $\$FL$ denote logical variables that gets bound to `sec_level` attributes of user and file objects respectively.³

As another example, we depict a separation of duty policy in Figure 2. This policy requires that for a particular pair of related accesses (in this case “request” and “approve” of a “purchase”), the users involved must be distinct. This is to avoid conflict of interest problems. We depict this by two edges with different user sources leading to single node representing a purchase. The system in this case is a system where there is a separate function for requesting policies and getting them approved.

We have also defined a formal semantics for LaSCO policies and have investigated operations on policies and

³In policy graph depictions in this abstract, the bold text by a node or edge is the domain predicate of that node or edge and the standard text annotation is the requirement predicate. Undepicted predicates have an implicit “True” expression as that predicate.

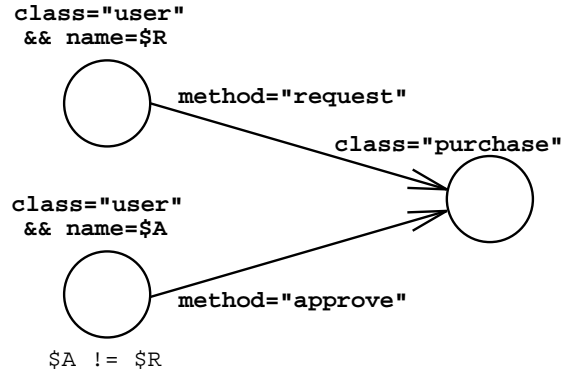


Figure 2: Policy graph for purchase request and approval separation of duty

policy interactions and extensions to LaSCO, which we omit due to space considerations. These are described in [HPL98]. We have a LaSCO policy enforcement framework for object-oriented programs, which we describe briefly here. A schema extraction tool constructs a program schema graph, with nodes denoting class definitions and edges between the nodes representing method invocations found in the program source. The security policy user interface provides a convenient mechanism for the user to create and edit LaSCO policies. This tool presents the schema graph for the program in question and a library of generic policies as a means of facilitating specification. A compiler takes the program, policy graphs, and program schema and generates code for both implementing the program and for enforcing the access constraints. We have begun implementation of this framework for Java.

References

- [And72] J.P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Vols. I and II, USAF Electronic Systems Division, Bedford, Mass., October 1972.
- [BL73] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, The MITRE Corp., Bedford, Mass., May 1973.
- [HPL98] James A. Hoagland, Raju Pandey, and Karl N. Levitt. Security policy specification using a graphical approach. Technical Report CSE-98-3, The University of California, Davis Department of Computer Science, Davis, CA, July 1998.

Instruction Scheduling Using Integer Programming

Jack Liu & Kent Wilken

Department of Electrical and Computer Engineering
University of California at Davis

1. Introduction

Optimal instruction scheduling means scheduling or reordering instructions to get the most efficient in program running time. Integer programming is a promising instruction scheduler which can schedule instruction optimally and hence decrease the number of clock cycles a program will take compared with the traditional list scheduling, which is a heuristic instruction scheduling.

This project explored instruction scheduling using integer programming, which had been successfully applied in optimal global register allocation^[1]. An efficient integer programming formulation for optimal local instruction scheduling has been developed and will be extended in future work. This paper presented an approach by using integer programming to schedule instructions optimally within a basic block. In this paper, the integer programming model was introduced and the experimental results were presented.

2. Directed Acyclic Graph Reduction Techniques

A *basic block* is a maximal sequence of instructions that can be entered only at the first instruction and exited only from the last instruction. The method of basic block scheduling requires the construction of a dependence graph that represents the instructions issue constraints on the possible schedules of the instructions in a block and the degrees of freedom in the possible schedules. To present the problem within a basic block, *Directed acyclic graph* (DAG) was used here. The nodes in a dependence DAG represent machine instructions and its edges represent dependencies between the instructions. The weight of the edges represent the latency between the two connected instructions. For further information about DAG and basic block, reader can refer to [4].

To simplify a DAG, a series of techniques had been developed as follows:

- Tighten lower bound and upper bound of each instruction. Lower bound of an instruction is the earliest clock cycle at which the instruction can be issued. Upper bound is the latest clock cycle the instruction can be issued. Lower bound and upper bound could be tightened by the node's position in the DAG.

- Remove redundant DAG edges. DAG edge_{jk} from node J to node K is redundant if there is a path_{jk} from J to K with length equal to or longer than the length of edge_{jk}. In here, path_{jk} preserves the partial order between J and K, path_{jk} enforces the required delay D_{jk}

- Order equivalent nodes. DAG nodes in a set S are equivalent if all nodes in S have common immediate predecessors and successors, and for each immediate predecessor/successor, the latency is the same from the immediate predecessor/successor to each node in S. DAG is rewritten with equivalent nodes placed in a sequential chain in arbitrary order without affect the program's correctness.

3. Integer Programming Formulation

Integer programming (IP) is the problem of computing a setting of n integer variables (x₁, ..., x_n), such that an objective function $\phi(x_1, \dots, x_n)$ is maximized/minimized under the constraint

$$A * (x_1, \dots, x_n)^T \leq B$$

for an integer matrix A and an integer vector B^[2].

An IP formulation is used to schedule instructions within a basic block. The formulation uses a series of scheduling variables. *Scheduling variable* is a 0-1 integer-program variable for each instruction, for each clock cycle. Decision variable, like x_iⁱ, represented the decision to schedule or not schedule the jth instruction at the ith clock cycle. The whole basic block should be finished with no more than U clock cycles. U is the clock cycles obtained from list scheduling.

There was a series of *constraints* in the form of linear expressions to restrict the possible values which the decision variables could take. The IP constraints were encoded as follows:

Must Schedule Constraint was used to ensure each instruction within current the basic block must be scheduled in one of the clock cycles. Hence, each instruction i should satisfy

$$\sum_{j=0}^{U-1} X_i^j = 1$$

Issue Constraint was used to ensure the single-issue restriction, which required that no more than one instruction can be issued at the same cycle. Hence, each clock cycle j should satisfy

$$\sum_{i=0}^{n-1} X_i^j \leq 1$$

where n was the number of instructions within current basic block.

Dependency Constraint was used to enforce the instructions' partial issue order and issue delays were satisfied. Hence, for

an instruction j should be issued D_{jk} cycle(s) earlier than instruction k . The issue time for j and k should satisfy

$$\sum_{i=0}^{U-1} (i * x_k^i) > \sum_{i=0}^{U-1} (i * x_j^i) + D_{jk}$$

After the IP formulation was set up, it was passed to IP's solver component. In here, CPLEX 6.0, a commercial linear programming solver, was used. The result of IP scheduler would show which instruction should be issued on which clock cycle. If the solution was feasible, meaning that all the constraints were satisfied, the upper bound of current basic block would be decreased by 1 cycle, and the IP model would be rebuilt again and let IP scheduler to solve until the problem was infeasible. Then, the last feasible solution would be the optimal instruction schedule.

4. Experimental Results

A prototype IP scheduler was built into the Gnu C Compiler, GCC version 2.8.0, to replace GCC's list scheduler. Using SPEC92 Integer Benchmarks, results were generated for the PA-RISC architecture using GCC's highest optimization level. Two target processors were used. Both of the processors were based on HP-PA700. The difference was one had a 1-cycle load delay; the other had a 2-cycle load delay.

Figure 1 showed the IP results for 1-cycle load delay. In this target processor, GCC generated 71,716 basic blocks. 71,691 basic blocks were solved optimally by list scheduling. The remaining 25 basic blocks were solved optimally using the IP scheduler. 11 of the 25 basic blocks had shorter schedules compared with list scheduling's result.

Figure 2 showed the IP results for 2-cycle load delay. In this target processor, GCC generated 71,719 basic blocks. 71,521 basic blocks were solved optimally by list scheduling. The remaining 198 basic blocks were solved by using IP scheduler. In those 198 basic blocks, IP scheduler solved 192 basic blocks optimally and 30 basic blocks had shorter schedule than list scheduling's result. IP scheduler also failed to solve 6 of the 198 basic blocks within the 100 seconds time limit.

5. Summary and Future Work

Lupers and Marwedel had done an approach to map NP-complete instruction scheduling problem into an IP model [3]. In Lupers' paper, the IP solver time would increase exponentially with respect to the IP problem's upper bound, which was the number of clock cycles a basic block will take to finish. This was due to the large search space that has to be investigated by the IP solver. The computation time may dramatically grow with the number cycles of upper bound on his paper. In our research, a more efficient formulation was generated. The solver time grew linearly with respect to the upper bound.

More efficient formulation and more techniques will be developed in future work. After all the IP problems on the single-issue RISC processor are solved, optimal local scheduling IP model for multi-issue, statically scheduled processor will be built based on current IP model. After that, optimal global scheduling for single-issue, statically scheduled processor will be studied. The final goal of the research is to build a complete IP model for multi-issue processor to do instruction scheduling globally.

