Erik Knechtel
Boston University College of Engineering
EC700 Final Project Report
7 May 2014

# Benchmarking a Unique Heterogeneous CPU System: The Beaglebone Black's Applicability for Space Science

## Introduction

For this project, I have chosen to examine the performance of various benchmarks on the Beaglebone Black platform running Ubuntu 14.04 (kernel version 3.8.13-bone47). Specifically, I am looking at optimization methods for several benchmarks in the Phoronix[1] and MiBench[2] test suites which focus on CPU utilization, memory access, and network throughput.

## Background and Related Work

I chose this project because I want to demonstrate the capabilities of the Beaglebone Black (hereafter referred to as the Beaglebone) and its suitability to handle multiple CPU- and memory-intensive operations. This development board will be used on board the ANDESITE student satellite project being designed and built here at Boston University, and our team needs to have a clear understanding of the capabilities of what will become the central computer of the satellite system, what software and hardware bottlenecks exist, and how to alleviate those bottlenecks to ensure solid performance in a variety of demanding situations.

During flight, the satellite will have to balance the operation of several payloads and a number of important tasks (such as downlinking data to the Earth) in a way which is quite similar to a data center's efforts to load-balance the work across many servers while meeting client's quality of service agreements.[3] Rather than load balancing virtual machines between identical servers,the ANDESITE system will have to duty cycle the payloads on and off in accordance with scientific requirements (for example, some payloads only need to be active while the spacecraft is at certain latitudes) as well as the hard requirements of available power[4] and heat dissipation. And while a datacenter can place multiple virtual machines from different clients on the same or different hardware as needed, the systems on board a satellite will all be running on unique, purpose-built hardware. These processes can therefore benefit from hardware acceleration but there are no opportunities to perform software load balancing among different subsystems.

Another important limitation is that unlike a client's VM in a datacenter or a high performance computing experiment, we cannot do any form of checkpointing[5] because a scientific instrument may have only a few minutes to gather the necessary data at a point in orbit, and if there is a failure or bottleneck then we cannot simply roll back to the previous checkpoint. The data flow must be continuous, although small amounts of buffering (on the order of kilobytes) are necessary.

The ANDESITE satellite will need to receive, analyze/sort, compress, encrypt, and transmit a large amount of data continuously down to Earth, as well as receiving data from us and decrypting it. The mission requirements call for gathering gigabits of data on every orbit and the transmission window will be sparse as the ANDESITE spacecraft passes over friendly GENSO stations[6] (the volunteer-run stations used for communicating with some educational and low-budget satellites), potentially having as little as several hours each day to transmit the data. The downlink will be further throttled by the low rate of transmission in the chosen frequency. Thus, it is necessary to optimize every step of this process as much as possible. And while it may be tempting to dismiss the encryption step as unnecessary for the science goals, there have been many attempts in the past to hack satellite command-and-control protocols with surprising successes[7,8,9,10] so the encryption must be world-class and performed on every piece of data transmitted.

The Beaglebone Black's processor is an AM335x ARM Cortex A8 chip with a single core that can operate at frequencies up to 1GHz. The main disk takes the form of 2GB of flash memory, and there are 512MB of DDR memory (490MB available on the system used for this paper). Interestingly the AM335x CPU has two smaller 32-bit RISC cores called the Programmable Realtime Units or PRUs. These two cores operate at 200MHz and must be programmed in an assembly language that is limited to programs of two thousand instructions per core or fewer, due to

the 8KB of instruction RAM available.[11, 12,13]

These extra cores offer exciting capabilities for accelerating the performance of the Beaglebone system by offloading certain operations, such as UART access to peripherals through the GPIO pins which they have access to (the PRUs can access 18 of the 66 GPIOs). Furthermore the usage of "lesser" cores in a CPU has been researched extensively as a way to accelerate mature computations[14] or provide a low-power alternative to the main CPU.[15,16]

**Project setup and methodology**

From these tight requirements, and going off work performed by Joshua Datko[17] and others, I decided to look at optimizing CPU energy-delay product using dynamic voltage and frequency scaling (DVFS) through the "governors" included in the cpufreq program,[18] optimizing memory access speeds, and enabling available hardware acceleration for cryptographic functions including AES, SHA, MD5, and an on-chip RNG.[11] Thus the project is a look at how much hardware acceleration and DVFS can help (or not) for the security, network and telecommunications themed benchmarks in MiBench and the scientific and compression benchmarks in the Phoronix suites. These specific benchmarks were chosen for their relevance to ANDESITE's scientific mission and engineering requirements. The benchmarking suite MEVBench was considered but ultimately deemed inappropriate, due to its focus on computer vision and multithreading,[19] and unnecessary since Phoronix and MiBench offer comprehensive test suites to cover a wide variety of application areas that are more relevant to ANDESITE's mission requirements.

