EE 382C EMBEDDED SOFTWARE SYSTEMS

Literature Survey Report

Characterization of Embedded Workloads

Ajay Joshi

March 30, 2004

ABSTRACT

Security applications are a class of emerging workloads that will play a central role in determining the performance of next generation embedded microprocessors. The objective of this research work is to understand the inherent workload characteristics of security applications, and analyze their impact on microprocessor architecture design. The outcome of this work will provide an insight into the performance bottlenecks in existing architectures that challenge security applications, and enable us to propose architectural enhancements required to boost the performance of embedded and digital processors running security applications. In this report we summarize the popular methodologies for characterizing workloads, survey classic studies that have been performed to measure the degree of parallelism in applications, and present a proposal to study the micro-architecture independent characteristics of security applications.

I. Introduction

The phenomenal growth in the Information Technology industry has resulted in the emergence of several new computer applications. The computer systems that run these applications were designed before the advent of these workloads, and their

architectural features may not match the application characteristics; possibly resulting in loss of performance. In order to ensure that microprocessors and computer systems of tomorrow deliver highest performance, it is extremely important for computer architects to identify these emerging workloads and understand their idiosyncrasies that challenge the existing architectures.

In this work, we identify security applications as a key emerging embedded workload that will play a central role in determining the performance of tomorrow's embedded microprocessors. The proposed research comprises of understanding the characteristics of these applications, accounting for the cycles spent during their execution, and identifying possible sources for the loss of performance.

This report is organized as follows: Section II explains the motivation behind characterizing the behavior of security applications. Section III describes the previous research in workload characterization, and techniques that have been used to measure parallelism in an application. Section IV outlines the objectives of this project and proposes a methodology for carrying out the research work. Section V summarizes the key findings and draws conclusions from the literature review that influence the methodology in our research work.

II. SECURITY APPLICATIONS – AN EMERGING WORKLOAD

Until a few years ago, general-purpose processors and computers have been the driving force in shaping the digital economy. However, recently we have seen the proliferation of embedded systems in our daily lives through telecommunications, consumer, automotive, and office automation applications. Many of these embedded applications like cell phones, text message pagers, wireless hand held devices, pay-TV,

DSL modems, audio-video consumer products, telephony systems, and network routers, heavily rely on security mechanisms. Data security will play a central role in the design of these embedded systems. Security applications use cryptography algorithms to encrypt and decrypt messages sent over an insecure medium. It is therefore important to understand the characteristics of cryptography algorithms in order to design efficient next generation embedded processor and digital signal processor architectures.

II. PREVIOUS WORK

A. WORKLOAD CHARACTERIZATION APPROACHES

Several early studies in characterizing program behavior are available in literature [3][4]. Workload characteristics have been measured at three different levels: source code, micro-architecture independent, and system level [7]. Source code level characterization yields information about inherent nature of the application. However, this technique has not gained much popularity because of the difficulty in standardizing the measured characteristics across different programming language implementations. Micro-architecture independent attributes, although not biased by a particular machine implementation, are influenced by the programming language, Instruction Set Architecture (ISA), compiler, and operating system. System level characteristics are micro-architecture dependent, and are widely used because they provide an insight into the match between an application and architecture.

B. WORKLOADS STUDIED

During the seventies, eighties, and early nineties, researchers mainly focused on studying the characteristics of scientific and high-performance applications written in Fortran. In the last ten years, general-purpose, e-commerce, web, graphics, and

multimedia workloads, developed in C, C++, Java, and other web technologies have gained popularity. It is only during the last few years, due the growing market segment of embedded systems, embedded workloads have been the focus of study of computer architecture researchers. However, characterization of embedded workloads has mainly concentrated on network, telecommunication, signal processing, and multimedia applications.