References

- [1] David Goodwin and Kent Wilken. *Optimal and near-optimal global register allocation using 0-1 integer programming*. Software-Practice and Experience, 26(8):929-965, August 1996.
- [2] H.P. William. *Model Building in Mathematical Programming*. New York: Wiley & Sons, 1990.
- [3] Rainer Leupers and Peter Marwedel. *Time-constrained code compaction for DSP's*. In IEEE Transactions VLSI Systems. 5(1):112-122, March 1997.
- [4] Steven S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufmann, 1997.

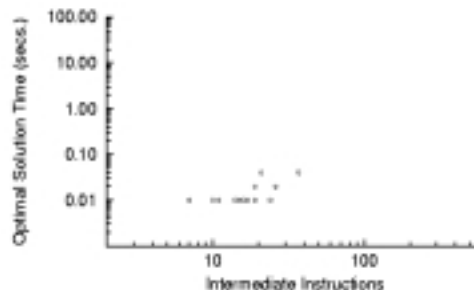


Figure 1. Experimental results for target processor with 1-cycle load delay.

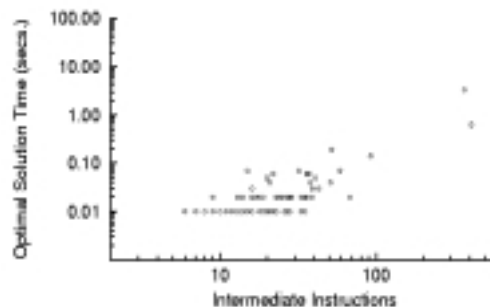


Figure 2. Experimental results for target processor with 2-cycle load delay.

Integer Programming Based Register Allocation

Chris Lupo and Kent Wilken

UC Davis, EU II, Rm. 2211

Davis CA 95616

lupo@ece.ucdavis.edu

wilken@ece.ucdavis.edu

Because of the central role that register allocation plays, both in speeding up the code and in making other optimizations useful, it is one of the most important—if not the most important—optimizations.[2]

Register allocation is the assignment of program variables and compiler generated temporaries to a machines real registers. With register allocation playing such a critical role in compiler optimization, it is desirable to have methods that can compute an optimal register allocation in which there are a minimum number of load and store instructions. Prior work has shown that integer programming, a mathematical procedure that attempts to find an optimal integer solution to an objective function subject to a set of linear constraints, can be effectively used to obtain optimal register allocation [1] The project described in this paper is based on that prior work.

The primary objective of this project is to significantly speedup the integer programming based register allocator ORA (Optimal Register Allocator). Significant speedup is expected from the discovery of application-specific methods for solving register allocation integer programs. Discovery and implementation of more efficient integer programming formulations is also expected to produce significant speedup. The speedup achieved from this project will make integer programming based register allocation more broadly applicable. The study of efficient formulation methods and application-specific integer program solution methods will also lead to better understanding of the register allocation problem.

Efficient integer programming formulations to the register allocation problem can produce significant speedup in the time required to solve the integer program. There are several methods that can be implemented to achieve this speedup.

One of these methods currently being implemented is the identification of nonspilling

symbolic registers. Prior to the register allocation phase of compilation, there are an unlimited number of symbolic registers used to represent variables and compiler generated temporaries that must be allocated to real registers when they are used during program execution. Symbolic registers that are not residing in a real register are said to have been spilled to memory. If a symbolic register has been spilled, it must be loaded from memory prior to each use of that symbolic register. Likewise, after each assignment (definition) of a symbolic register that must be spilled, a store instruction is required. It is then left to the register allocator to determine whether a symbolic register is spilled at a given point in the program.

A symbolic register can be marked as a nonspilling symbolic register if it can be shown that the symbolic registers live range is contained in a region of the program where it is not necessary to spill that symbolic register. A symbolic register's live range consists of the points in the program from which the symbolic register is first defined to the point of its last use. A symbolic register is said to interfere with another symbolic register if both symbolic registers' live ranges intersect.

Consider a symbolic register A. To determine whether A can be marked as nonspilling requires analyzing the region of code that contains the live range of A. If within the live range of A it can be shown that there are fewer instances of symbolic registers other than A used or defined than there are real registers in the machine, then it can be guaranteed that A does not need to be spilled. Notice that this analysis does not consider the total number of symbolic registers that interfere with A. There may be symbolic registers that interfere with A that are not used or defined within the live range of A. As an example, consider the partial control flow graph in Figure 1. For this example, symbolic registers A, B and C all interfere with each other. Assume for this example that there are

two real registers available for allocation in the target machine. When the region of code containing the

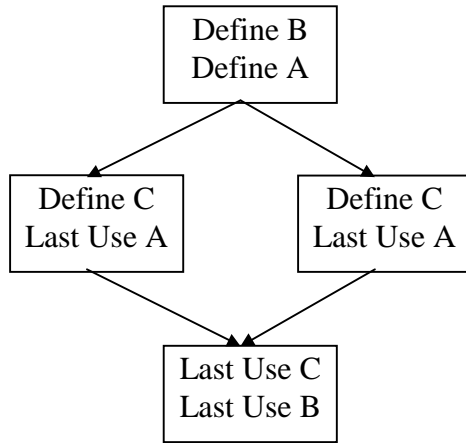


Figure 1: Code Fragment With Nonspilling Symbolic Register A

live range of A is analyzed, it can be shown that there is reference to only one other symbolic register, C, within the live range of A. Since there is only one other reference and there are two real registers available, it can be guaranteed that A does not need to be spilled. Stated otherwise, it can be guaranteed that there is another symbolic register that can be spilled at an equivalent or lesser cost than the symbolic register that was marked as nonspilling. In this example, the other symbolic register is B.

Determining that a symbolic register is a nonspilling symbolic register helps to simplify the integer program model. Since it can be guaranteed that nonspilling symbolic registers do not spill, the integer program does not need to determine whether these symbolic registers are stored to or loaded from memory. This effectively reduces the size of the integer program, thus obtaining a speedup in solution time.

Currently, a nonspilling analysis is being implemented on a basic block level. Preliminary results of this analysis are promising. Initial comparisons were made for xliisp benchmark program in the SPEC92 integer benchmark suite. This program contains 357 functions, each of which were solved optimally both with and without the basic block nonspilling analysis performed. The average solution time of the functions when nonspilling analysis is not performed is 0.94 seconds, while the average solution time of the functions when nonspilling analysis is performed is 0.86 seconds. This is an

average speedup of approximately 8.5 %. It was also found that the total number of constraints in the integer program was reduced by 3.9 % and the total number of integer program variables was reduced by 4.9 %.

This analysis will be extended to a global nonspilling analysis. This is expected to provide significantly more speedup for the time required for optimal register allocation.

Other methods for obtaining speedup are also being investigated. These methods include trying to eliminate redundant or unnecessary constraints in the integer programming model, as well as potentially using a heuristic to provide the integer programming solver with an initial, suboptimal solution. Cumulatively, these developments are expected to dramatically speedup the solution time of the integer programming based register allocator ORA.

References

- [1] David Goodwin. *Optimal and Near-Optimal Global Register Allocation*. PhD thesis, Computer Science Department, UC Davis, March 1996.
- [2] D. Patterson and J. Hennessy. *Computer Architecture A Quantitative Approach*, Second Edition, p. 92, Morgan Kaufmann, 1996.

MoHCA-Java^{*}: A Tool for C++ to Java Translation

Scott Malabarba[†]

malabarb@cs.ucdavis.edu

Department of Computer Science

University of California at Davis

Introduction

MoHCA-Java is a tool created to assist with converting C++ to Java. It performs detailed analysis on a C++ abstract syntax tree (AST); the parameters of the analysis can be specified and extended very quickly and easily. This tool should prove to be quite useful. As Java increases in popularity and effectiveness, many people find it desirable to convert old C++ or C programs to Java. The complexity of this task varies greatly between programs; sometimes the conversion is trivial, sometimes high level program design must be reconsidered, and the entire program rewritten. Tools exist to automate the conversion process, generally relying on some form of text stream processing or partial parsing. This type of tool is effective for the easier cases of conversion. However, these tools have certain limitations - their capacity for semantic analysis is minimal, and any change which requires a design decision simply cannot be handled. Our hypothesis is that a tool which performs rigorous analysis on a C++ program, providing detailed output on the changes necessary, will make conversion a much more efficient and reliable process.

Implementation

MoHCA-Java is built as a layer of abstraction on top of the Gen++ code analysis language [DE]. Gen++ provides constructs for traversing and analyzing AST's generated by the front end of a C++ compiler. We have defined a simple

rule-based language, dubbed MJL (MoHCA-Java Language), which is used to specify patterns in the C++ program to search for, and the action(s) to take when those patterns are found. The user simply supplies a file containing analysis rules written in MJL, which MoHCA-Java checks against the C++ AST. This allows for very rapid and flexible specification of MoHCA-Java functionality, including extension to cover custom C++ libraries or new features in Java or C++. MJL captures the semantics needed to perform basic pattern-matching on an AST, while dispensing with the complexity of a more powerful language like Gen++. The simplicity of the rule language also permits efficient verification of correctness.

