EXTENDING VOLUNTEER COMPUTING TO MOBILE DEVICES

BY

JEFFREY RAYMOND EASTLACK

A thesis submitted to the Graduate School

in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE

Major Subject: ELECTRICAL ENGINEERING

Minor Subject: COMPUTER ENGINEERING

NEW MEXICO STATE UNIVERSITY

LAS CRUCES, NEW MEXICO

OCTOBER 2011

"Extending volunteer computing to mobile devices" a thesis prepared by Jeffrey R. Eastlack in partial fulfillment of the requirements for the degree Master of Science, has been approved and accepted by the following:

Linda Lacey

Dean of the Graduate School

Steven Stochaj

Chair of the Examining Committee

Date

Committee in charge:

Dr. Steven Stochaj, Chair

Dr. Eric E. Johnson

Dr. Jeanine Cook

Dr. Richard Oliver

ACKNOWLEDGEMENTS

Mom

VITA

2003	Graduated from New Mexico State University
	Las Cruces, New Mexico
2002-2004	Engineering / Research Intern, IBM Microelectronics
2004	Teaching Assistant, Department of Electrical Engineering
	New Mexico State University
2006-2008	Product Engineer, Freescale Semiconductor
2008-2011	Embedded SW Engineer, Freescale Semiconductor

Professional and Honorary Societies

Institute of Electrical and Electronic Engineers (IEEE)

Field of Study

Major Field:	Electrical Engineering
Minor Field:	Computer Engineering

ABSTRACT

EXTENDING VOLUNTEER COMPUTING TO MOBILE DEVICES

BY

JEFFREY RAYMOND EASTLACK

NEW MEXICO STATE UNIVERSITY

LAS CRUCES, NEW MEXICO

October, 2011

TABLE OF CONTENTS

LIST OF TABLES

PROJECT RESOURCES

Resources, such as source code, kernel patches, porting procedures, profiling guides and the exact procedure and methodology used in this thesis project are housed online at the following web site.

http://code.google.com/p/boincmobile/

LIST OF FIGURES

ABREVIATIONS

ABSTRACT

The number of mobile computing devices (cell phones, tablets, etc) with internet access has increased to over 6 billion as of 2010. Many of these devices are equipped with multi-core systems on a chip (SoC) that include powerful graphics' processing units, multiple floating point co-processors, and SIMD vector co-processors—all running at 1GHz and higher and capable of executing billons of floating point instructions per second (GFLOPS). Furthermore, this computing power goes unused the majority of the time the device is turned on. This thesis investigates the feasibility of using the untapped computing power of mobile devices for "volunteer" computing projects in research areas such as biomedicine, climate study, particle physics, and astrophysics. For this study, a middleware platform for volunteer scientific computing was ported to an ARM/Linux platform to evaluate the performance capabilities of cell phone processors like the ARM Cortex-A8 and Cortex-A9 for the purpose of comparing them to typical scientific computing architectures such as Intel's Nehalem architecture, as well as embedded x86 processors such as Atom. In addition, the energy consumed by these processors was also considered as the performance per watt ratio was examined, showing that cell phone processors are as much as six to nine times more energy efficient than a Nehalem based processor running popular benchmarks. The performance results indicated that extending volunteer computing to mobile devices could provide a theoretical peak value of 141 ExaFLOPS of additional

computational processing power. For comparison, this peak value is more than 17 thousand times more processing power than the world's fastest supercomputer.

1 INTRODUCTION

This thesis explores the possibility of using embedded processors for scientific volunteer computing. Volunteer computing is a type of distributed computing in which computer owners donate their computing resources (such as processing power and storage) to computing projects. These computing projects usually have a public interest such as cancer research or climate prediction. Significant background on volunteer computing and embedded processors is provided to establish a baseline for this research. In addition, the processor micro-architecture of the test systems is discussed in detail. The performance results when running synthetic benchmarks and real application are also examined and discussed along with the associated energy consumption and then final conclusions are made regarding feasibility of the project.

1.1 Project motivation

For the past decade, marketplace forces of mobile handheld computing have brought about a convergence of devices, resulting in a single do-everything mobile solution. The semiconductor industry has responded to customer demand for high performance and highly connected low power systems on a chip (SoC). Advancements in wireless

connectivity continues from 3G, as 4G technologies have enabled wireless speeds of up to 128 Mbit/s downlink and 56 Mbit/s uplink over 20 MHz wide channels. Additionally, wireless internet access points have become commonplace in homes, offices, businesses, retail stores and restaurants, making the absence of an internet connection a rare event. The semiconductor performance advancements which are consistent with Moore's Law and the increasing ubiquity of convergent mobile devices have resulted in the generation of a colossal pool of untapped computing resources that could be used for volunteer computing. Volunteer computing is an arrangement where people (volunteers) provide computing resources to research projects. These projects then use the volunteered resources to do distributed computing over the internet. To evaluate the capacity of this processing power, a client version of the Berkeley Open Infrastructure for Network Computing (BOINC) was ported to an ARM/Linux platform. BOINC middleware enabled the consolidated pooling of resources for volunteer computing from a mix of computers throughout the world.

Due to the large and increasing number of mobile embedded computing devices, the possibility of extending volunteer computing to mobile devices cannot be ignored. A 1GHz ARM Cortex-A9 processor coupled with a pipelined FPU was measured to compute 648 BOINC whetstone MFLOPS per core. Thus, with overhead ignored, a 1GHz quad core Cortex-A9 based SoC such as the i.MX6Q is capable of computing an aggregate 2568 MFLOPS compared to a dual core Nehalem processor's 3204

MFLOPS running at 2.13GHz. Employing a quad core ARM Cortex-A9 coupled with multiple FPUs like i.MX6Q running at 1GHz, could add an additional 15.4 ExaFLOPS of peak processing power to the volunteer computing resource pool. In addition, and ignored in this thesis, graphics processing units such as the Vivante GPU found on Freescale's i.MX6Q SoCs are capable of computing 21 billion floating point operations per second, which could add an additional 126 ExaFLOPS of peak theoretical processing power to the same resource pool while adding only a few 100mW to the power budget of the SoC. This, of course, is an ideal scenario that is unrealistic in practice and it is discussed and dissected in the final conclusion in section 7.2.