I also used the Tor internet anonymizing utility[20] to provide some unpredictable network activity in the background. The Tor daemon uses about 17MB of memory on the Beaglebone and a small (<1%) amount of CPU resources continuously. This is subject to unpredictable spikes every few seconds (up to about 20% CPU usage and about 35MB of memory) and so it does a good job of simulating unpredictable low-level background activity. Aside from Tor, I wrote a script which repeatedly uses dd and gzip to create, compress, and delete a large file over and over to put stress on both memory access (via dd) and CPU usage (via gzip).

Ubuntu and many other versions of linux support DVFS "governors" which are programs that control CPU frequency scaling. On the Beaglebone black there are four frequencies which can be chosen, 300MHz, 600MHz, 800MHz, and 1GHz. The benchmarks were run first with the DVFS governor set to "ondemand", which tries to keep the CPU frequency as low as possible and only makes short exceptions to raise the frequency above 300MHz. When ondemand raises the frequency it sends it straight up to maximum, so the CPU only throttles between min and max. Then the benchmarks were performed with the "performance" governor which keeps the frequency at maximum (1GHz) even when idle. Thus we expect to see better performance due to a lack of switching costs from going between 300MHz and 1GHz. A more realistic governor for usage on board ANDESITE would be the "conservative" governor, which is similar to ondemand (preferring 300MHz unless CPU usage goes above 95%) but unlike ondemand, when it scales up the DVFS it goes step by step, first going to 600MHz, and then if CPU usage is again above 95%, moving to 800MHz and so on up to maximum.

**Benchmarks Used**

The first benchmark I used was SciMark 2.0, a benchmark suite produced by the National Institute of Standards and Technology, which was chosen to present a variety of scientific tasks that the ANDESITE computer will most likely have to perform. These include fast Fourier transforms, Monte Carlo integration, and three different types of matrix math: LU factorization, sparse matrix multiplication, and Jacobi "successive over-relaxation" which is a method of solving linear equations.[21] These tests were performed first with and without Tor running in the background, then with both Tor and the script I created in the background, and finally with different levels of DVFS and cryptographic hardware acceleration enabled.

The second benchmarking suite is another Phoronix test suite which uses different types of compression algorithms to analyze the processor's performance. ANDESITE will be doing a very large amount of complicated data compression to try and rectify the difference between the gigabits of data it receives from its flock of sensors, and the kilobits of bandwidth that it has available to downlink that

data.[22] Some of this compression will be done with unique, purpose-built algorithms that are specific to the types of data we anticipate acquiring and the specific parts of that data we are interested in. Much of this depends on the finalized design and performance of the sensor systems. However, widely available open-source algorithms can approximate these uses quite closely. The Phoronix suite uses 7zip,[23] Parallel BZIP2 (pbzip2),[24] and LZMA[25] algorithms.

The MiBench embedded benchmark suite offers a wide range of commercially representative algorithms for different areas of use. I have selected networking, security, and telecommunications. The networking benchmarks are a matrix computation based algorithm known as Dijkstra which simulates computing a shortest path between nodes in a network, and the traversal of a Patricia Tree which simulates altering routing tables dynamically. In the cryptographic suite I have chosen SHA encryption and the Blowfish cipher because variants of SHA can be used to encrypt data or compute hashes, and Blowfish or algorithms similar to it will be used for secure communication of data and commands. Thus these two benchmarks cover everything from error checking (SHA is used for calculating MD5 hashes) to securing a communication stream. Finally in the telecommunications suite there is another version of the fast Fourier transform, a GSM (Global Standard for Mobile) communications encoder/decoder operating on a large sample of voice data, an Adaptive Differential Pulse Code Modulation (ADPCM) algorithm also operating on speech samples while performing 4:1 compression, and a 32-bit error checking algorithm called Cyclic Redundancy Check (CRC) using the same data set as the ADPCM benchmark. The ADPCM benchmark performance was highly variable, far more so than the others in MiBench, so I ran it ten times and averaged. The MiBench tests were not run with Tor, dd, or gzip in the background.