We designed the system to be easily extendable on a deeper level as well, in a modular style. The MJL parser, intermediate representation, and Gen++ analyzer program are distinct components connected by well-defined interfaces; it is possible to modify or replace one module without disturbing the others. For example, the MJL syntax and grammar are constantly evolving - this is done easily by editing the lex and yacc specifications and rebuilding the analyzer executable. Figure 1 provides a high-level overview of MoHCA-Java's design and function.

Rules take the general form *properties:conditions=>action*. MJL contains constructs for nested conditions, boolean operations, slot traversal, and explicit invocation of rules. Rules can be assigned several optional properties, including name, category, difficulty and link to another rule. For the present, the "action" contains the message text to be printed upon satisfaction of the rule. The language is sufficiently expressive to allow rules

^{*}Most Helpful C++ Analyzer for Java

[†]This work was initiated as a term project for ECS 289E, Practical Tools for Programmers/Domain Specific Languages, in Spring 1998.

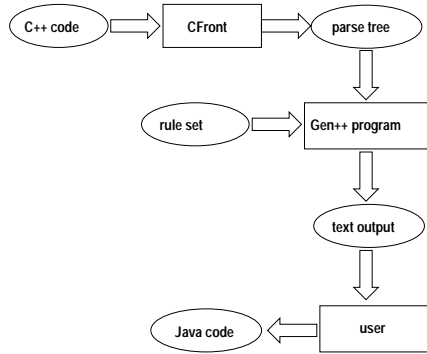


Figure 1: Overview of system.

for all Java/C++ differences currently catalogued [DD98] [Jav96].

Analysis Methods

We have cataloged differences between Java and C++ and separated them into categories and difficulty levels. The categories are syntax, semantic, library, and design. Syntactic differences involve simple changes in the syntax of an expression; for example, C++ classes contain public/private sections, while in Java each member is classified as public or private. Semantic differences are deeper, involving changes in the semantic definition of an expression. An example is the condition of an "if" statement - in Java this must be a boolean, unlike C++. Library changes are simply a switch from one function or global variable to another - say, from printf to System.out.print. Design issues require some redesign of the program, typically because of a feature which exists in one language but not the other - for instance, multiple inheritance or pointer usage. Difficulty level is a measure of the time and system knowledge required to enact a given change. We make the assumption that for most programs the bulk of the changes required will be low-difficulty syntax and library issues, with a few complex design issues on the other end of the scale. This categorization lends itself to efficient division of labor - the more numerous, time-consuming minor tasks could be assigned to a junior programmer,

while a senior programmer handles more difficult issues requiring design decisions or specialized knowledge. At present, output consists of a simple text file with a series of messages describing the changes needed, with the location of each in the original source code. Future plans include a more complex and user-friendly scheme, where an HTML version of the source code is generated, including hyperlinks from each piece of offending C++ code to a description of the change needed. Some statistical analysis of how "fit" the source code is for conversion will also be provided.

Discussion

Currently, no actual translation is done. The user must code all changes manually, using the output provided as a guide. MoHCA-Java would be even more useful if it could be extended to perform the conversion wherever possible, leaving only those changes which are impossible to automate for the user to do. Gen++ does not have any mechanism for translation, so it will be necessary to either extend Gen++ or use another tool. An optimal solution may be to use MoHCA-Java in conjunction with one of the previously mentioned translation tools. We have recently begun to test the system by using it to convert several programs of varying complexity both with and without its assistance, and comparing the time and expertise required for each task. Manually identifying necessary changes and locating them in code can be difficult, and we predict significant savings in time and energy.

References

- [DD98] Deitel and Deitel. *Java: How to Program*. Prentice Hall, 1998.
- [DE] P. Devanbu and L. E. Eaves. Gen++ - an analyzer generator for c++ programs.
- [Jav96] JavaSoft. The java language environment. Technical report, JavaSoft, 1996. <http://www.javasoft.com/docs/white/langenv/index.html>.

Double-ended Queue on Hostile Platform

Henry Naftulin
naftulin@cs.ucdavis.edu

Department of Computer Science
University of California at Davis

This work is an addition to the paper “Stack and Queue Integrity on Hostile Platforms” by P.Devanbu and S Stubblebine. This work considers double-ended queue structures as an addition to the stack and queue structures.

Devanbu and Stubblebine introduced a specific way to protect the integrity of the stack and the queue structures used for computation on a powerful but potentially hostile machine. Such computations could be initiated by a smart card -- device composed of a circuit card in epoxy or a similar substance, that has various electronic tamper detection devices. These devices are usually of a credit card size, and therefore have heat dissipation difficulties. As a result, smart cards have low data transfer rates, very limited memory etc. The goal of this paper is to suggest several ways of creating double-ended queues and compare them. Two implementations of a double-ended queue structure are suggested: a three-stack implementation and two queues implementation. A third way, “limited knowledge” digest, was tried for double-ended queue implementation. It has been conjectured that this strategy could be used for implementing queues and stacks, but a successful replay attack was found for a double-ended queues.

The three-stack implementation (Figure 1) has an advantage of not requiring keyed cryptographic signature scheme. It can, therefore, be implemented with MD5 algorithm, or any other strong non-keyed hashing algorithm.

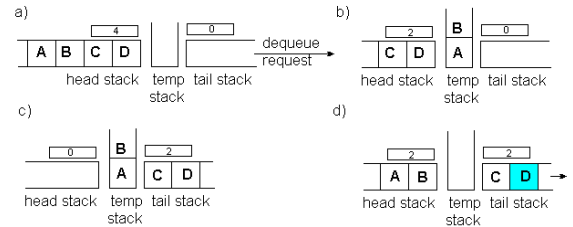


Figure 1: Three-stack implementation of a double-ended queue.

The protocol that uses the three-stack implementation is the simplest one among the three protocols discussed. Another advantage worth mentioning is the fact that stacks are used as black box algorithms, so any implementation of stack that guarantees stack's integrity will be applicable. Three-stack implementation has several drawbacks. First of all, it requires a counter, which limits the number of elements that can be pushed onto the stack. The length of signatures also affects the maximum depth of the stack, though. Secondly, it requires eight cells in the smart card – all other methods have lower requirements. Finally, its response time ranges between $O(1)$ and $O(n)$ where n is the number of the items in the double-ended queue. Though amortized cost for each operations turns to be $O(1)$, applications might be sensitive to the fact that sometimes the response will be delayed.

The two-queue implementation (Figure 2) of the double-ended queue has an advantage of relatively simple protocol. It requires only four smart card memory cells, which is worse than the “limited knowledge” implementation, but better than three-stack implementation.

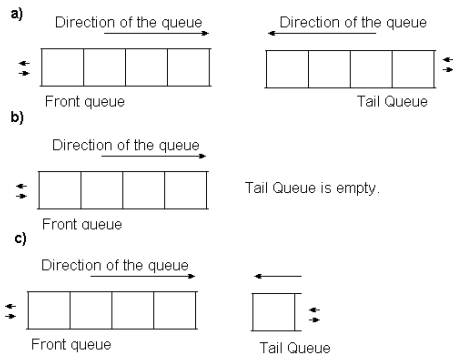


Figure 2: Two-queue implementation of a double-ended queue.

All double-ended queue requests take $O(1)$ operations to complete. Keyed cryptographic signature requirement is one of the disadvantages

of this implementation. Another possible disadvantage is that this scheme uses the internal structure of queues, so it might not work for other queue implementations, which would still guarantee integrity of the queue.

References:

Devanbu P. and Stubblebine S., "Stack and Queue Integrity on Hostile Platforms," 1997

Blum M., Evans W., Gemell P., Kannan S. and Noar M. "Checking the correctness of memories," Algorithmica, 1994

Amato N. and Loui M. "Checking linked data structures," FTCS, 1994

Fredman M., Komlos J., and Szemerédi E. "Storing a sparse table with $O(1)$ worst case access time," Journal of ACM, July 1984

Applications of Traditional Compiler Technology to Behavioral Logic Synthesis

Mark Oskin
University of California at Davis

Abstract: While behavioral synthesis of digital systems has existed for several years, wide spread adoption of the technology has remained relatively sparse. Several reasons exist for this, including compilation time and conservatism within the industry. Acceptance, however, is also linked to the quality of the logic produced. In this paper we explore the effects of traditional compiler optimization techniques on logic production. To do this, a MIPS assembly instruction set to high-level VHDL translator was constructed. Using the latest GNU¹ C² compiler targeted to MIPS, a series of test circuits were synthesized when coded directly in VHDL and when transformed through the GNU optimizing compiler. The circuits were then compiled to CMOS logic using the Synopsys design compiler. The results are explored in terms of resulting logic area and performance as well as the specific effects of traditional C optimization techniques.

1 Introduction

High level compilers reduce design time and project maintenance costs. Today, compilers for software generation improve overall code execution time and size, comparable to even hand-coded implementations. The principle reason for the improved high level of quality available in software compilers is the relative complexity involved in hand-optimizing machine code for modern multiple issue pipelined processors, and the advances in compiler optimization technology.