C. MEASURING PARALLELISM IN PROGRAMS

Parallelism in a program can be classified into three different types: Instruction Level Parallelism (ILP), Data Level Parallelism (DLP), and Thread Level Parallelism (TLP) [6]. These workload characteristics respectively measure the number of instructions, data, and threads in the program that can be concurrently executed - assuming that infinite machine resources are available. Understanding the parallelism in programs is an important in selecting the type of computer system (uniprocessor or multiprocessor), and the microprocessor architecture philosophy (superscalar, VLIW, Vector etc.). The possible limits of parallelism have been investigated for almost three decades. Numerous experiments have been performed that yield widely varying results on the limits of parallelism; primarily due to the differences in machine models assumed [12]. In this section we review three such studies that used different approaches and methodologies.

Wall's [2] study on the limits of ILP is considered to be the most thorough limit study to date, accounting for speculative execution, memory disambiguation, and other factors. Wall used the instruction trace of programs from the Standard Performance Evaluation Council (SPEC) benchmark suite and other standard benchmarks, to compute

the ILP speedup for various scenarios, generating a lot of valuable data, but no simple answers. Speedups ranging from 7-60 were observed on these sample programs, but Wall himself is much more pessimistic and sees an average parallelism of only 5-7.

Arvind, Culler, and Maa [1] used dataflow program graphs to develop a methodology for quantifying the parallelism in real programs. This technique allows the programs to be studied in full detail, without biasing their behavior by implementation constraints, and hence draws a clear distinction between the parallelism inherent in a program and that achieved with a specific implementation. The results of this study bring us close to understanding the characteristics of a program, and analyze whether a program has enough parallelism to benefit from an n-way machine.

Lam and Wilson [10] measured the parallelism of programs in the SPEC benchmark suite on seven abstract machines that are a combination of three techniques – control dependence analysis, speculation with branch prediction, and multiple flows of control. The results suggest that the constraint imposed by control flow is the bottleneck that limits parallelism in programs. They observe that local regions of code have limited parallelism, and control dependence analysis is useful in extracting global parallelism from different parts of a program.

From this survey we can conclude that the real limit on program parallelism is a combination of inherent parallelism in the algorithm, and the parallelism that is further exploited by speculative micro-architectures and aggressive compilers. Although it is possible to place an upper bound on the former, we can only place a lower limit on the latter.

III. THE PROJECT

A. OBJECTIVES

The problem being addressed in this research is to characterize the behavior of security workloads by measuring their micro-architecture independent attributes. This approach will enable us to understand the intrinsic behavior of security applications without being influenced by a specific micro-architectural implementation.

The deliverables of this project are: quantification of the inherent parallelism in security applications, their micro-architecture independent characteristics, and an analysis of the impact of these characteristics on microprocessor architecture design.

The contributions of this study are: pin-pointing bottlenecks in existing architectures that challenge these applications, and propose architectural enhancements to embedded and digital signal processors to boost their performance on these classes of applications. This study will also lead to a better understanding of the nature and intrinsic characteristics of security applications that will help in effectively interpreting performance measurements and simulation results.

B. METHODOLOGY

In this section we outline the methodology by specifying the selection of representative workloads, approach used to quantify the parallelism, and the simulation environment used to measure the micro-architecture independent characteristics.

Cryptography algorithms are at the heart of all security applications. These algorithms use mathematics to encrypt and decrypt messages that are transmitted over in insecure communication medium. Private Key (symmetric cipher) and Public Key (asymmetric cipher) are the two most popular varieties of encryption algorithms. Public

Key algorithms are popular because they eliminate the need to transmit a key over an insecure network by the using different keys for ciphering and deciphering the data. We therefore select the following symmetric key ciphers as our representative workloads: Pretty Good Privacy (PGP) sign and verify, Rijndael, Blowfish Encrypt and Decrypt, and Secure Hash Algorithm (SHA).

In order to study the inherent parallelism that exists in the program we will use the methodology suggested by Arvind, Culler, and Maa [1]. This enables us to analyze the program parallelism without restricting to an implementation in a particular programming language. The algorithms mentioned above will be modeled using Synchronous Data Flow (SDF) or Boolean Dataflow (BDF) models of computation in Ptolemy. These programs will then be simulated to understand the sequential nature that is inherent in the program.

