Performance Analysis of Compressed Instruction Sets on Workloads Targeted at Mobile Internet Devices

Chander Sudanthi *Mrinmoy Ghosh* *Kevin Welton* *Nigel Paver*
ARM Inc
{chander.sudanthi, mrinmoy.ghosh, kevin.welton, nigel.paver}@arm.com

ABSTRACT

This paper describes the performance advantages of a two and four byte variable length instruction set, Thumb2, over a four byte fixed length instruction set, ARM. Both instruction sets are found in ARMv7-A ISA compatible processors, such as the ARM Cortex-A8 and Cortex-A9. The code size reduction when using a variable length instruction set is well understood and can be significant. The focus of this paper is the performance advantage of increased code density. With Thumb2 more instructions are stored in the I-cache, increasing I-cache hit rates, and in turn increasing the performance of the processor. To demonstrate the performance advantage of Thumb2, a Mozilla based web browser built for Thumb2 and ARM on Linux is run in a full system emulator and in a full system instruction set simulator with a cache model. Switching from the ARM four byte fixed length instruction set to the Thumb2 two and four byte variable length instruction set results in a 1.07x improvement in performance and 33% improvement in code density.

1. Introduction

System on Chips (SoCs) are built into physically space constrained products and are becoming increasingly complex, limiting the available room for memory. SoCs can be found in smartphones, netbooks, and Mobile Internet Devices where size is a key design point. In such devices adding additional memory can be costly or jeopardize the form factor. Modern application processors such as Texas Instruments (TI) OMAP processors [6] not only include complex processing cores such as the ARM Cortex-A8 [1] but also contain hardware support for 3D graphics and video. Package and power constraints limit the amount of I/O that can be dedicated to memory and the speed at which the memory can run while staying within a mobile power envelope. In these platforms the memory resource is shared by the many masters in the system. This makes memory bandwidth and usage a premium, so that any techniques that reduce an individual masters memory usage can improve overall system performance.

With the continuing push to reduce memory requirements and thus cost in the embedded space came the advent of the Thumb2 instruction set. Code density increases with the variable length Thumb2 instruction set, but the instruction count increases slightly as well. The Thumb2 instruction set may show a slight degradation in processor performance when compared to the ARM instruction set. However when the total system performance is considered, as we will demonstrate in this paper, Thumb2 actually improves performance, decreasing miss rates and execution time. A web browser, a typical and often used application in smartphones and Mobile Internet Devices, is used to evaluate the performance of Thumb2 versus ARM.

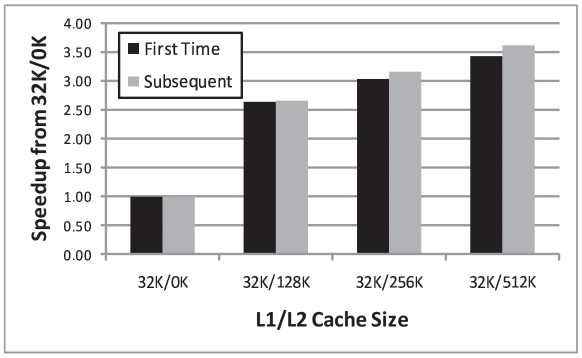

Figure 1: Web browser cache size sensitivity

2. Motivation

The web browser is a critical component of Mobile Internet Devices and is quickly becoming the user interface convergence point. Previous work has shown that the web browser has a significant memory dependence [9]. Investigating further, instruction cache miss rates in the browser are high, which makes the web browser a good candidate for evaluating the impact of variable length instruction sets.

Cache sizing, memory bandwidth, and memory latency all significantly affect web browser performance. Figure 1 shows the speedup of web browser load time from a system without an L2 cache to systems with different L2 cache sizes. The speedup plotted in Figure 1 is derived from time taken to load five web pages four times. Figure 1 shows two bars for each cache configuration,*"First Time"* is referring to the load time of the first loop iteration and *"Subsequent"* is referring to the average load time of the remaining iterations. Looking at the *"Subsequent"* speedups, there is a 2.66x speedup from 0k L2 to 128K L2, 1.19x speedup from 128K L2 to 256K L2, and 1.14x speedup from 256K L2 to 512K L2 [9].