Currently, high-performance logic designs are largely hand-implemented. However, it is likely as transistor density and overall project complexity increase, the same forces which drove the software engineering market towards compiler-based code generation will force the hardware engineering market to high-level behavioral synthesis of digital systems. In order to attract designers and remain competitive with manual logic designs, behavioral logic compilers must produce hardware of comparable area and speed. Eventually, non-regular hardware designs will be faster require less design time and fewer post-market maintenance costs when implemented in a high-level behavioral description language.

Abstractly, software and behavioral logic compilers both read as input a description of an extended control data-flow graph (CDFG) and transform it into either machine language or logic, respectively. Syntactic differences exist since each form of compiler is designed for a separate problem domain. However, if we focus on only a specific subset of each problem domain we find a common core subset of language constructs. This core subset includes all arithmetic operations, all basic non-looping control flow statements, and depending upon the problem domain of the behavioral compiler, either fixed or fixed, and open-ended loop constructs. With this core subset of language constructs we can construct a hypothetical compiler that reads in this description, constructs a CDFG and then contains two code-generators, one tuned for software generation and another for logic design. Of course, before either software code or logic designs are generated from a CDFG, the CDFG is first optimized to remove any common sub-expressions, fold-constants, and compute statically available values, along with a host of other potential optimizations.

Given that such commonality exists between software and behavioral logic compilers a natural question to pose is whether components of existing traditional software compilers can be used to improve behavioral logic synthesis. Internally, after parsing of the core subset of language constructs, each form of compiler essentially manipulates the CDFG representation, optimizing it before applying further target-specific optimizations. In order to experiment with this concept a tool was constructed which transforms the output of an optimizing C compiler into VHDL. This tool was then executed on various combinatorial circuits, and the results were examined. Although the results are inherently specific to the synthesis path, and the behavioral compiler compared against, they do indicate that certain traditional compiler optimization techniques are beneficial to logic design, and further work on the high-level CDFG optimizations is required in the behavioral logic synthesizer used for comparison.

2 MIPS Assembly to VHDL Code Transformation

In order to explore the effects of traditional compiler techniques on logic synthesis a system was constructed which uses the GNU C/C++ compiler to transform a C function into optimized MIPS assembly language, and from this intermediate representation to synthesizable VHDL code. To simplify the task of MIPS assembly to VHDL code transformation only a restricted subset of C is supported. This subset is summarized as:

- Only 32bit integer data-types are supported
- Only a single function is supported. Multiple supporting functions can be used, however, the C compiler must inline these functions completely.
- All global variables are transformed into entity port declarations in the resulting VHDL code. Due to the combinatorial nature of the circuit produced, a global variable can only be read or written to, but not both within the function.
- Arbitrary conditional nested “if” statements are supported.
- Only “for” loop constructs are supported with fixed upper and lower bounds and increment stride.

This subset of C is expressive enough to encode several example and real-world combinatorial circuits.

3 Methodology

In order to evaluate the effects of traditional compiler optimizations on resulting logic production a sequence of short C programs were constructed. The short programs were chosen to illustrate various optimizations the C compiler could perform on the CDFG representation of the C function. After constructing a C representation of the function a hand-coded VHDL implementation was performed which closely paralleled the C version. The C code was then processed through the GNU C/C++ compiler using the highest level of overall optimization (“-O3”). The resulting MIPS assembly language output was then transformed using the tool described in Section 2 above, into VHDL code. This VHDL code, along with the original analogous VHDL code were both then synthesized to logic using the Synopsys behavioral logic compiler. The compiler was instructed, in both cases, to put maximum effort into compilation and optimization. The results were then compared. Several example circuits were constructed designed to explore specific optimizations. These example circuits contain expressions that can be optimized by efficient compilers. These optimizations include: constant folding, strength reduction, constant value detection, code motion, and common sub-expression elimination.

4 Results

In this section we present the results of synthesizing the application benchmarks described in above. The results are explored in terms of overall area consumption, performance, and logic-macro functions used. Since high-level synthesis involving scheduling and

Benchmark	Native VHDL Speed	Native VHDL Size	Native VHDL Comp.	Native VHDL Optms.	Trans. C Speed	Trans. C Size	Trans. C Comp.	Trans. C Optms.
Application1	125.90ns	593	2 add. 1 inc.	None	127.09ns	458	2 add.	Constant Folding
Application2	123.69ns	658	1 add. 3 inc. 1 mult2	Poor Strength Reduction	118.32ns	425	1 add. 1 inc.	Strength Reduction.
Application3	239.08ns	5920	2 add. 1 inc. 1 mult.	None	244.00ns	5568	1 add. 1 inc. 1 mult.	Zero Detection
Application4	137.86ns	543	4 add. 1 inc.	Loop Invariant Code Motion	137.86ns	543	3 add. 1 inc.	Loop Invariant Code Motion And addition by zero.
Application5	288.08ns	21634	6 adder 4 mult. 2 sub.	CSE	373.79ns	23256	8 adder 4 mult. 2 sub.	CSE and Strength Reduction

Table 1

pipelining can obscure the results, all circuits were implemented as pure combinatorial logic. The resulting combinatorial logic is an indirect indicator of the resulting scheduled and pipelined circuit area and performance. A smaller and faster combinatorial logic circuit indicates fewer operations to schedule and a smaller maximum path.

Table 1 summarizes the results from the five application benchmarks. As can be seen, the Synopsys behavioral logic compiler performs some optimizations, namely, common-sub-expression elimination, loop invariant code-motion, and to a limited extent, strength-reduction. However, the C compiler was able to do far more optimizations, including constant folding, zero-detection, loop invariant code-motion, common-sub-expression elimination, and a more intelligent strength-reduction. Note, however, that the actual speed and area numbers are benchmark, and technology dependent.

Overall the C to VHDL translation process effectively uses the GNU C/C++ compiler to optimize combinatorial circuits. Application 5 is a larger example (HAL³), that indicates that although the specific behavioral compiler used performed poorly on the overall code-style produced from the translation step, specific optimizations available in the GNU C/C++ compiler were not implemented, or were implemented poorly. These results are encouraging and future work in this area will allow the translation process to perform better on full-size logic examples.

5. Related Work

Previous work has used the C programming language for hardware design. Several academically sponsored projects implement a specialized C processor and produce XNF or suitably equivalent net-lists. These include the *Transmogripher C⁴* compiler from the University of Toronto, *Handel C⁵* from Oxford University, and *nlc⁶* from the École Polytechnic University of Sweden.

The C to VHDL and subsequent net-list compiler presented in this paper is unique from these three in that a standard stock compiler is first used to transform C expressions into MIPS assembly language. The majority of the work of the translation process thus becomes analyzing the assembly language statements to reconstruct the CDFG and produce VHDL code. In some fashion this is similar, to the work performed by Rahul Razdan and Michael Smith at Harvard University⁷. Their work concentrated on optimizing short, non-branching DFGs described originally as processor machine code. The DFGs were extracted by first profiling existing applications and searching for sequences of instructions that could be implemented in an FPGA attached to the core processor. The DFGs were then transformed into FPGA net-lists and using a new instruction to interface between the processor and FPGA, the DFG was executed inside the FPGA instead of the core processor. The work presented here goes far beyond simple DFGs by permitting arbitrary DFGs with variables, as well as conditional and fixed-loop based control-flow.

6 Conclusions and Future Work

Although this work demonstrated performance and area gains for small example circuits using a C to VHDL translation tool, for larger circuits the Synopsys tool failed to produce efficient logic from the style of VHDL code generated. Future work can address this with modifications to the VHDL code generator, or to the specific behavioral VHDL compiler used. Future work can expand the C subset that is permitted. Finally, future work may explore more in-depth optimizations. Although the GNU C/C++ compiler is an accepted standard for a relatively high-performance C compiler, it usually does not contain the most robust optimizer for a particular hardware platform. One reason for this is the long history of the GNU C/C++ compiler project. The optimizations implemented in the compiler are dated.

This paper introduces a tool that facilitates the translation of C source code into synthesizable VHDL code. Furthermore, this tool can work in conjunction with the optimizer inside a traditional C compiler to improve logic area and performance comparative to a native VHDL optimizing compiler. This work has demonstrated that the Synopsys behavioral compiler can implement more aggressive optimizations that can improve logic performance. If we presuppose that the Synopsys tools are the “state of the art” in the behavioral logic synthesis realm, then the industry as a whole can benefit from the study and implementation of traditional software compiler optimization techniques.

¹ GCC – The GNU C Compiler, <http://www.fsf.org/software/gcc/gcc.html>

² B.W. Kernighan, D.M. Ritchie, *The C Programming Language*, Prentice-Hall, New Jersey, 1978.

³ N.D. Dutt and C. Ramachandran, “Benchmarks for 1992 High-Level Synthesis workshop,” Tech. Rep. 92-107, Dept. of Information and Computer Science, U.C. Irvine, 1992.

⁴ D. Galloway, “*Transmogripher C Reference Manual*”, available via anonymous ftp from <ftp.eecg.toronto.edu> in /pub/software/tmcc.

⁵ Oxford Hardware Compilation Group, “*Handel C*”, available via <http://www.comlab.ox.ac.uk/oucl/users/ian.page/handel>.