Another motivation to extend BOINC to embedded devices is that the number of connected computing devices is increasing on a logarithmic scale as seen in Figure 1-1. By 2015, 15 billion embedded devices are predicted to be connected to the internet (1). Most of them will be ARM processors with floating point capabilities. Thus, extending BOINC middleware to embedded devices will provide the framework to scale the resource pool with the logarithmic increase of mobile devices as shown in Figure 1-1.

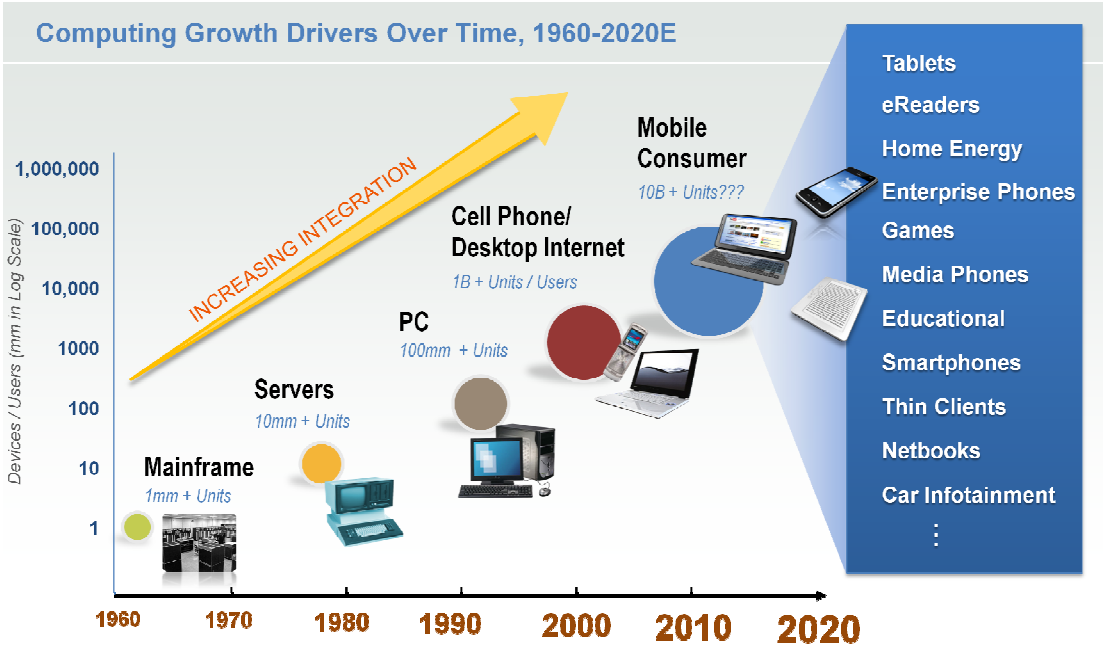

Figure 1-1- Logarithmic Increase with Connected Devices (2)

An additional motivation is that BOINC's traditional resource pool, the desktop PC, is on a path to be replaced by mobile computers such as tablets, as seen in Figure 1-2. In 2010 tablet computers displaced 6% of Desktop PC sales by 38% to 32% in just one year as seen in Figure 1-2; by 2015 Desktop PC sales are expected to drop to 18% as tablet sales are expected to climb to 23%. Additionally, these highly connected mobile handsets and tablets are currently on a very aggressive path to match the processing performance of a low end notebook PC while maintaining low power operation. Advancements in wireless connectivity technologies continue to enable this redefinition of personal computing. The embedded processor performance improvements, wireless connectivity, and the emergence of cloud

computing have enabled a wide array of new application paradigms that could now be unbound to a desktop PC and run on a mobile device. These new application paradigms are further accelerating the adoption of embedded mobile computing devices and, thus, decreasing the traditional BOINC resource pool at the same time.

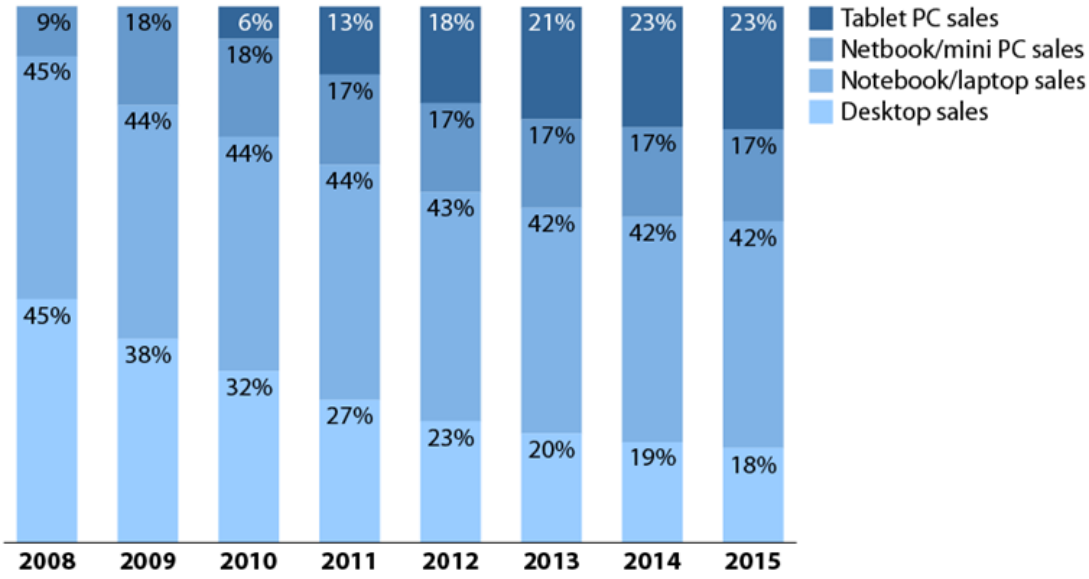

Figure 1-2 - Mobile devices cannibalizing PCs (2)