Adjusting the bus and memory frequency in unison affectively lowers the bandwidth and latency to memory. We use the nomenclature $r\_core : r\_bus : r\_mem$, where r_core/r_bus is equal to the ratio of core frequency to bus frequency and r_core/r_mem is equal to the ratio of core frequency to memory. As shown in Figure 2, changing system ratios from 8:1:1 to 4:1:1 shows a speedup of

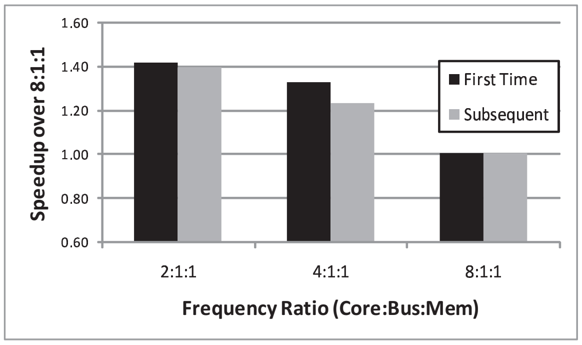

Figure 2: Web browser system ratio sensitivity

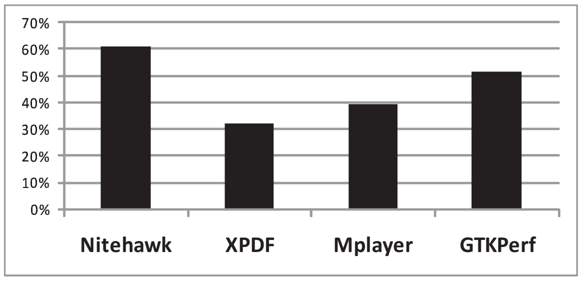

Figure 3: Percentage of Instructions in L2 Cache Accesses

1.23x and 4:1:1 to 2:1:1 shows a speedup of 1.13x in the web browser.

These results show that the web browser has significant memory subsystem dependence. A logical method to increase web browser performance is to improve cache efficiency. By using an instruction set simulator with a cache model, we discovered that a significant portion of cache accesses during execution of the web browser are in the instruction side. Specifically, Figure 3 shows that the instruction side constitutes 61% of the L2 cache accesses in the *Nitehawk* web browser. Our hypothesis is that the web browser has a large code base and uses a large number of shared libraries. One innovation that will help in this situation is Thumb2 since it can increase code density and decrease instruction side misses. In the sections below, we show that switching to Thumb2 in the web browser reduces both miss rate and execution time.

3. Thumb2 instruction set

The Thumb2 instruction set is a variable length instruction set with 16-bit and 32-bit instruction encoding. It is based on the Thumb instruction set, which was first introduced as a subset of the ARMv4 instruction set. The processor either operates exclusively on ARM or exclusively on Thumb2 instructions, using a special branch instruction to switch between operating states. Thumb is a 16-bit instruction set that has equivalents for a subset of the 32-bit ARM instruction set. Thumb2 augments Thumb with additional 16-bit instructions for improved program flow and adds 32-bit instructions [11]. This variable length instruction set has equivalents for almost all ARM instructions, removing the need to switch to the ARM instruction set to execute code sequences and thus increasing the execution efficiency of Thumb2.

Thumb2 has a code size advantage over ARM when Equation (1) is true. Equation (1) is true when the number of 16-bit Thumb2 instructions is large enough to offset any increase in instruction count. We will show that as code density improves with Thumb2, the instruction cache hit rate increases and the overall execution time decreases.

$$\begin{aligned} &16bits * (No.\ of\ 16bit\ Thumb2\ instructions) + \\ &\quad 32bits * (No.\ of\ 32bit\ Thumb2\ instructions) \\ &\qquad < 32bits * (No.\ of\ ARM\ instructions) \end{aligned} \quad (1)$$

4. Methodology

To analyze the target applications we use:

- a full system emulator to gather execution time
- a full system instruction set simulator with a cache model to gather cache statistics

An important and time consuming task required to do this study is to rebuild the entire Linux stack (kernel, libraries, applications) for the target instruction sets.