⁶ C. Iseli, École Polytechnic University of Sweden, available via ftp from lslww.epfl.ch in /pub.

⁷ Rahul Razdan and Michael D. Smith. High Performance Microarchitectures with Hardware-Programmable Functional Units. *Proc 27th Annual IEEE/ACM Intl. Symp. On Microarchitecture*, pp. 172-180, November 1994.

ActiveOS: An Operating System Framework for Intelligent Memory Systems

Mark Oskin, Timothy Sherwood, Justin Hensley, Aneet Chopra, Lucian Lita and Frederic T. Chong
Department of Computer Science
University of California at Davis

Abstract

Current trends in DRAM memory chip fabrication have led many researchers to propose “intelligent memory” architectures that integrate microprocessors or logic with memory. Such architectures offer a potential solution to the growing communication bottleneck between conventional microprocessors and memory. Previous studies, however, have focused upon single-chip systems and have largely neglected off-chip communication in larger systems.

We introduce *ActiveOS*, an operating system which efficiently provides virtual memory for Active Pages [OCS98], a page-based intelligent memory architecture. We present results from multiprogrammed workloads running on a prototype operating system implemented on top of the SimpleScalar processor simulator. Our results show that paging and inter-chip communication can be scheduled to achieve high performance for applications that use Active Pages with minimal effects to applications that only use conventional pages. Overall, ActiveOS allows Active Pages to accelerate individual applications by up to 1000 percent and to accelerate total workload from 20 to 60 percent as long as physical memory can contain the working set of each individual application.

1 Introduction

Microprocessor performance continues to follow phenomenal growth curves which drive the computing industry. Unfortunately, memory systems are falling behind when “feeding” data to these processors. Processor-centric optimizations to bridge this *processor-memory gap* [WM95] [Wil95] include prefetching, speculation, out-of-order execution, and multithreading. Unfortunately, many of these approaches can lead to memory-bandwidth problems [BGK96].

Memory technology, however, is growing significantly denser. The Semiconductor Industry Association (SIA) roadmap [Sem94] projects mass production of 1-gigabit DRAM chips by the year 2001. If we devote half of the area of such a chip to logic, we expect the DRAM process to support approximately 32 million transistors. This density has led many researchers to propose intelligent memory systems that integrate processors or logic with DRAM. The increased bandwidth and lower latency of on-chip communication promises to address many aspects of the processor-memory gap [P⁺97] [KADP97].

These improvements, however, are limited to single-chip systems with limited memory requirements such as PDAs. We expect that the memory demands of most systems will scale with DRAM density and multiple memory chips will be required. Unfortunately, the issues of off-chip communication and virtual memory have largely gone unstudied in intelligent memory systems. We address these issues with *Active Pages*, a page-based architecture for intelligent memory. Active Pages consist of a superpage of data and a collection of functions which operate on that data. Unlike most other intelligent memory systems, Active Pages are explicitly designed to support applications that do not fit on a single memory chip. This support involves both the communication of data between chips and the movement of Active Pages to and from disk.

Previous work [OCS98] has shown that Active Pages are a flexible architecture that can lead to dramatic performance im-

provements (up to 1600X) in a single-process over conventional memory systems. This paper evaluates Active Pages in a multi-process environment. Active Pages fundamentally change memory systems from a uniform storage resource, to a non-uniform collection of storage and computational elements that require both management and scheduling. We introduce *ActiveOS*, a prototype operating system which illustrates the interaction between an OS and Active Pages. Our results show that while paging Active Pages to and from disk can be expensive, opportunities to overlap memory and processor computation increase dramatically in a multi-process environment. On mixed workloads of conventional and Active-Page applications, Active Page single-process performance translates well to a multi-process environment as long as memory pressure is moderate.

2 ActiveOS

ActiveOS is a prototype operating system which demonstrates mechanisms to manage Active Pages in a multi-process environment. These mechanisms fall into three key categories: process services, interpage communication, and virtual memory.

Our implementation uses a *processor mediated* approach to satisfying inter-page memory references. Each Active Page explicitly distinguishes between local and non-local references by checking if a virtual address is between its starting address, kept in a base register, and the starting address plus the Active-Page size. When an Active Page reaches a non-local reference it raises an interrupt and blocks. The interrupt causes the processor to obtain the memory request through a memory-mapped read and executes an OS handler. The handler consults a per-process table which will determine if the Active Page containing the reference is in physical memory. If so, the page is located and the request is satisfied. If not, a page fault occurs and the page is brought in from disk. In practice, a large number of non-local references from different pages can be satisfied upon a single interrupt, substantially reducing overhead.

When insufficient physical memory is available to execute all processes in the system, memory must be paged to swap space. We define the maximum required swap space to execute a set of processes as the *memory pressure*. As memory pressure increases, Active Pages will need to be paged out to disk. This implies that the data, the functions, and any active state must be written to disk. The choice of which pages to swap in and out is critical to both the correctness and performance of Active Pages. If the processor or an Active Page in memory references a page on disk, a page fault occurs. If no reference occurs, however, a page on disk may still need to be swapped in to complete a computation for a page group. Active Pages can only make computational progress if they are in physical memory. Managing Active Pages is more than traditional virtual memory management [Den70]. It is also scheduling of computation.

ActiveOS currently implements a round-robin memory page scheduler for Active Pages residing in swap space. Using this memory resource scheduler, we evaluated several page replacement policies in ActiveOS. LRU and Clock-Work are representative of commonly used policies in conventional operating systems. Random is included for comparison. SPLRU and AP-SPLRU are modifications to LRU that exploit super-page and Active Page information.

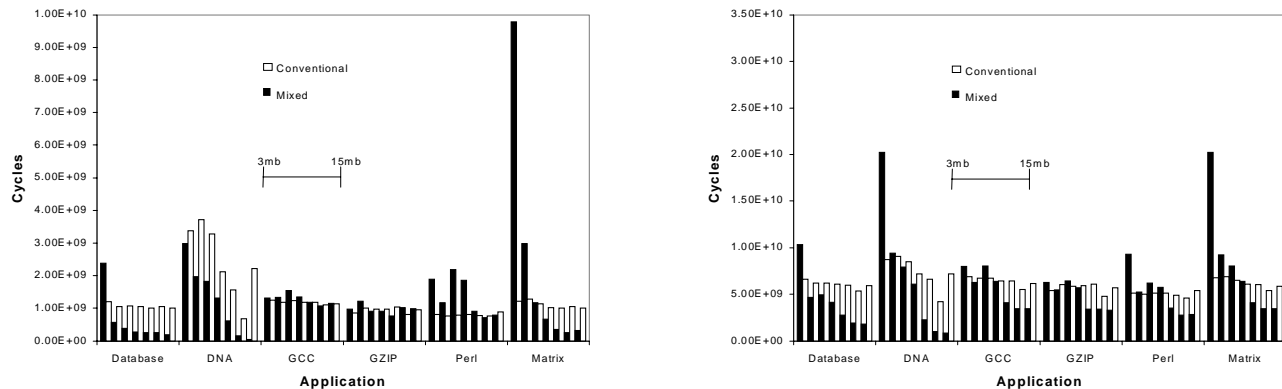


Figure 1: for LRU page replacement. Process time is given on the left and wall-clock time on the right (bars are for 3, 5, 7, 9, 11, 13, and 15 Mbytes of physical memory).

3 Results

In this section, we present our results from executing our multiprogrammed workload using the LRU page replacement algorithm. Our results indicate that Active-Page computations tolerate moderate memory pressure well, but performance degrades as physical memory size approaches the working-set size. This degradation tends to happen at slightly higher rates for Active-Page computations than conventional computations.

Figure 1 plots application time versus physical memory size for each application executing with the LRU page replacement policy. The performance of the Active Page *DNA* application is most affected by moderate reductions in physical memory size. Also note a pathological case in which the interaction of LRU with the access patterns of *DNA* cause 13 Mbytes to perform better than 15 Mbytes of memory.

Further note that the Active-Page version of *matrix* performs extremely poorly when physical memory is very low. At 3 Mbytes, *matrix* performs an order of magnitude worse than at 7 Mbytes. This occurs because of frequent and repeated access to each Active Page. Although each page is touched repeatedly, the quantity of data used from them is very small. Under extreme memory pressure, intense swapping occurs with very little access to data by the processor.

Referring to the wall clock times in Figure 1, we can see that LRU does a good job of distributing the effects of memory pressure among applications for conventional processor only applications. However, it overly penalizes Active Page applications under heavy memory pressures.

4 Conclusions and Future Work

Although ActiveOS has shown promising results in virtualizing Active Pages for a multi-process environment, many issues remain. A major effort is underway to automate the application partitioning process. The number of applications in the multiprogram workload needs to be increased, and the working set size of each application also needs to be expanded. Furthermore, virtual memory paging and memory resource scheduling techniques need to be investigated in more detail. Operating system support for Active Page memories with hardware communication facilities will need to be addressed.