1.2 Project benchmarks

BOINC middleware supports a large variety of projects that explore various topics within multiple scientific disciplines. The BOINC client contains two benchmarks that measure floating point and integer computation which is used by the BOINC server to determine how much work to give the system. These benchmarks also provide some insight to the processing performance of the CPU cores. For this thesis,

the Search for Extra Terrestrial Intelligence at Home (SETI) is used as the BOINC supported scientific application for this evaluation. BOINC middleware provides mechanisms for anonymous platform support, where applications such as SETI can be ported to different ISAs. Both BOINC and SETI were compiled to the ARM instruction set for this project, using cross compilation tool chains. SETI is the most popular of all BOINC applications; SETI enthusiasts donated their unused computer clock cycles for radio signal processing. SETI is supported by a total of 5.2 million users worldwide with a current average of 286K active users as of September 2011, and it has the ability to compute 461 TFLOPS. SETI includes a static test work unit (WU) that can be executed in multiple concurrent instances to utilize more than one processor on a multi-core SMP system. Depending on the processing power of the system, the execution of the WU will take more or less time. During this time the power was sampled at every second, so that an approximate value for energy (in joules) can be calculated for each system executing the SETI test WU.

1.3 Architectural performance

Both ARM processors and Intel processors are equipped with performance event counters which provide statistical information regarding the frequency of architectural performance limiting events such as cache misses, TLB misses, branch mis-predictions, and instruction throughput. The Linux profiling package "oprofile" runs in the background, logging event data from the performance counters with low

overhead during the profiling period. The oprofile driver is supported in Linux kernel versions 2.6 and higher, and provides the user-space profiling application with access to the performance counters. Oprofile was used in this thesis to profile SETI and associated libraries to determine architectural bottlenecks within the test systems.

1.4 Energy consumption

The energy efficiency of mobile processors is also impressive; the i.MX6Q quad core Cortex-A9 system (used in this thesis) was realized on a Taiwan Semiconductor Manufacturing Company's (TSMC) 40nm process technology node and draws only 4.6W (average) of total system power while running four instances of the SETI test WU (one for each core). The Core i3 Nehalem architecture was manufactured on Intel's 32nm process node and pulls 34.74W (average) while running two instances of the SETI WU. Using the SETI WU as the power baseline and comparing it with the BOINC Dhrystone and Whetstone benchmarks for integer and floating point performance shows that i.MX6Q is several times more energy efficient than Nehalem in terms of a performance/Watt ratio.

The remainder of this thesis will explore the use of the modern high end embedded processor based SoC as a compute node in a volunteer computing project. Viability will be determined by examining execution time of the workload (SETI) and the associated energy used. If the execution takes too long, then more energy is

consumed by the processor and possible gaping system level data stalls are introduced as overhead into the workload execution, thus, potentially reducing throughput of other processors on the system, depending on the application.

2 BACKGROUND

2.1 Scientific computing

Scientific computing is a field of science that is mainly concerned with constructing mathematical models and quantitative analysis techniques by using computers to analyze and solve scientific problems. It is typically the application of computer simulation and other forms of computation to problems in various scientific disciplines. These models typically require massive numbers of calculations (usually floating-point) and are often executed on supercomputers or distributed computing platforms including volunteer computing platforms. Thus, the floating point and memory performance of embedded processors is critically important for these processors to make viable contributions to the aggregate scientific workload in a timely manner.

2.2 Volunteer computing

Volunteer computing uses a set of open standards and protocols to gain access to applications and data, processing power, storage capacity and a vast array of other

computing resources over the Internet. The first volunteer computing project was the Great Internet Mersenne Prime Search, which was started in January 1996 (3). It was followed in 1997 by distributed.net. The term "volunteer computing" was coined by Luis F. G. Sarmenta, the developer of Bayanihan. In 1999 the SETI@home and Folding@home projects were launched by the University of California at Berkeley and Stanford University respectively. These projects received considerable media coverage, and each one attracted several hundred thousand volunteers. In 2002, the Berkeley Open Infrastructure for Network Computing (BOINC) open source project was founded to provide a middleware framework for volunteer computing projects, and became the software running the largest public computing grid (World Community Grid) in 2007 (4).

2.3 The Berkeley Open Infrastructure for Network Computing

The Berkeley Open Infrastructure for Network Computing (BOINC) is an open source software middleware platform for volunteer computing and desktop grid computing. BOINC enables scientists to create and operate public-resource computing projects and it supports applications with diverse requirements. PC owners can participate in multiple BOINC projects as resource volunteers, and they can specify how their resources are allocated among these projects (5). BOINC was originally created to support the SETI at home project, but now it supports other areas of science as well which are explained in the next section. The BOINC development

team at UC Berkeley's Space Science Laboratory is funded by multiple NSF grants. The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world. As of September 2011, BOINC has an average performance 5.382 PetaFLOPS (average over 24hr period) that is contributed by over 286,180 active volunteer with approximately 416k computers worldwide.