4.1. Verilog RTL Emulation Platform

Results derived from execution time are measured using a full system emulation platform. The hardware for the platform is based on the Cadence Incisive Palladium series of emulators and is capable of running synthesized Verilog among other languages [4]. Various system components made of ARM Intellectual Property (IP) glued together constitute the full system RTL that runs on the emulator. Full system in this context means that the environment includes the core, memory, memory controller, interconnection fabric, and other peripherals necessary in an end user system. Given this environment, an OS can boot on the emulator.

Targeting the emulator with Linux is straightforward since the platform closely adheres to ARM development board specifications. Full X Windows can be brought up on the emulator and the LCD output can be seen on the host machine. The emulated system runs within the 1-2 MHz range which is sufficient to run reasonably large applications.

One of the emulation platforms's key advantages is the ability to fully configure systems. Since all the components are ARM RTL, registers can be configured as they would be in an end user system. Some of the registers in the system that can affect performance include the L2 controller registers and dynamic memory controller registers. Also, modifications can be made that could not be made in a fixed hardware environment. Such configuration options include L2 size, L1 size, synchronicity of interconnection fabric, and the frequencies of various components in the system.

4.2. System-Level Simulation Platform

An ARM simulator created for accurate software simulation was used for comparison against the RTL emulation platform. The system simulator is a very capable

Table 1: List of Benchmarks

Benchmark Name	Description
Nitehawk	A Mozilla based web browser loading a set of media rich web pages
GTKPerf	A benchmark that is designed to test GTK+ performance
MPlayer	A media player playing a 30 second King Kong movie trailer
XPDF	The XPDF application loading a media rich PDF file

platform that can run the same system images and benchmarks as the RTL emulator. It has the ability to perform instruction-accurate simulations on validated high-speed models of ARM IP. Additionally, it properly models peripherals like UARTs, color LCD controllers and dynamic memory controllers. The simulator deviates from the RTL emulation platform in the sense that it is a functional model, providing no timing information. The simulator was originally built with the motivation to provide a fast, functionally accurate virtual prototype for SoC products. The current version was modified for this work by integrating a multiple cache hierarchy simulator. This infrastructure enabled the data and instruction stream from the simulator core to be passed through a large number of cache hierarchies and gather statistics like hits and misses.

4.3. Software Stack

We performed our experiments using a number of benchmarks pertinent to the SoC domain. A list of the benchmarks used and their brief description is given in Table 1. The software platform consists of a Linux 2.6 Kernel, Nitehawk, XPDF, MPlayer, GTKperf, and an ARM Optimized X Window System. The most important benchmark in our evaluation is *Nitehawk* as described below.

Nitehawk is a Mozilla based web browser. The test environment for *Nitehawk* emulates a user visiting multiple media-rich websites on a Mobile Internet Device or smartphone. To emulate visiting multiple websites, five HTML pages are loaded one after the other and each set of pages is loaded four times. The suite's web pages are locally stored, eliminating potential network delays from impacting performance numbers. Emulated user interactions include rapidly loading, scrolling, and navigating links present within the included test sites. This web browser execution invokes multiple user-level processes which must use interprocess communication for rendering, image conversion, and GUI interactions. As these applications exist within separate address spaces, proper modeling of operating system internals and virtual memory are essential for identifying any microarchitectural bottlenecks related to the rendering process.

The Linux stack, including the kernel, applications, and libraries, are built for ARM and Thumb2 ISAs. Tools are available to help create a Linux stack [2] [5]. The framework includes applications such as Busybox, Cario, D-BUS, GStreamer, GTK, and Xserver [2]. The same Codesourcery GNU GCC cross compiler is used for the Thumb2 and ARM Linux stacks to ensure accurate comparison of Thumb2 and ARM. The compiler options for ARM and Thumb2 builds are the same except for the addition of the *-mthumb* option in Thumb2 builds.