The second part of this study involves studying the micro-architecture independent characteristics of the cryptography algorithms. Advanced RISC Microprocessor (ARM) is one of the most popular embedded processor core used in embedded applications. We therefore choose *Strong-ARM* cycle accurate simulator environment to measure the micro-architecture independent characteristics.

IV. SUMMARY & CONCLUSION

We have identified security applications as an emerging workload and propose to study the micro-architecture independent characteristics of these classes of applications. The outcomes of this study will involve identifying the bottlenecks in existing architectures that challenge these applications, and a propose techniques for architectural

enhancements required in embedded and digital signal processors to eliminate these bottlenecks.

The previous work in workload characterization has mainly concentrated on scientific, web, commercial, e-commerce, and graphics workloads in the general-purpose processor domain, and networking, telecommunication, and signal processing workloads in the embedded processor domain. There has been very little work in characterizing the behavior of security applications.

Researchers have used three different approaches to quantify the parallelism in applications: dataflow graphs, constructing a dependency flow graph using instruction traces, and measuring speedup on abstract machines. We choose the dataflow graph technique for the purpose of our study because the outcome will be independent of particular limitations and features of programming languages.

REFERENCES

[1] Arvind, D. Culler, and G. Maa, "Accessing the benefits of Fine-Grain Parallelism in Dataflow Programs", *The International Journal of Supercomputer Applications*, 2(3), November 1988.

[2] D. Wall, "Limits of instruction level parallelism", In *Proceedings of ASPLOS-4*, volume 26, pages 176-189, April 1991.

[3] D.J. Kuck, Y. Muraoka, and S.C. Chen, "On the Number of Operations Simultaneously Executable in FORTRAN-like Programs and their Resulting Speedup", *ACM Transactions on Computers*, C-21: 1293-1310, December 1972.

[4] K. Agrawala, R. M. Bryant, and J. M. Mohr, "An approach to the workload characterization problem", *Computer 9(6),* pp. 18-32, June 1976.

[5] K. Theobald, G. Gao, L. Hendren, "On the limits of Program Parallelism and its Smoothability", *In 25th Annual Symposium on Microarchitecture*, pages 10-19, December 1992.

[6] Karkowski, H. Corporall, "Exploiting Fine-and Coarse-grain Parallelism in Embedded Programs", In *International Conference on Parallel Architectures and Compilation Techniques*, October 12-18, 1998.

[7] L. John, P. Vasudevan, and J. Sabarinathan, Workload Characterization: Motivation, Goals and Methodology, in *Workload Characterization: Methodology and Case Studies, IEEE Computer Society*, edited by Lizy John and A. M. G. Maynard, 1999. ISBN 0-7695-0452-3.

[8] L. Wills, T. Taha, L. Baumstark Jr, S. Wills, “Estimating Potential Parallelism for Platform Retargeting”, *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE)*, pages 55-64, IEEE Computer Society Press, Richmond, VA, October 2002.

[9] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, R. Brown, “MediBench: A free, commercially representative embedded benchmark suite”, *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.

[10] M. Lam, R. Wilson, “Limits of Control Flow on Parallelism”, In *Proceedings of the 19th Annual International Symposium on Computer Architecture* Gold Coast, Australia, May 19-21, 1992, pp. 46-57.

[11] Nicolau, J. Fisher, "Measuring the Parallelism Available for Very Long Instruction Word Architectures", *IEEE Transactions on Computers*, Vol. C- 33, No. 11, pp. 968-976, November 1984.

[12] Ramakrishna Rau, J. Fisher, “Instruction-Level Parallel Processing: History, Overview, and Perspective”, *The Journal of Supercomputing* 7(1-2), 1993.

[13] T. Austin and G. Sohi, “Dynamic Dependency Analysis of Ordinary Programs”, *In Proceedings of ISCA-19*, pages 342--351, May 1992.