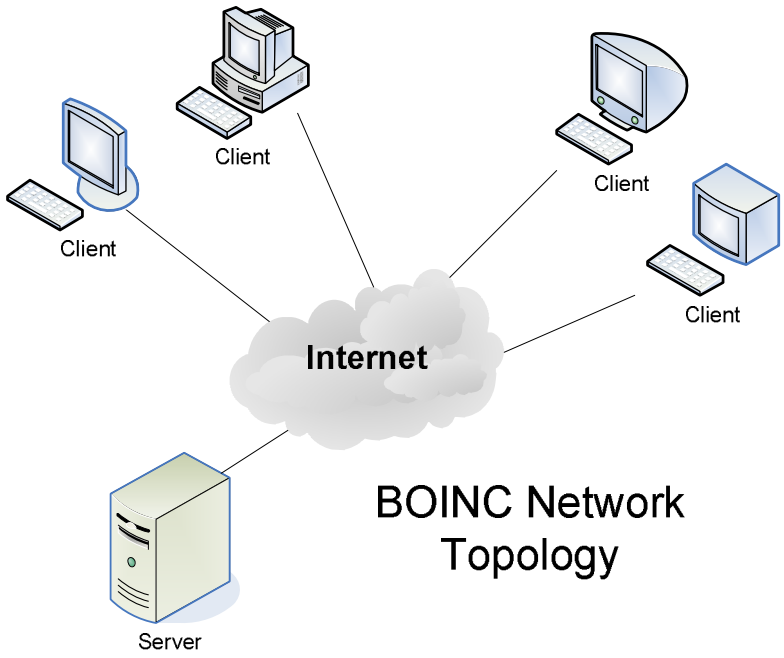

Figure 2-1- BOINC Network Topology (6)

2.3.1 The ideal BOINC application

Due to the communication overhead involved with some types of parallel processing, diminishing returns on performance are commonly observed if the parallel system spends too much time communicating with the processing nodes. With BOINC, there are at least 4 general sources of performance degradation that exist. 1) Starvation which results from not having enough work to do due to insufficient parallelism or poor load balancing among distributed resources. 2) Latency from waiting for access to memory or other parts of the system. 3) Overhead, which is the extra work that has to be done to manage program concurrency and parallel resources with the real work to be performed. 4) Contention causes delays due to tasks using a shared resource such as network bandwidth. Because BOINC taps computing resources from all over the world with varying communication speeds and processing power, it is not recommended to develop a BOINC project that requires significant inter-process communication. The ideal BOINC project has the characteristics of having what is referred to as a low data to compute ratio. Therefore, a coarse grained parallel application such as shown in Figure 1.2 is best suited for this type of organization.

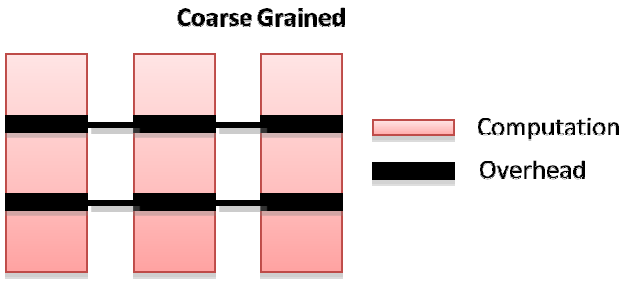

Figure 2-2- Course Grained Work Units (7)

2.3.2 How BOINC works

First, the client gets a set of instructions from the project's scheduling server. The instructions depend on the computer to be used: for example, the server won't give it work that requires more RAM than is available. The instructions may include many multiple pieces of work. Projects can support several applications, and the server may send the computer to be used work from any of them. Second, the client downloads the executable and input files from the project's data server. If the project releases new versions of its applications, the executable files are downloaded automatically by the client to the computer. Third, the client runs the application programs, producing output files. Fourth, the client uploads the output files to the data server. Finally, later (up to several days later, depending on user preferences) the client reports the completed results to the scheduling server, and gets instructions for more work. (6)

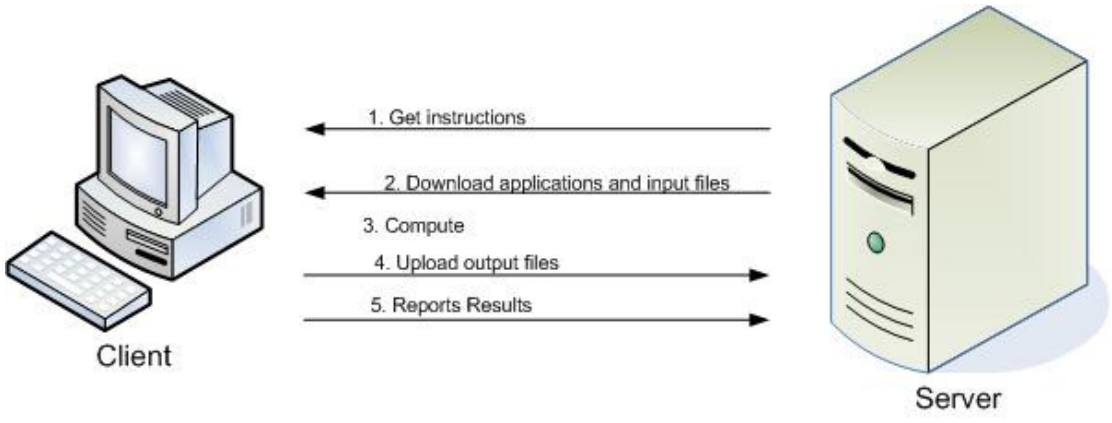

Figure 2-3 - BOINC compute sequence (6)

2.3.3 BOINC mobile topology

The BOINC model proposed in this thesis and seen in Figure 2-4 is the same as the original BOINC network topology with the exception of using mobile devices as compute nodes. The low data to compute ratio of the traditional BOINC application is especially important with this model as data transfer rates for mobile devices are not as consistent as that of a desktop PC with a wired internet connection. Mobile devices are equipped with multiple communications technologies such as 2G, 3G, 4G, WiFi, and Bluetooth, all with varying communication ranges and transfer rates. In addition, mobile devices often have periods of time where no internet connection is available at all.