5. Results

Some benchmarks like the SPEC 2000 benchmark suite have extremely high instruction cache hit rates [3]. For these benchmarks, increasing code density and hence reducing instruction cache miss rates will not have a significant effect on system performance. However, certain applications that are extremely important in the SoC domain, like the web browser, have very different instruction cache characteristics as compared to SPEC. The L1 I-cache miss rate for the *Nitehawk* web browser with a 32KB 4-way L1 I-cache is 2.23%, compared to 0.02% for SPEC.

In order to analyze the effect of applications with appreciable instruction cache miss rates, we analyze the mix of accesses to the L2 cache. In Figure 3 we plot the percentage of L2 cache accesses that are generated by the L1 instruction cache. We can see that the maximum effect is in *Nitehawk*, whose L2 accesses consist of 61% instruction side accesses. The benchmark *GTKperf* also has a large percentage of instructions, 51%.

Figure 4 and Figure 5 show the improvement from using Thumb2 in L1 I-cache miss rates for the four benchmarks. We can clearly see that there is significant reduction in instruction cache miss rates for all four benchmarks. The miss rates for *Nitehawk* reduces by 40% while *GTKperf* reduces by 43%. This significant decrease in L1 I-cache miss rates is due to increased code density of the Thumb2 ISA. However, we must analyze how this decrease in I-cache miss rate affects application performance. It should be noted that XPDF and MPlayer from Figure 5 have significantly lower instruction cache miss rate than Nitehawk and Gtkperf from Figure 4.

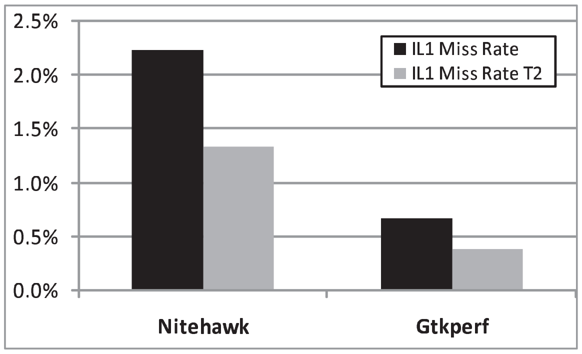

Figure 4: L1 I-cache Miss Rates

To analyze the impact of the reduction in instruction cache miss rate we investigate *Nitehawk* further, as of the four application miss rates analyzed *Nitehawk* has the highest L1 I-cache miss rate and percentage of instructions in L2 cache accesses. The performance of the same web browser benchmark was measured with the use of the full system emulation platform previously discussed. Figure 6 shows that switching from ARM to Thumb2 improves load time of *Nitehawk* by 1.07x.

The *Nitehawk* web browser makes use of 64 shared libraries. The code density improvements in the six largest

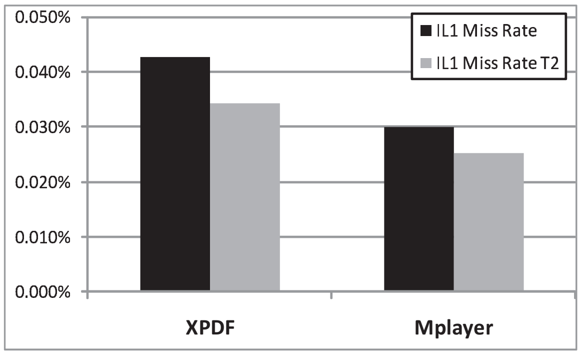

Figure 5: L1 I-cache Miss Rates

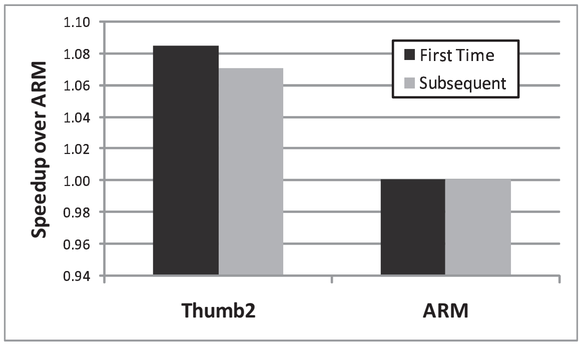

Figure 6: ARM v/s Thumb2 load time on Nitehawk using a dual issue 768 MHz 4:1:1 processor

shared libraries by code size and in the browser object are enumerated in Table 2. Based on Table 2, the code size advantage of Thumb2 is 33%.