Finally, all of these benchmarks were run again after implementing the steps outlined by Joshua Datko to enable hardware crypto acceleration, and using the "performance" governor to create a best-case scenario for comparison.

**Experimental results**

As mentioned above, these tests were run first with a "fresh install" system using the defaults for Ubuntu 14.04 and the kernel image 3.8.13-bone47. Then a second run was performed for calibration, a run with Tor active in the background, a run with dd, gzip and Tor all active, and then with none of those programs active while I performed DVFS scaling using the performance and conservative CPU frequency governors. The ondemand governor is the default and so that was being used during all the other runs.

SciMark 2.0 Results

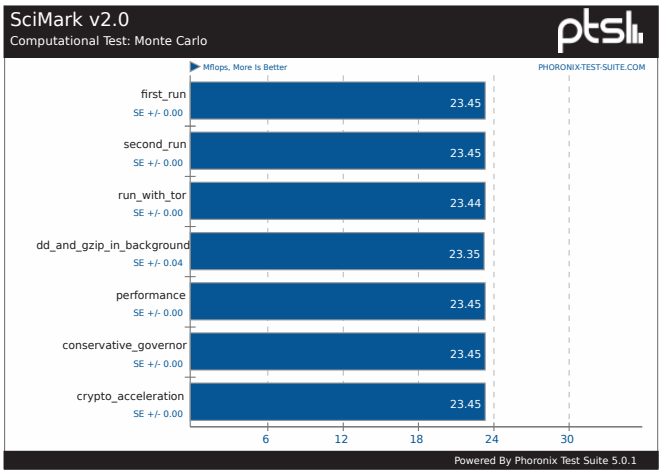


*Figure 1: SciMark 2.0 - Monte Carlo estimation*

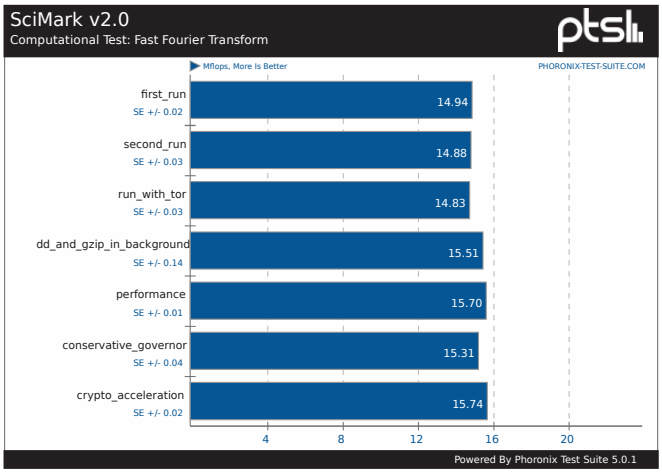


*Figure 2: SciMark 2.0's version of FFT*

From the results shown in Figure 1, the Monte Carlo estimation was nearly unaffected by any alterations to CPU frequency or background activity. This is more surprising given that this benchmark is

supposed to exercise the random number generator, the hardware for which is enhanced by the addition of the cryptographic hardware acceleration kernel module. It appears that the calculation is simple enough for the processor to handle without trouble, even under heavy loads.