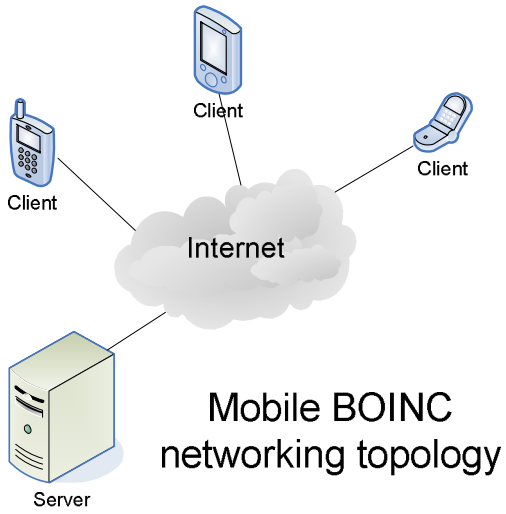

Figure 2-4 - BOINC topology with mobile devices

2.4 Introduction to embedded processors

Although the definition is changing, embedded processors have been loosely defined as processors embedded inside of devices other than conventional computers. The traditional embedded processor performs both computation and control of the device. The performance is sometimes classified as more than a microcontroller, but less than a traditional desktop PC processor. In addition to mobile devices like phones and tablets, embedded processors are used in several product domains including automotive, networking, printing, and medical devices. Embedded processors are developed with three main design criteria in mind 1) Performance, which includes parallelism, instruction encoding efficiency, CPI, clock frequency, memory bandwidth, etc. 2) Power, a function of voltage, clock frequency, die area, and

process technology. 3) Cost, which is a function of die area, complexity of design/implementation, and manufacturability. Thus, tradeoffs such as adding caches, multipliers, write buffers, TLBs, branch predictors, MMUs, and scratch pad memory are carefully weighed against the design criteria and target application domain. However, at the high end (smart phones and tablets) embedded processors have evolved into powerful processing cores with multi level caches and deep superscalar pipelines with multi issue, out of order execution as well as aggressive branch prediction with multi-core support. Furthermore, they can be used inside highly integrated SoCs running at frequencies higher than 1GHz. In addition, these SoCs include nearly all the traditional external support and logic such as IO, video, graphics, cryptography, and power management. With the high end embedded processor, the order of the day is low power, mainly due to two main criteria: battery power and die temperature. Processors running at high temperatures will accelerate the life cycle of the IC, limit reliability, and (in extreme cases) can lead to what is called “thermal runaway”. In general, if a SoC consumes less energy, it operates at a cooler temperature. Integrating logic from the system level circuit board onto the silicon die or SoC reduces the system level energy consumption, but may introduce thermal density issues on the die. Among other reasons, this reduces power with driving off chip signal parasitic with high power output buffers, and efficiency is increased when logic or IP blocks can share infrastructure. The modern high end

embedded SoC has such a high level of integration that they are sometime referred to as a cell phone on a chip.

2.4.1 Embedded processor bottlenecks

Embedded mobile processors such as the ARM Cortex-A8 and Cortex-A9 have constraints that are dictated by the embedded market and, thus, influence configurations that can limit performance and thus energy consumption. ARM develops IP processor cores that are typically licensed by vendors like Freescale, Nvidia, and Texas Instruments to be used in a highly integrated SoCs. Embedded SoCs have tight mobile power budgets and several cost constraints, including die size and the number of pins on the package. SoCs with dense pin placement on the package typically require additional circuit board layers to route all the signals out of the package. This increases manufacturing complexity and, thus, adds cost. High end embedded SoCs have dozens of integrated logic blocks such as on board, USB, SATA, SDHC, VPU, CSPI, ATA, FEC, SPDIF, GPIO, UART and several other IO controllers seen in the block diagram of i.MX6Q in Figure 4-6. As such, each IO signal at the SoC level is multiplexed by as much as 8-16 ways, which makes the pin availability very limited. Additionally, they have as many as 30-50 separate power domains to service the increasing level of integration with IP blocks that require specific voltage and power inputs. These separate power domains require dedicated power pins to isolate the voltage levels. As a result of this pin scarcity, the main

memory channels are usually limited to 32, and sometimes 64 bits wide and commonly interface to the main CPU through a series of on-chip arbiters. Thus, the latency penalties to main memory are higher and bandwidth is lower on embedded SoCs than a server architecture like Nehalem. In addition to the restricted number of pins for memory channels, these same memory channels typically cannot operate at frequencies much higher than 400MHz because of transmission line effects. Transmission line effects occur on memory busses when the characteristic impedance of the signal line is mismatched with the target and generator impedance. Data signals in the form of electric current behave as electromagnetic waves transmitting through a medium. When the impedance of the medium changes with respect to the traveling wave, a reflection co-efficient of the wave gets reflected back to the generator and causes the voltage level of the line to adhere to the superposition principle. Signal data busses with mismatched impedances introduce signal integrity issues at high frequencies. Line impedance can be easily matched by implementing terminating resistors on each of the data signals. Terminating resistors are usually placed in parallel to the input pin of the target memory chip with the other end of the resistor connected to ground with a low resistance to match the characteristic impedance of the line. System level power consumption becomes a major concern when there are wide data and address busses connected with parallel terminating resistors that have a low impedance path to ground. As a result of this power drain, mobile devices

typically do not use parallel terminating resistors and thus, the memory bus frequency is usually much lower than a standard desktop PC as a means to save energy.

The memory hierarchy is also influenced by die size and the associated leakage power. Smaller die size results in higher wafer yield, and, thus, higher profit margins. Thus, TLBs and Level 1 (L1) and Level 2 (L2) cache memory and other performance improvers are limited in order to increase profits and limit leakage current which can drain the battery of the mobile device faster. This means that TLB misses and L2 misses will be more frequent and carry a greater penalty when accessing main memory than a conventional processor system. However, ARM cores are very flexible and configurable; the Cortex-A9 has the capability to expand from 1 to 4 cores with L2 cache sizes from 512k to 4MB, depending on vendor requirements. Thus, performance on one ARM Cortex-A9 based SoC could have different performance results than another Cortex-A9 based system running at the same clock frequency for certain applications. Because of this, the project benchmark results are examined across three different Cortex-A9 based systems as well as a Cortex-A8 based system, a dual core Atom system, and a dual core (Core i3) Nehalem based system.