Currently, ActiveOS implements only limited Active-Page aware paging policies with the SPLRU page replacement algorithm. With more applications available a better understanding of their paging characteristics can be achieved. Future work will focus on

characterizing Active-Page program behavior in more detail, and then designing efficient paging policies to support both Active-Page and conventional applications. Furthermore, work will address active Active-Page priorities, and memory resource priority scheduling. All current paging policies rely upon access information generated by the processor. However, Active-Pages degrade the quality of this information because activity can be occurring solely inside the memory system. This activity is currently not tracked by the current processor access statistics.

ActiveOS demonstrates that intelligent memory systems can be virtualized to achieve high multiprogrammed performance. Active Pages provide a page-based intelligent memory architecture that makes this virtualization possible. In fact, multiprogramming increases the opportunities for overlapping processor and memory computation. The fact that memory performs computation, however, means that some reasonable amount of physical memory must be available. Consequently, our results show high performance from zero to moderate memory pressures, but performance degrades significantly as available memory falls below the working set size of any one Active-Page application.

References

- [BGK96] D. Burger, J. Goodman, and A. Kagi. Quantifying memory bandwidth limitations in future microprocessors. In *International Symposium on Computer Architecture*, Philadelphia, Pennsylvania, May 1996. ACM.
- [Den70] Peter J. Denning. Virtual memory. *Computing Surveys*, 2(3):153–189, September 1970.
- [KADP97] Kimberly Keeton, Remzi Arpacı-Dusseau, and David A. Patterson. IRAM and SmartSIMM: Overcoming the I/O bus bottleneck. In *Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, Denver, Colorado, June 1997.
- [OCS98] Mark Oskin, Frederic T. Chong, and Timothy Sherwood. Active pages: A computation model for intelligent memory. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA '98)*, Barcelona, Spain, 1998. To Appear.
- [P⁺97] D. Patterson et al. The case for intelligent RAM: IRAM. *IEEE Micro*, April 1997.
- [Sem94] Semiconductor Industry Association. The national technology roadmap for semiconductors. <http://www.sematech.org/public/roadmap/>, 1994.
- [Wil95] M. Wilkes. The memory wall and the CMOS end-point. *Computer Architecture News*, 23(4), September 1995.
- [WM95] W. Wulf and S. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1), March 1995.

DEFENDER: A Device to Protect Networked Computers

Nick Puketza*

puketza@cs.ucdavis.edu

Department of Computer Science

University of California at Davis

July 21, 1998

Computer networks are already important components in the infrastructure of the United States. In the future, they will become even more vital, as new cutting-edge applications for these networks are developed for the military, government, business, medicine, and education. Such applications will be reliable only if they are secure against attacks. In response to this need for security, intrusion detection systems (IDSs) and firewalls have emerged as important computer security tools in recent years. We propose to develop a powerful new device called the DEFENDER. This *DE*vice *For* *EFF*ective *NE*twork *DE*tECTION and *R*ESPONSE combines the capabilities of an IDS and a firewall to protect its host computer against network attacks. We further propose to create a device called the Safe Modem, a modem enhanced with DEFENDER technology to protect a computer from attacks over a phone line.

Like an IDS, the DEFENDER will search for *attack signatures*, which are the identifying characteristics of particular attacks. The DEFENDER will also compare incoming network traffic to a statistical profile of normal traffic to detect conditions that are anomalous and thus possibly suspicious. Like a firewall, the DEFENDER will block certain packets from reaching the protected host. The DEFENDER will also be able to *respond* to attacks. The DEFENDER's packet-filtering mechanism will be dynamic. When an attack is detected, the DEFENDER can start to block packets that are related to that attack, and in some cases the DEFENDER will initiate a tracing mechanism to pinpoint the attack source.

The DEFENDER will provide strong protection against a particular class of widely-used attacks. These are attacks that are triggered by abnormal network packets, packets with characteristics that would never occur during normal operation. Often, these packets cause a computer's operating system to

“crash”, so that the computer must be restarted. Several such attacks have been documented, each using a different kind of abnormal packet to disrupt computers. For each of these attacks, we will create an attack signature that specifies the identifying characteristics of the abnormal packet that is the root of the attack. For example, Figure 1 displays signatures for several well-known attacks. The signatures will be stored in a database in the DEFENDER, and the database will be designed to easily accommodate more signatures as they are developed. The DEFENDER will compare each incoming packet to this database of signatures, and issue a warning signal for each packet that matches a signature.

Many attacks, though, do not involve abnormal packets. For example, in a *message-flooding* attack, the attacker simply sends a flood of messages to the target, thereby consuming the target's resources so that its performance for legitimate users is degraded. The DEFENDER can detect these attacks too, by constantly measuring various parameters of incoming traffic and comparing those measurements to a statistical profile of normal behavior. For example, one well-known attack is the *SYN flooding* attack [SKK⁺97], in which the attacker sends several TCP SYN packets to a target TCP port, but does not send the subsequent packets necessary to establish TCP connections. The result is several “half-open connections,” which prevent the operating system from accepting legitimate connection requests for that port. The DEFENDER could detect the attack by recognizing that the arrival rate of TCP packets to a particular port is much higher than normal.

In addition to detecting attacks, the DEFENDER can prevent and respond to attacks. Like a firewall, the DEFENDER can behave as a filter, blocking certain types of packets intended for the host computer, and thereby limiting the number of possible attacks. The DEFENDER can initiate packet-filtering as a response mechanism. For example, when it recognizes a flooding attack, the DEFENDER can start to discard

*Nick Puketza is supported in part by ARPA. Address: Dept. of Computer Science, UC Davis, One Shields Avenue, Davis CA 95616

Attack Name	Signature
Land	(source IP address = dest IP address) and (source port = dest port)
Pepsi	(source port = echo port) and (dest port = char gen port)
Ping o' Death	total no. of bytes (as specified in the IP header) > maximum acceptable value
Smurf	(packet type = ICMP echo request) and (dest = a broadcast address)

Figure 1: Attack Signatures for Recent Network Attacks.

packets of the type associated with the flood. The DEFENDER can also respond to a flooding attack by communicating with other DEFENDERS to identify the source of the attack. Each DEFENDER on the path between the attack source and target should notice a surge in network traffic. Thus, the DEFENDER on the target can initiate a chain of communication to discover the address of the probable attack source.

After developing the DEFENDER, we plan to adapt it and combine it with a modem to create a device called the Safe Modem. The Safe Modem will use DEFENDER technology to detect and respond to attempted attacks over phone lines. The Safe Modem will not only protect home computers against attack, but it will also prevent “back-door” attacks on large-scale corporate networks, attacks that attempt to circumvent firewalls by using a modem as the network entry point.

The DEFENDER will have some significant advantages over current IDSs. Many intrusion detection systems analyze only the audit records generated by their host computers, and they do not inspect network packets directly. The problem with this approach is that an attacker’s packet may disable the computer before any audit records are created. The DEFENDER will detect an “attack packet” before the packet reaches the host computer, so it can block the packet or at least signal its arrival.

Some intrusion detection systems *do* monitor network packets directly. For example, Network Security Monitor (NSM) [HDL⁺90] searches packets for specific strings (in ASCII code) that an attacker might use (e.g., the name of a widely distributed attack script). NSM also records statistics for each network connection, such as the number of times a similar connection between the same two computers has occurred in the last month. The DEFENDER will take NSM’s approach further. Instead of searching packets only for strings, the DEFENDER will check each packet against a database of attack signatures. The signature mechanism will be flexible so that the DEFENDER can be instructed to check several different features of a packet (e.g., all the different fields in a packet’s IP header). The DEFENDER will also keep more exten-

sive statistics related to incoming packets than NSM, and therefore the DEFENDER will be able to detect a wider variety of anomalous conditions.

As described earlier, the DEFENDER will also have an advantage over many existing firewalls: a packet-filtering mechanism that can change its filtering rules when an attack is detected. DEFENDER’s ability to trace a flooding attack to its source is another response mechanism that most firewalls and IDSs cannot provide.

We do not intend for the DEFENDER to replace firewalls and IDSs. Both tools still have important roles and some functions that will not be provided by the DEFENDER (e.g., analysis of audit records on the protected host). However, the DEFENDER will provide excellent protection for hosts against network attacks, and therefore it is a vital and necessary complement to existing security tools.

References

- [HDL⁺90] L. T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. “A Network Security Monitor”. In *Proc. 1990 Symposium on Research in Security and Privacy*, pages 296–304, Oakland, CA, May 1990.
- [SKK⁺97] C.L. Schuba, I.V. Krsul, M.G. Kuhn, E.H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, CA, May 1997.