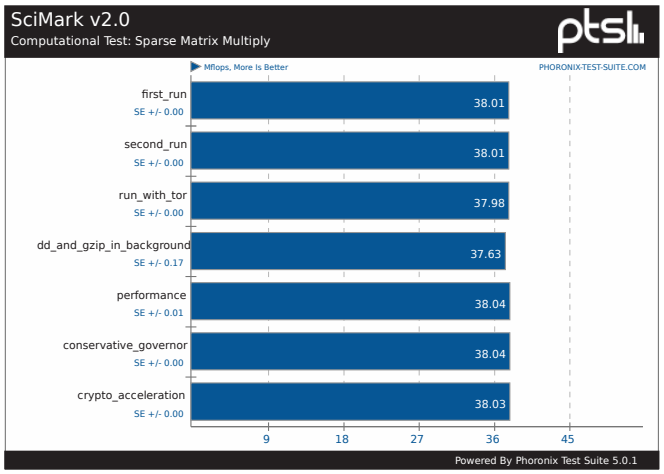


*Figure 3: SciMark 2.0 - Sparse Matrix Multiplication*

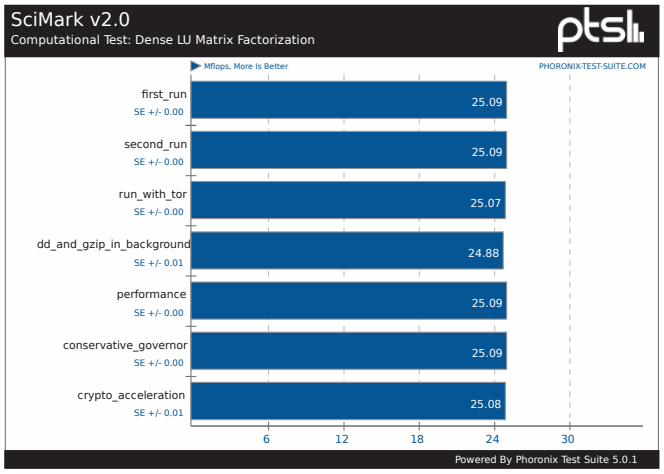


*Figure 4: SciMark 2.0 - LU matrix factorization*

Figures 2, 3, 4, and 5 show the impact of these environmental changes on the other algorithms in the SciMark 2.0 suite. The fast Fourier transform showed erratic results and high variability. The three linear math applications showed reduced performance when dd and gzip were running in the background, but were otherwise agnostic to environmental changes.

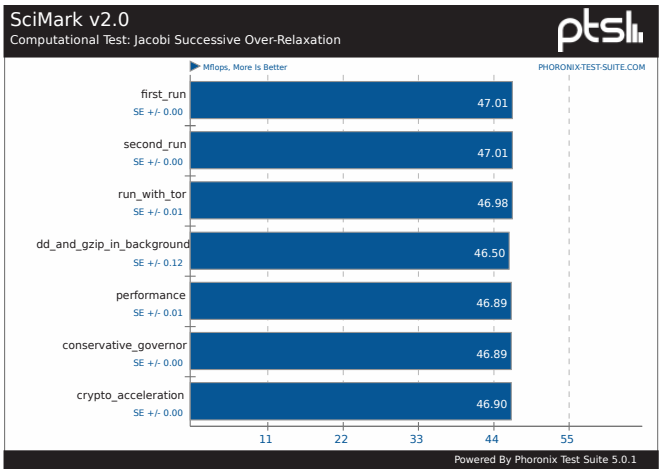


*Figure 5: SciMark 2.0 - Jacobi SOR matrix math*

It appears that the AM335x CPU is well optimized for these calculations and is robust against heavy workloads causing performance degradation for linear calculations.