2.4.2 Embedded processor architectures

The three leading architectures in the embedded processor space are ARM, MIPS, and Power (formerly Power PC). All three employ native RISC architectures and

have established respective dominance within specific product domains. ARM's superior energy efficiency has allowed them to dominate the portable battery operated embedded processor space such as cell phones and tablets. ARM architectures are one of the few if only architectures that can deliver high processing performance and still operate within what is called the mobile power envelope which is under 5W of continuous draw. As such, approximately 98% of all cell phones and tablets employ the use of ARM based SoCs. MIPS dominates the mid range embedded processor space such as Set top boxes and DVD players. Among other products, Power architecture can be found in high end enterprise networking equipment. Inevitably, Intel moved into the low power embedded processor space with the introduction of the Atom processor as the definition of the personal computer is changing toward the mobile realm and embedded processors outsell PC processors by approximately 50 to 1.

3 PROJECT EVALUATION PROCESSORS

In this thesis the Cortex-A series processors are evaluated as they are coupled with the NEON SIMD engine and FPU co-processors and are fully IEEE754 compliant in hardware. The embedded Intel solution to be evaluated is the dual core Atom D525 which uses the standard Atom architecture with a system platform that has similar low power goals as the ARM based development kits. To provide a performance

baseline with regard to scientific computing, this thesis includes Intel's incumbent server architecture code named Nehalem, which was realized inside a Core i3 processor as a 32nm die shrink called Westmere.

3.1 The ARM Cortex-A8

The ARM Cortex-A8 was designed to meet the increasing performance demands of emerging mobile devices without exceeding the power and temperature budgets of a typical mobile device. The Cortex-A8 is a 2-way superscalar processor with a 13 stage integer pipeline. It has a clock frequency range between 600MHz and 1.2GHz with a TSMC 90nm to 65nm process technology respectively (8). The 13 stage pipeline is considered very deep for an embedded processor as with it comes a hefty penalty for branch mis-predictions which is also a 13 stage penalty. To minimize branch wrong prediction penalties, the dynamic branch predictor achieves 95% accuracy across a wide range of industry benchmarks by employing a branch target buffer (BTB), a global history buffer (GHB) and a return stack. The Cortex-A8 integer unit is responsible for the program flow, fetching instructions from memory, and writing data back to memory. The core is a dual in-order issue, statically scheduled, 2 way superscalar processing core. The integer decode unit maintains a scoreboard with one entry per ARM integer register, which keeps track of when and where that register value is being produced in order to implement the static scheduling. The Cortex-A8 scoreboard can determine when an operand will become

available several cycles in the future. It can also compare that time with the cycle when the operand itself will be needed in the future, and not just whether a register is currently available. This static scheduling provides timing benefits since hazards can be resolved before instruction issue. Aside from timing benefits, this provides energy saving capabilities, as well, since advanced knowledge of the execution stages involved in each pipeline cycle can be used to clock gate unused logic. A replay queue is used in the case of a pipeline flush due to a memory system stall event; it stores in-flight instructions that are reissued from the replay queue when ready. The time penalty with flushing the execution pipeline is balanced to match the minimum latency of 9 clock cycles from the L2 Cache. Identifying NEON and VFPv3 instructions is done by the instruction decode unit, and are simply passed down as no-ops through the integer pipeline execution units E0-E5.

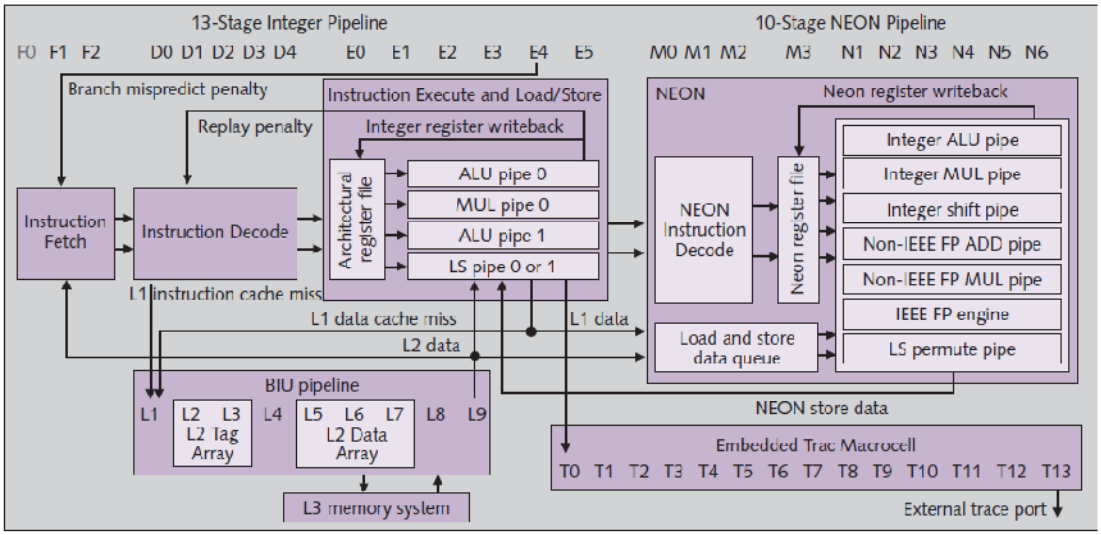

Figure 3-1 - ARM Cortex-A8 Pipelines (8)

Both Instruction and Data L1 caches are configured as four way set-associative resources employing a Hash Virtual Address Buffer (HVAB) to improve timing and energy consumption. The algorithm is capable of turning on clocking to the appropriate "way" of the cache. The replacement policy for the L1 data cache is write-back with no write allocates. The L2 cache is shared by both instructions and data misses by the L1 data and instruction caches. An L2 hit results in an L1 miss penalty of 9 clock cycles, plus the 1 cycle access to the L1 for a total of 10 clock cycles. The L2 cache employs a write allocate replacement policy (8).