Light-Trees: Exploiting Optical Multicasting to Improve the Performance of Unicast and Broadcast Traffic in Wavelength-Routed Optical Networks*

Laxman Sahasrabudh[†] and Biswanath Mukherjee

Department of Computer Science, University of California, Davis, CA 95616

sahasrab@cs.ucdavis.edu, mukherje@cs.ucdavis.edu

[†]Correspondence Author (Tel: (530) 752-5129; Fax: (530) 752-4767)

July 24, 1998

We introduce the concept of a *light-tree* in a wavelength-routed optical network. A light-tree is a point-to-multipoint generalization of a *lightpath*. A lightpath is a point-to-point all-optical wavelength channel connecting a transmitter at a source node to a receiver at a destination node [1]. Using wavelength-routing switches (WRSs) at intermediate nodes and via appropriate routing and wavelength assignment, a lightpath can create logical (or virtual) neighbors out of nodes that are geographically far apart in the network. Using lightpath communication, a significantly large number of lightpaths may be set up on the network in order to embed a logical (or virtual) topology. Now, a lightpath carries not only the direct traffic between the nodes it interconnects but also traffic from nodes upstream of the source (including the source) to nodes downstream of the destination (including the destination). A major objective of lightpath communication is to reduce the number of hops (or lightpaths) a packet has to traverse because this reduction can, in turn, significantly improve the network's throughput [2].

Under lightpath communication, the network employs an equal number of transmitters and receivers because each lightpath operates on a point-to-point basis [1]. However, this approach may not be able to fully utilize all of the wavelengths on all of the fiber links in the network; also, it may not be able to fully exploit all of the switching capability of each WRS [3].

Thus, we extend the lightpath concept by incorporating an optical multicasting capability. That is, if there is a connection from transmitter A to receiver B on a certain wavelength and there exist

free resources (fiber link and the same wavelength) to direct A's transmission to some other receivers C and D as well, then why not do it? Now, A will have three logical downstream neighbors (B, C, and D) instead of one earlier (B). Thus, the logical connectivity of the network is increased and the hop distance is decreased. We refer to such a point-to-multipoint extension of a lightpath as a *light-tree*. Since a light-tree is a generalization of a lightpath, the set of light-tree-based virtual topologies is a *superset* of the set of lightpath-based virtual topologies. Hence, an "optimum" light-tree-based virtual topology is *guaranteed* to perform better than an "optimum" lightpath-based virtual topology.

Note that light-trees can not only provide improved performance for unicast traffic (as outlined above), but they can naturally better support multicast traffic and broadcast traffic because of their inherent point-to-multipoint nature. In this study, we shall concentrate on the application and advantages of unicast and broadcast traffic only; the application to multicast traffic is an on-going study and will be reported at a later date.

It should also be noted that optical multicasting (which is used to implement a light-tree) has some improved characteristics over electronic multicasting since "splitting light" is *conceptually* easier than copying a packet in electronic buffer. What are the corresponding issues in designing WDM optical switches that can support multicasting; how do we design algorithms for setting up the corresponding "light-trees"; and how do we quantify the corresponding performance benefits? Our study will attempt to answer these questions. A related issue which we will not analyze here is how do we compensate for power penalties in signal splitting.

In summary, a light-tree is a point-to-multipoint

*This work has been supported in parts by the National Science Foundation (NSF) under Grant Nos. NCR-9508238 and ANI-9805285

all-optical channel which may span multiple fiber links. Hence, a light-tree enables “single-hop” communication between a “source” node and a *set* of “destination” nodes; thus, a light-tree-based virtual topology can significantly *reduce the hop distance*, thereby *increasing the network throughput*. Figure 1 shows a light-tree (thick lines) which connects node *UT* to nodes *TX*, *NE*, and *IL*. Thus, an optical signal transmitted by node *UT* travels down the light-tree till it reaches node *CO* where it is “split” by an “optical-splitter” into two *identical* copies. One copy of the optical signal is *routed* to node *TX*, where it is terminated at a receiver. The other copy of the optical signal is routed towards node *NE*, where it is again split into two identical copies. At node *NE*, one copy of the optical signal is terminated at a receiver, while the other copy is routed towards node *IL*¹. Finally, a copy of the optical signal reaches node *IL*, where it is terminated at a receiver. Thus, the “virtual topology” induced by this light-tree consists of three logical links as shown in Fig. 2. (The link labels in Fig. 2 indicate the fraction of the source node’s traffic that is destined for the link destination. The sum of these labels should be no larger than unity.)

We explore *new* architectures for the next generation of networks employing light-trees for wide-area (nation-wide) coverage. We examine an “optical” wide-area wavelength-division multiplexing (WDM) network which utilizes multicast-capable optical switches at routing nodes, so that an “highly-connected” arbitrary virtual topology can be embedded on a given physical fiber network.

We formulate the light-tree-based virtual topology design problem as an optimization problem with one of two possible objective functions: for a given traffic matrix, (i) minimize the network-wide average packet hop distance, or (ii) minimize the total number of transceivers in the network. We consider two types of traffic: (i) unicast and (ii) broadcast. For broadcast traffic, we only consider the minimization of the total number of transceivers in the network. We demonstrate that: (i) an optimum “light-tree”-based virtual topology has a lower value of average packet hop distance than the average packet hop distance of an optimum “lightpath”-based virtual topology, and (ii) an optimum “light-tree”-based virtual topology requires fewer opto-electronic compo-

¹This operation at node *NE* can also be performed by a “drop-and-continue” optical device.

nents than the number of opto-electronic components required by an optimum “lightpath”-based virtual topology.

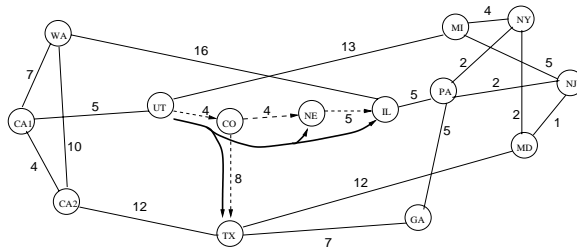


Figure 1: NSFNET backbone topology. (Link labels correspond to propagation delays.)

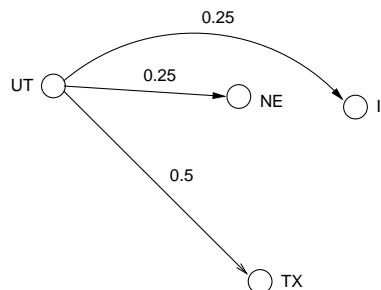


Figure 2: Virtual links induced by the light-tree consisting of nodes *UT*, *NE*, *TX*, and *IL*.

References

- [1] I. Chlamtac, A. Ganz, and G. Karmi, “Light-path communications: An approach to high-bandwidth optical WAN’s,” *IEEE Transactions on Communications*, vol. 40, pp. 1171–1182, July 1992.
- [2] B. Mukherjee, D. Banerjee, S. Ramamurthy, and A. Mukherjee, “Some principles for designing a wide-area optical network,” *IEEE/ACM Transactions on Networking*, vol. 4, pp. 684–696, Oct. 1996.
- [3] D. Banerjee and B. Mukherjee, “Wavelength-routed optical networks: Linear formulation, resource budgeting tradeoffs, and a reconfiguration study,” in *Proceedings, IEEE INFOCOM ’97*, (Kobe, Japan), Apr. 1997.

Texture Planes: Real-Time Volume Rendering Using Hardware Texture Mapping and Alpha Blending

Issac J. Trotts*

trotts@cs.ucdavis.edu

Department of Computer Science

University of California at Davis

Bernd Hamann†

hamann@cs.ucdavis.edu

Department of Computer Science

University of California at Davis

1 Algorithm Description

We present an efficient algorithm, called *Texture Planes*, for directly rendering 3D volumetric data on workstations with hardware for alpha blending (transparency) and 2D texture mapping. Our implementation runs about two orders of magnitude faster than a naive conventional volume renderer and generates images of comparable quality.

Conventional volume rendering techniques proceed by casting a ray through each pixel of the display, integrating several lighting calculations along each ray to obtain the color for the pixel. Our algorithm approximates this computation by rendering a stack of rectangles which slice through the volume. We apply a texture map to each of the rectangles based on the scalar values it touches in the volume. We assign each texel of the texture a (possibly) different color and transparency. We use the density of the volume at the location of of texel to find the its lighting properties in a look-up table and use the gradient to determine the opacity as well as the normal for the lighting computation.

*Web: <http://graphics.cs.ucdavis.edu/~trotts>

†Web: <http://graphics.cs.ucdavis.edu/people/hamann>

‡This work was supported by various grants and contracts awarded to the University of California, Davis, including the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center under the NRA2-36832(TLL) Program, and the Lawrence Livermore National Laboratory under contract W-7405-ENG-48 (B335358). We thank all members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, for their help.

To ensure the quality of the renderings from all viewing directions, we compute a stack of slices for each of the x , y , and z axes. To render, we select the stack whose slices are most nearly perpendicular to the vector from the camera point to the center of the volume.

2 Discussion

2.1 Hybrid Images with Volume Rendering and Z-Buffer Rendering

For some applications it is useful to render scenes containing both volumetric data and polygonal geometry. Levoy describes a way to do this with a conventional volume renderer in [Lev88], either by ray tracing the polygons or by sampling the polygons to incorporate them into the volume. However, ray tracing is usually a time-consuming process and sampling the polygons is likely to introduce spurious bumps on their surfaces (from voxel aliasing). Sampling the polygons into the volume might also complicate the classification of tissue types.

In contrast, volume renderings produced with our method can easily and accurately be displayed in a scene with polygon-based geometry, using a standard 3D graphics library such as *Open GL* or *Open Inventor*. Polygonal models passing through the volume are rendered correctly.