## Compression Suite Results

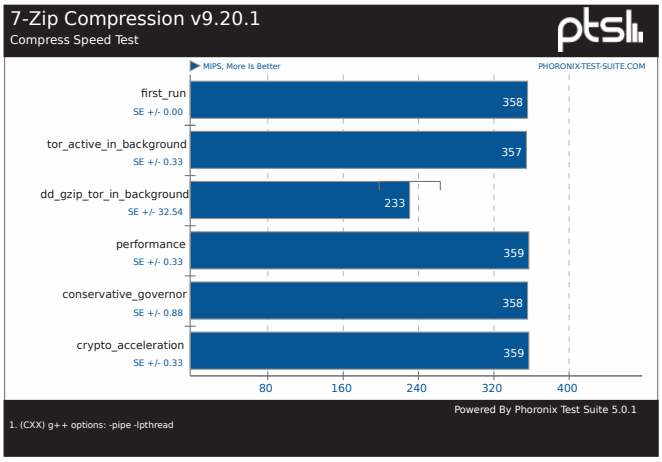


*Figure 6: 7zip compression. More is better.*

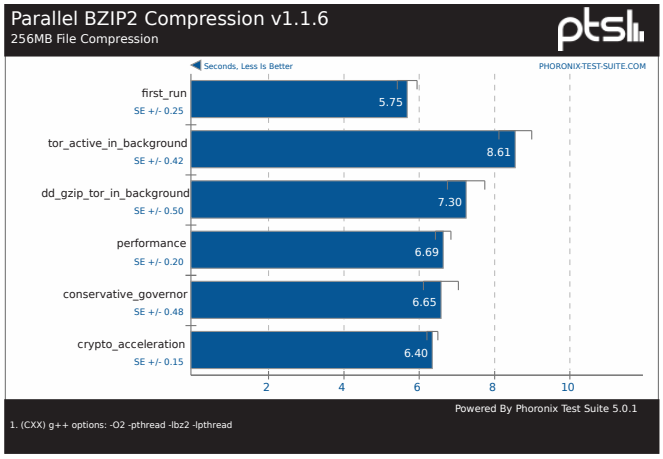


*Figure 7: Parallel BZIP2 compression duration in seconds. Less is better.*

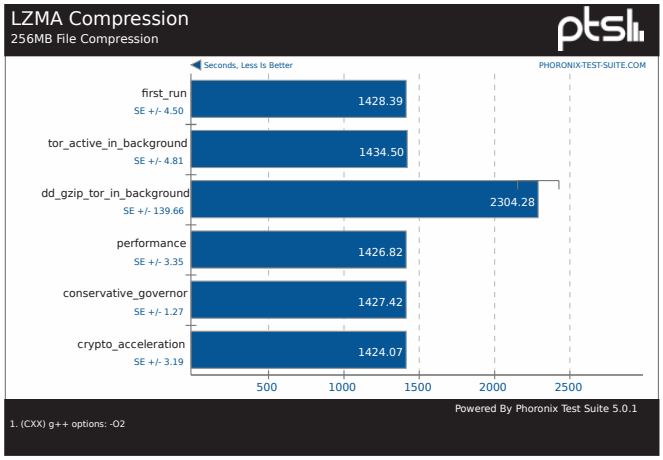


*Figure 8: LZMA compression in seconds. Less is better.*

The compression suite of benchmarks showed much more variability and dependency on the environment, specifically on the CPU load. LZMA is used as part of 7zip, so one would expect similar results between those two benchmarks. The graphs here show MIPS for 7zip and seconds-to-completion for LZMA, so although the graphs are inverses of each other they do show a strong connection since both suffer dramatically from having dd and gzip run in the background while being agnostic to other environmental variables. They even perform equally well despite the frequency switching costs of the ondemand and conservative governors, compared with the performance governor.

The Beaglebone has only a single core processor, so one would expect no benefit from the multithreaded nature of pbzip2. However, upon running the pbzip2 benchmark with the Tor program active in the background, there was considerable slowdown as demonstrated in Figure 7. The excellent performance during the first run on a clean install indicates that in the course of setting up the tests and using the Beaglebone, I may have introduced a number of active threads into the environment despite my efforts to eliminate unnecessary activity in the background. It's possible that the Tor daemon has to check frequently for activity and may cause frequent short interruptions to thread scheduling on the processor, disrupting a very thread-dependent program like pbzip2.

## MiBench Results

The MiBench suite contained programs which ran extremely quickly, on the order of 1-8 seconds rather than 1-4 hours like the Phoronix testbed. As a result, I deemed it impractical to do comparisons with dd, gzip, and Tor running in the background and instead focused on the effects of dynamic frequency scaling and hardware acceleration. These graphs show the amount of time the process spent in the kernel (shown in orange as “sys”), outside the kernel (in red, as “user”), and overall real-world time shown in blue. The user and sys time are only for time spent on this exact process, while real-world time takes in to account all delays that a user of the system experiences in waiting for the process to be completed. Thus frequent interruptions from other threads or delays in accessing main memory would show up in the “real” time segment of each graph.