The Cortex-A8 employs a 128-bit SIMD engine called NEON that is pipelined with 10 stages and enables high performance media processing for Audio, Video, and 3D Graphics workloads. Coupled with the Vector Floating Point unit (VFP), the Cortex-

A8 is fully IEEE 754 compliant in hardware, and requires no software emulation of the IEEE754 subnormals by software in the integer pipelines like its predecessor the ARM11. However, the Cortex-A8 VFP is not a pipelined unit for IEEE754 double precision floating point operations as shown in Figure 3-2. In the case of non-IEEE754 single precision floating point, operations can be routed to the NEON co-processor in "run-fast" mode for increased throughput in the via the NEON 10 stage pipeline (8). As seen in Figure 3-2 the NEON co-processor is equipped with multiple types of pipelines which include an integer MAC pipe, a shift pipe, an arithmetic conversion pipe, two floating point DSP/Graphics pipes (one for addition and the other for multiply), a one stage vector floating point unit (VFP), and a load / store and permute pipeline.

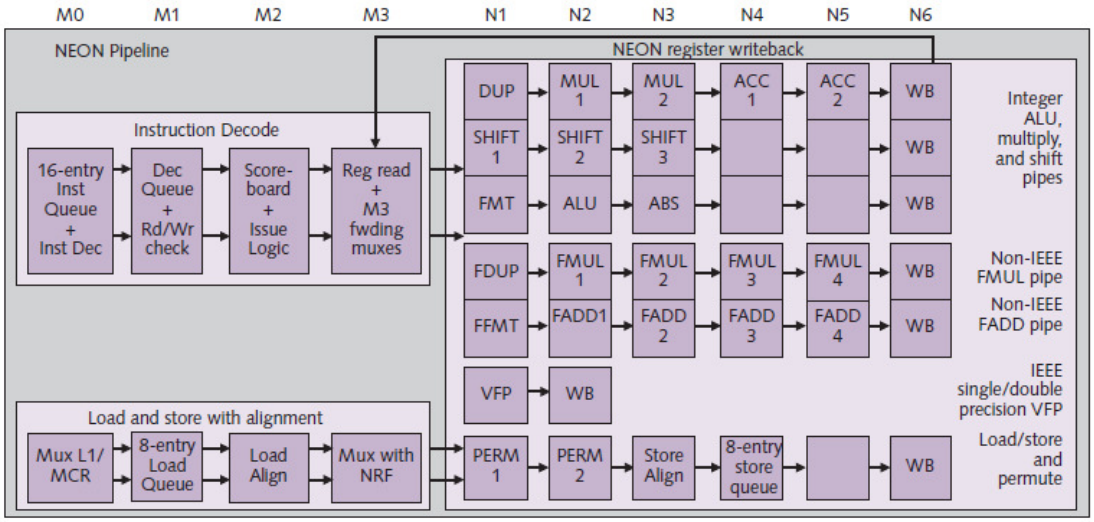

Figure 3-2 - ARM Cortex-A8 NEON Pipeline (8)

3.2 The ARM Cortex-A9

The Cortex-A9 is the successor of the Cortex-A8 with several performance enhancing capabilities in both the integer execution units and as well as the NEON and floating point units VFPv3 which make it much better suited for scientific computing than the Cortex-A8. The integer core is two-way superscalar like the Cortex-A8, only the pipeline is much shorter with only 8 stages which reduce the penalty of a branch mis-prediction and a pipeline flush. The Cortex-A9 employs out of order execution with speculation and register renaming; it can also issue between 2 and 4 instructions per cycle, and is capable of operating at up to 2GHz with a semi-custom hard macro block tailored to TSMC's 40nm GP process. The instruction set of the cortex-A9 also implements the ARMv7 ISA as well as the VFPv3 instruction interface. In addition, the Cortex-A9 has multi-core capabilities supporting up to 4 cores. The name Vector Floating Point Unit (VFP) has been deprecated and is now referred to as the Floating Point Unit (FPU); however, VFPv3 still refers to the instruction set architecture (ISA). Another important distinction between the Cortex-A8 and the Cortex-A9 is that the NEON co-processor is now a component of the Media Processing Engine (MPE). The MPE is composed of both the NEON and FPU and is a mutually exclusive option with the FPU. In other words, it is possible to have the FPU or the MPE (which includes the FPU) in the Cortex-A9, but not the NEON by itself. The Cortex-A9 supports up to four instruction cache lines prefetch-pending which is

intended to reduce the impact of memory latency by maintaining instruction delivery. The core issues between two and four instructions per cycle into instruction decode units to maintain optimal pipeline utilization. The two-way decoder is capable of decoding two full instructions per cycle. The core is capable of dynamically renaming physical registers from an available pool of virtual registers for speculative execution of instructions and increased pipeline utilization by removing data dependencies between adjacent instructions. The virtual renaming of registers also allows the acceleration of code through a hardware based unrolling of loops while still maintaining the code density of rolled loops. Any of the four subsequent pipelines shown in Figure 3-3 can select instructions from the issue queue, providing out of order dispatch to increase pipeline utilization via dynamic scheduling without static scheduling by the compiler. This enables greater performance optimization from previous versions of the ARM ISA as a means to leverage existing code investment with smarter pipeline scheduling. The Cortex-A9 allows concurrent execution across its dual arithmetic pipelines, load-store or compute engine, plus resolution of any branch at every cycle. The Cortex-A9 supports up to four data cache line fill requests to reduce stalls due to memory latency with either automatic or user driven pre-fetching to increase the availability of pending data.