6. Related Work

There are papers [8] [10] [7] that discuss advantages of compressing instruction streams, but none evaluate advantages of compressed instruction sets. Also, when evaluating compressed instruction streams no papers evaluate the execution time impact on a realistic workload, like a web browser on Linux, and on a realistic platform, like a full system RTL emulator.

Chen, et al. in [8] provide data on miss rate reduction when introducing an instruction stream compression technique. It does not address compressed instruction set advantages, the performance impact on the rest of the design by introducing the hardware compression technique, nor the execution time on a realistic application and platform.

[10] and [7] are examples of the many papers that discuss hardware techniques to compress instruction streams. However, these papers fail to evaluate the performance tradeoffs when introducing the compression technique, focusing rather on the compression or energy advantage of the technique.

7. Conclusion

This paper shows that the variable length Thumb2 ISA provides an appreciable speedup and significant reduction in code size in applications targeted at Mobile Internet Devices. This paper provides the details as to why both performance improvement and code size reduction can be possible. The Thumb2 variable length instruction set is able to hit a sweet spot of code size and functionality between the original Thumb ISA and the ARM ISA. The sweet spot is where code density increases without a proportional execution efficiency loss, resulting in a net speedup. After evaluation of several application miss rates, a Mozilla based web browser is chosen to highlight the advantages of Thumb2. The web browser is a key application in smartphones and Mobile Internet Devices. The paper shows a 1.07x performance improvement combined with a 33% reduction in code size for the web browser.

Table 2: ARM and Thumb2 Nitehawk Code Size

	Code Size (KiB)		
	ARM	T2	Decrease
Shared Libraries			
libxul.so	9408	6122	35%
libgtk-x11-2.0.so.0.1200.3	1796	1195	33%
libnss3.so	827	526	36%
libc-2.5.so	610	612	0%
libmozjs.so	538	365	32%
libX11.so.6.2.0	531	346	35%
Application Binary			
xulrunner	23	15	36%
Total	**13735**	**9180**	**33%**

8. REFERENCES

[1] ARM Cortex-A8. http://www.arm.com/products/CPUs/ARM_CortexA8.html, 2009.

[2] ARM Linux Internet Platform. http://linux.onarm.com, 2009.

[3] Cache performance of SPEC CPU2000. http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/index.html, 2009.

[4] Incisive Palladium Series. http://www.cadence.com/products/sd/palladium_series, 2009.

[5] Linux Operating System Download. http://www.arm.com/products/os/linux_download.html, 2009.

[6] OMAP 3 Processors: OMAP3640. http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12837&contentId=52605, 2009.

[7] Luca Benini, Alberto Macii, and Alberto Nannarelli. Cache-code compression for energy minimization in embedded processors. In *Proceedings of the 2001 international sympossium on Low power electronics and design*, 2007.

[8] I-Cheng K. Chen, Peter L. Bird, and Trevor Mudge. The impact of instruction compression on i-cache performance. Technical Report Technical Report CSE-TR-330-97, University of Michigan, 1997.

[9] Mitchell Hayenga, Chander Sudanthi, Mrinmoy Ghosh, Prakash Ramrakhyani, and Nigel Paver. Accurate system-level performance modeling and workload characterization for mobile internet devices. In *Proceedings of the 9th workshop on MEmory performance: DEaling with Applications, systems and architecture*, 2008.

[10] Aleksandar Milenkovic and Milena Milenkovic. An Efficient Single-Pass Trace Compression Technique Utilizing Instruction Streams. In *ACM Transaction on Modeling and Computer Simulation*, 2007.

[11] Richard Phelan. Improving ARM Code Density and Performance. http://www.arm.com/pdfs/Thumb-2CoreTechnologyWhitepaper-Final4.pdf, June 2003.