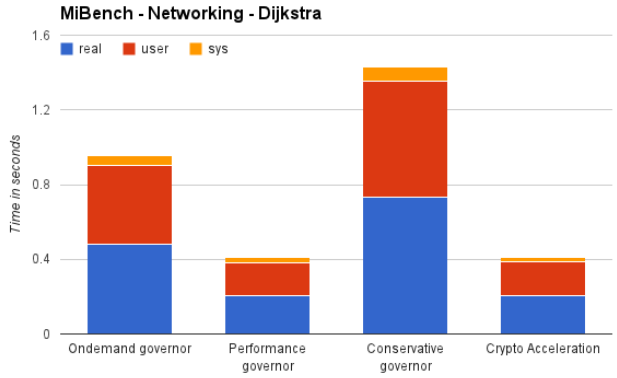


*Figure 9: The Dijkstra algorithm*

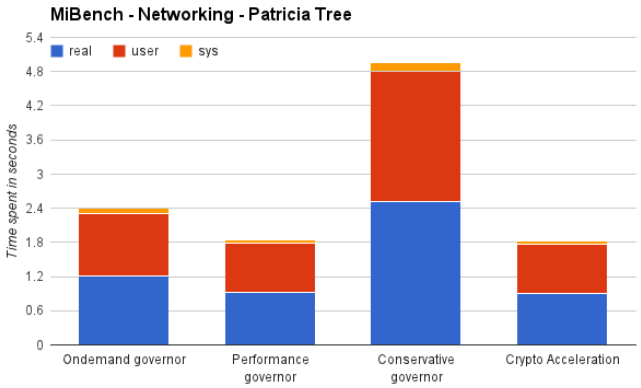


*Figure 10: The Patricia Tree data structure*

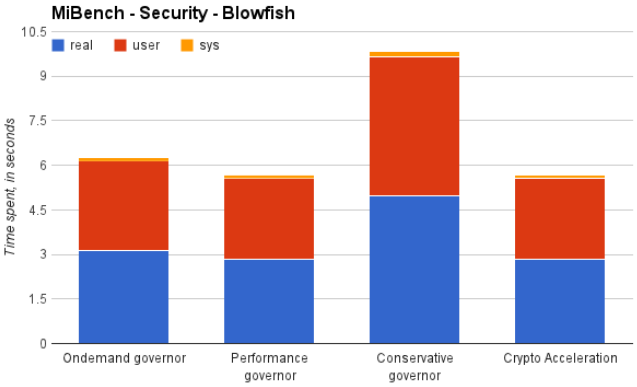


*Figure 11: Blowfish encryption*

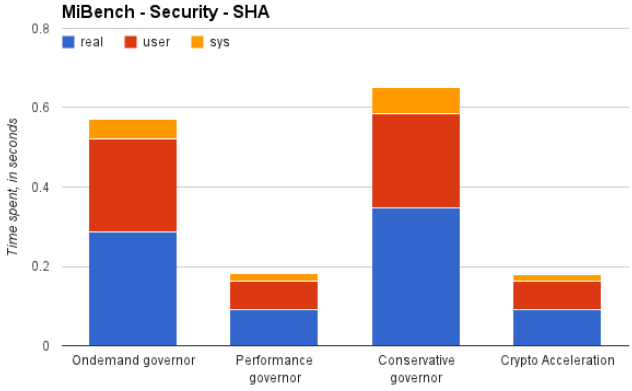


*Figure 12: SHA-1*

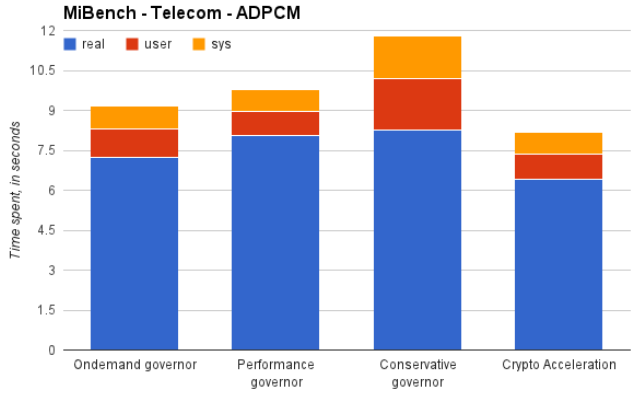


*Figure 13: Adaptive Differential Pulse Code Modulation*

For all of these benchmarks except ADPCM, the real time spent in the process is almost entirely a sum of the time spent in user-space and kernel-space working on that process. However, for ADPCM there are large delays from other processes which affect the time to completion. The implementation of ADPCM here may be heavily reliant on memory access, possibly during the compression phase, and thus suffers more from short interruptions from other activity using the bus. As mentioned previously the ADPCM benchmark was highly variable, an indication of susceptibility to the CPU and memory environment. It was also the only algorithm which appeared to benefit from use of the cryptographic hardware acceleration.