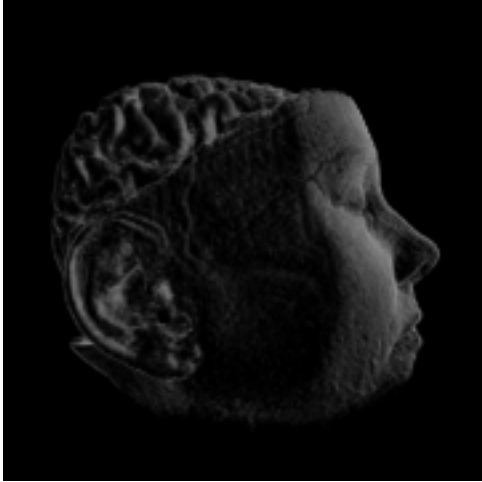


Figure 1: A rendering generated by *Texture Planes* in 1.136 seconds on an SGI Onyx 2 with 512 MB RAM, an Infinite Reality graphics engine and four R10000 processors. Only one processor was used. The computation of the slices and generation of an Open GL display list took 120 seconds.

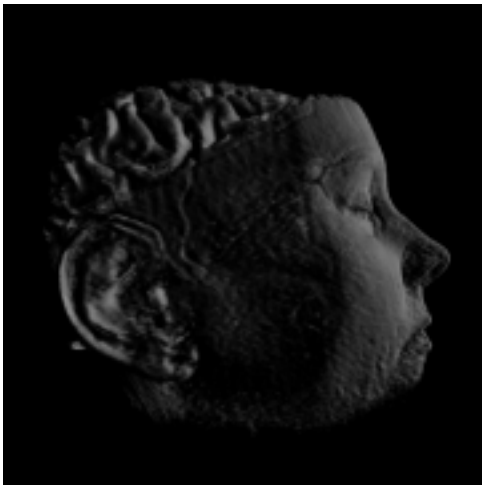


Figure 2: A rendering generated with a naive conventional volume renderer in 302 seconds on the same Onyx as in Figure 1.

2.2 VRML Implementation

Another advantage of our technique over conventional volume rendering is that it can be easily implemented in the Virtual Reality Modeling Language, which makes possible the publication of interactive volume renderings to the web. It should also be able to seamlessly integrate these volume renderings into existing VRML worlds without any additional coding.

2.3 Non-Cartesian Grids

Our implementation currently handles only Cartesian grids (rectangular volumes with rectangular voxels). However, our algorithm could also be used for interactive rendering of volumes containing tetrahedra, hexahedra, and other voxel types. We would do this by resampling the irregular grid into a Cartesian grid, and then applying the Texture Planes algorithm as usual.

3 Conclusion

We have presented a simple algorithm that exploits hardware texture mapping and alpha blending to produce renderings of scalar fields at interactive frame rates in scenes with arbitrary polygon-based geometry. Our algorithm generates images of quality nearly equal to that of conventional volume rendering in a small fraction of the time. An investigation of methods for reducing the time spent preparing slices is slated for future research.

References

- [Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, pages 29–37, July 1988.
- [Max95] Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, pages 99–108, June 1995.

Photonic Slot Routing in All-Optical Packet-Switched WDM Mesh Networks *

Hui Zang, and Jason P. Jue[†]
 zang@cs.ucdavis.edu, juej@cs.ucdavis.edu
 Department of Computer Science

University of California at Davis

	slot header			ϵ padding
λ_1				
λ_2	used	sub-header	data	
λ_3	used	sub-header	data	
λ_4	used	sub-header	data	
λ_5	used	sub-header	data	

Figure 1: Format of a photonic slot.

Wavelength-division multiplexing (WDM) has been rapidly gaining acceptance as a means to handle the ever-increasing bandwidth demands of network users [1]. In addition to providing huge amounts of bandwidth, all-optical WDM networks also offer the benefit of high-speed data transmissions without electronic conversions at intermediate nodes. By transmitting the signal entirely in the optical domain, data transparency can be achieved. One such type of all-optical network is the wavelength-routed WDM network, in which all-optical lightpaths are set up on specific wavelengths between pairs of nodes. Some of the major challenges in designing wavelength-routed WDM networks include the complexity and scalability issues which arise from the need to demultiplex and individually route each wavelength at a node.

An emerging alternative to the wavelength-routed WDM network is a WDM network based on optical packet switching. One recently proposed approach to optical packet switching is Photonic Slot Routing (PSR) [2]. Although PSR has been studied in bus and ring networks, it has not been considered in mesh networks until now. Also, no analytical model has previously been developed to measure the performance of a PSR network. In this study, we consider the application of PSR to an arbitrary mesh network.

In PSR, time is slotted, and data is transmitted in the form of photonic slots (Fig. 1) which are fixed in length and span across all wavelengths in the network. Each wavelength in the photonic slot may contain a single packet, and all packets in the photonic slot are destined to the same node. By requiring the packets to have the same destination, the photonic slot may be routed as a single integrated unit without the need for demultiplexing individual wavelengths. Thus, wavelength-insensitive components may be used at each node, resulting in less complexity, faster routing, and lower network cost [3].

The photonic slot header, which may be carried on a separate wavelength (as shown, for example, in Fig. 1), contains information such as the destination of the slot, and which wavelengths in the slot are occupied by packets. The destination information is used to determine/configure the switch setting at the node so that the slot can be appropriately routed towards its destination using a standard protocol (e.g., shortest-path routing). The slot occupancy information determines the extra allowable packet transmissions for each outgoing photonic slot.

On a given output fiber link, the node may insert

*This work has been supported in part by NSF Grant Nos. NCR-95-08239 and ANI-98-05285.

[†]Address: EU2 Rm 2243, Dept. of Computer Science, UC Davis, Davis CA 95616

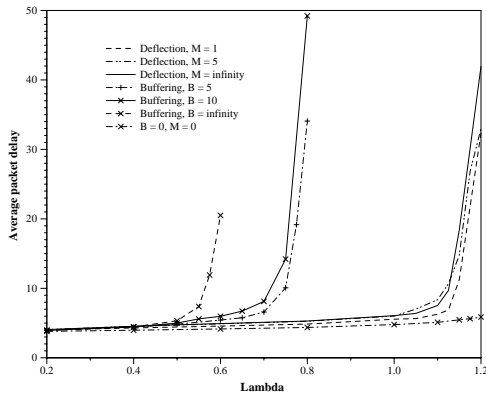


Figure 2: Average delay versus packet arrival rate (λ) for different policies to resolve contention.

packets into existing photonic slots which are headed for the same destination, or the node may transmit newly created photonic slots if no other slots are contending for the link.

In a mesh network, nodes may have multiple input and output ports, resulting in a higher degree of contention than in ring-based networks. Different approaches for resolving contentions when they occur are studied. A slot that does not win the contention may be handled in different ways:

- in a Buffering scheme, the slot may be buffered in an optical buffer of size B ,
- in a Deflection scheme, it may be deflected to another outgoing link with the hope that the slot will eventually make it to the destination. Usually a slot can be deflected up to M times before being dropped.

We compare the average packet delay when two schemes with varying parameters B and M are used in simulation and the results are shown in Fig. 2.

We investigate various approaches for transmitting packets and reducing the amount of contention. Once a node creates a new slot, it should decide whether or not to transmit packets within this slot, and which destination should be assigned to this slot. Two

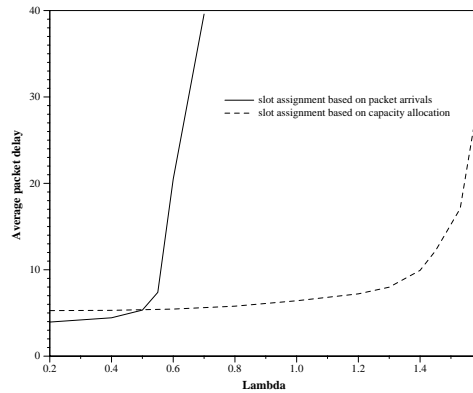


Figure 3: Average delay versus packet arrival rate (λ) for different slot-assignment schemes.

types of slot-assignment policies are considered: slot-assignment algorithm based on packet arrivals and slot-assignment algorithm based on capacity allocation. Two policies are both simulated and the results is shown in Fig. 3. We provide a detailed analytical model for measuring the performance of a proposed network with the latter policy. We validate the analysis through simulation and present numerical examples. We also outline approaches for dealing with issues such as synchronization.

References

- [1] B. Mukherjee, *Optical Communication Networks*, McGraw-Hill, New York, 1997.
- [2] I. Chlamtac, V. Elek, and A. Fumagalli, "A fair slot routing solution for scalability in all-optical packet-switched networks," *Journal of High Speed Networks*, vol. 6, pp. 181-196, 1997.
- [3] I. Chlamtac, V. Elek, A. Fumagalli, and C. Szabo, "Scalable WDM network architecture based on photonic slot routing and switched delay lines," *Proceedings, IEEE INFOCOM '97*, Kobe, Japan, vol. 2, pp. 769-776, April 1997.