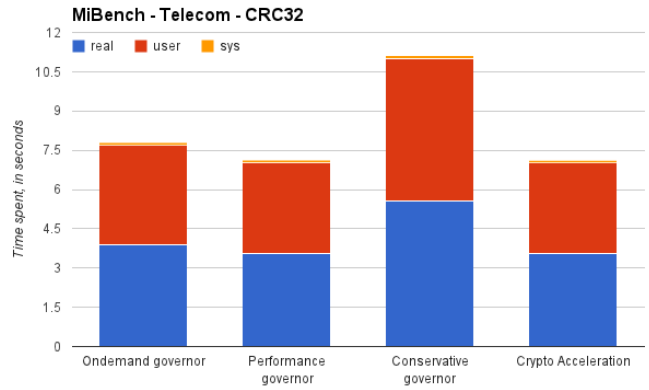


*Figure 14: 32-bit Cyclic Redundancy Check*

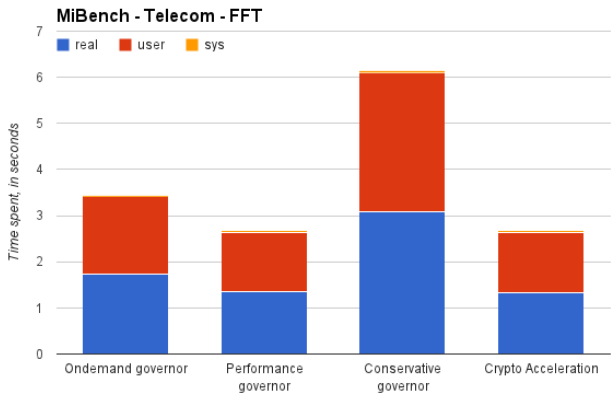


*Figure 15: Another version of the FFT*

The version of the fast Fourier transform used in the MiBench suite differs from the version used in Phoronix by being two-dimensional rather than one-dimensional, and using pseudo-random complex data. The Phoronix version also performs the benchmark in two discrete steps rather than one.

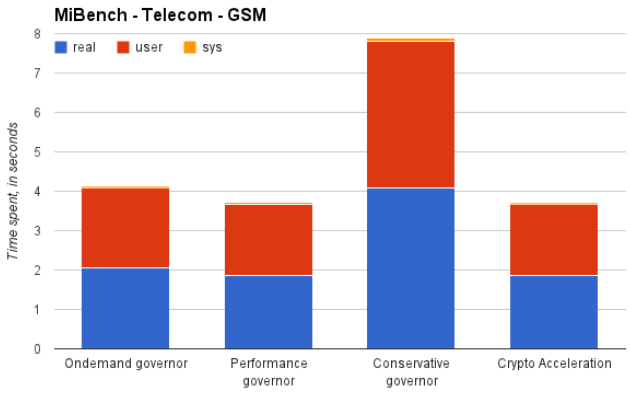


*Figure 16: Global Standard for Mobile encode/decode using Time- and Frequency-Division Multiple Access encoding.*

## Conclusions

From Figures 1-5 it appears that the Beaglebone's CPU is quite optimized for matrix math, showing nearly the same time of computation at 300MHz as it does at 1GHz.

Figures 6-8 demonstrate that compression is quite a different matter, being reliant on CPU frequency as well as memory access times, and not robust against sharing the CPU and memory bus with other processes. The need to access memory to process files which are hundreds of megabytes in size means that the 64KB L1 cache and 256KB L2 cache are not nearly enough and calls to main memory are a major factor in performance. This is not something which could be optimized and is a likely bottleneck for ANDESITE's main computer.

The MiBench suite provides an interesting look at the switching costs inherent in using any of the dynamic governors for DVFS, seen in Figures 9-16. Specifically the conservative governor, which does the most frequency scaling (with resolution on a 10ms time scale, meaning the frequency may be readjusted every 10ms) created the most delays in performance which is unfortunate since it is also the governor most optimized for battery-dependent systems such as phones, laptops and satellites. Cryptographic hardware acceleration did not play much of a role, even for SHA-1 computations, which is surprising but these algorithms are also between 15-20 years old and so the AM335x CPU may already be highly optimized for them. It is unfortunate the MiBench suite does not include more recent algorithms.

The time saved by using the static "performance" governor, which locks the CPU frequency at 1GHz, is a major contributor to shorter algorithms like Dijkstra and SHA-1 (Figures 9 and 12 respectively) where the switching cost from ondemand's transition from 300MHz to 1GHz leads to more than triple the time spent in computation. This effect is less noticeable in direct proportion to time spent in the process. This means that if ANDESITE's system designers intend to do heavy network activity or telecommunications during a certain time period, requiring many uses of hashing and encryption,[26] they should manually set the DVFS governor to maximum frequency for the whole duration of that activity and only reduce it to a more power-conscious mode after the communications window has closed. The "userspace" governor allows users with appropriate permissions to manually set the frequency without going in to the kernel so this is my recommendation for detailed system-level optimization.

This project has shown that Linux running on the Beaglebone offers many opportunities to adjust the performance and energy-delay product for processes on the system, allowing room for optimization geared towards security, networking,

and telecommunications in a resource-constrained environment like space. With further work on utilizing the PRUs we expect to see tremendous efficiency gains for dynamic operation in the field.

**Bibliography**


[1] Phoronix Test Suite
http://www.phoronix-test-suite.com/

[2] MiBench: A free, commercially representative embedded benchmark suite
http://www.eecs.umich.edu/mibench/Publications/MiBench.pdf

[3] Efficient Resource Provisioning in Compute Clouds via VM Multiplexing by Meng, Isci, Kephart, et al. June 2010.

[4] Renewable and Cooling Aware Workload Management for Sustainable Data Centers by Liu, Chen, Bash, Wierman et al. June 2012.

[5] ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection by Ni, Meneses, Jain, Kale.

[6] Global Educational Network for Satellite Operations
https://en.wikipedia.org/wiki/Global_Educational_Network_for_Satellite_Operations_%28GENSO%29

[7] China May Have Hacked US Satellites
http://defensetech.org/2011/10/28/china-may-have-hacked-u-s-satellites/

[8] Hacking Risks for Satellites
http://worldspaceriskforum.com/2012/wp-content/uploads/2012/03/29FELI1.pdf

[9] Satellite TV Technology
https://www.defcon.org/images/defcon-11/dc-11-presentations/dc-11-OldSkoolS/dc-11-OldSkoolS.pdf

[10] Arrests Made in TV Satellite Hacking
http://abcnews.go.com/Technology/story?id=99047&page=1

[11] AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. J)
http://www.ti.com/lit/ug/spruh73j/spruh73j.pdf

[12] Sitara AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. F)
http://www.ti.com/lit/ds/symlink/am3359.pdf

[13] AM335x Thermal Considerations
http://processors.wiki.ti.com/index.php/AM335x_Thermal_Considerations

[14] Conservation Cores: Reducing the Energy of Mature Computations by Venkatesh, Sampson, Goulding, et al. March 2010.

[15] Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction by Kumar, Farkas, Jouppi et al. 2003

[16] ARM big.LITTLE Processor
http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php

[17] Quest for Crypto Acceleration on the BeagleBone Black
http://datko.net/2013/09/22/quest_bbb_crypto/

[18] Linux Kernel Documentation: cpu-freq governors
http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt

[19] MEVBench: A Mobile Computer Vision Benchmarking Suite
http://www.eecs.umich.edu/mevbench/publications/IISWCFinalCopy.pdf

[20] Tor: Overview
https://www.torproject.org/about/overview.html.en

[21] SciMark 2.0
http://math.nist.gov/scimark2/

[22] Data Budget_V1. A spreadsheet outlining ANDESITE's data budget. May 6, 2014. Document not included (ITAR restricted).

[23] 7zip
https://en.wikipedia.org/wiki/7-Zip

[24] Parallel BZIP2 (pbzip2)
http://compression.ca/pbzip2/

[25] Lempel-Ziv-Markov chain algorithm
https://en.wikipedia.org/wiki/Lzma

[26] BUSAT Command & Data Handling Subsystem Critical Design Review. March 6, 2014. Document not included (ITAR restricted).