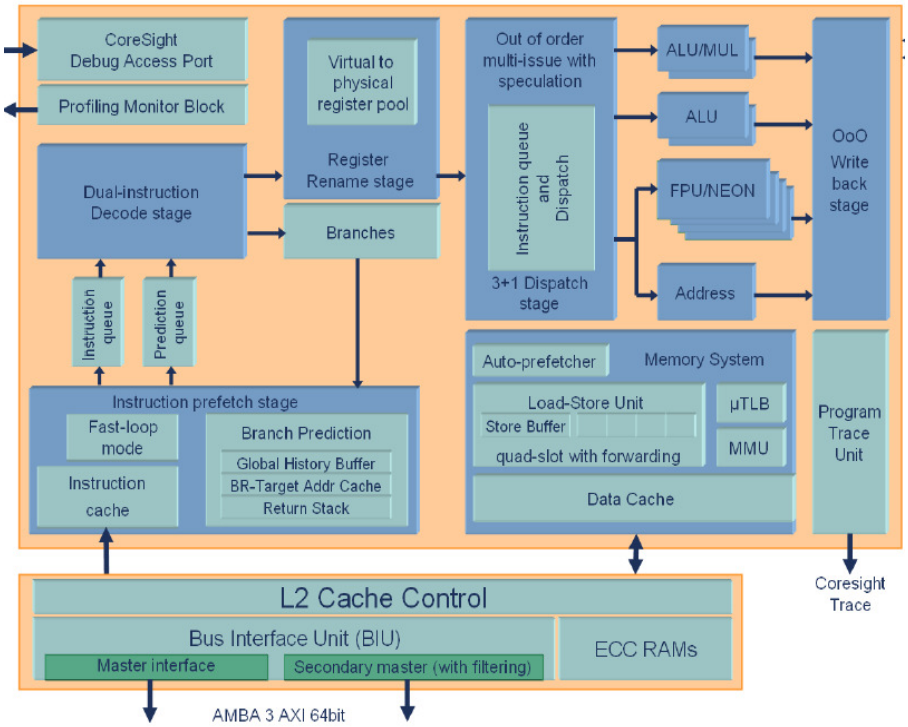

Figure 3-3 - ARM Cortex-A9 Block Diagram (9)

The Cortex-A9 MPCore multicore processor includes an enhanced version of the ARM MPCore technology first introduced in the ARM11. The Cortex-A9 MPCore provides the flexibility for vendors to implement between 1 and 4 CPUs in a cache coherent Symmetric Multiprocessor (SMP) system architecture design. Each processor can be configured independently for its cache size, and FPU, MPE and PTM interfaces. The L1 Caches are configured as 4-way set associative with flexible density options from 16k, 32k and 64k. The processor in any configuration may expose the Accelerator Coherence Port (ACP) to other non-cached system mastering peripherals and accelerators such as a DMA engine or cryptographic accelerator core

to be cache coherent with the L1 processor caches. The processor can support either a single or dual 64-bit interconnect interface. The ACP provides an interconnect point for a range of system masters that for overall system performance, power consumption or reasons of software simplification are interfaced directly with the Cortex-A9 MPCore processor.

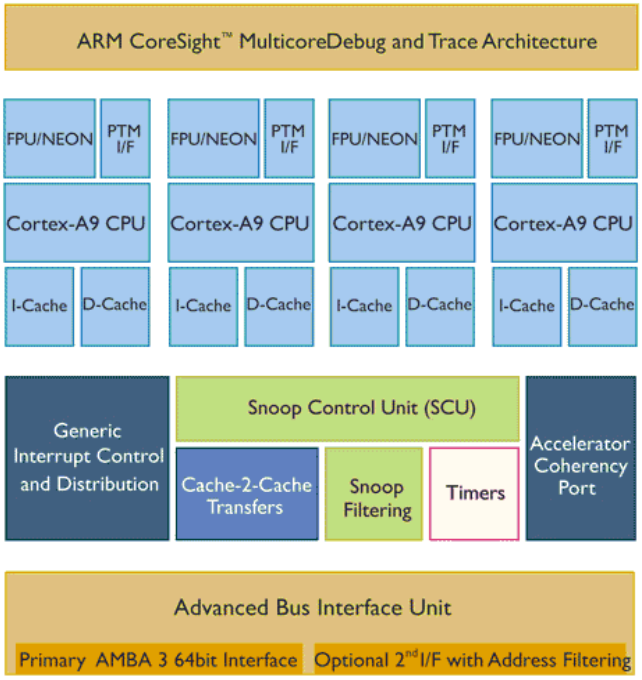

Figure 3-4 - ARM Cortex-A9 MP Block Diagram (9)

The L2 cache controller is capable of supporting multiple outstanding bus transactions on each interface, with premaster per-way lockdown to allow managed-sharing between multiple CPU or components using the ACP and effectively using the L2 cache controller as a buffer between accelerators and the processors. The

Cortex-A9 supports up to 8 MB of L2 cache memory, with between four and sixteen-way associativety. The L2 cache controller supports the optional integration with both parity and ECC supporting RAM and is capable of operating at the same frequency as the processor. The snoop control unit (SCU) is not only responsible for managing cache coherence, but also the interconnects, arbitration, communication, cache to cache, and system memory transfers. MPcore logic is also extended to other system accelerators and logic blocks and non-cached DMA driven mastering peripherals to reduce the SoC system level power consumption and improve performance by sharing access to the processor's entire memory hierarchy, including the L1 caches as shown in Figure 3-4. This interface acts as a standard bus slave, and supports all standard read and write transactions without additional coherence requirements placed on attached components. However, any read transactions to a coherent region of memory will interact with the SCU to test whether the required information is already stored within the processors L1 caches. If it is, it is returned directly to the requesting component. If it missed in the L1 cache, then there is also the opportunity to hit in L2 cache before being forwarded to the main memory. In the case of Write transactions to any coherent memory region, the SCU will enforce coherence before the write is forwarded to the memory system. The L2 can be configured either as write through or write back mode to give additional power saving options as writing to off chip memory carries with it a heavy power penalty.

The optional FPU provides high performance pipelined single, and double precision floating point instructions compatible with the ARM VFPv3 ISA which is binary compatible with previous generations of FPUs (formerly VFPs) such as on the ARM Cortex-A8. The Cortex-A9 FPU is fully IEEE754 compliant in hardware operating for the first time at the same speed as previous "run-fast" modes as its execution is now pipelined, so it now operates with no trapped exceptions, simplifying software and further accelerating the performance of floating point code.

The Cortex-A9 MPE can be used with either of the Cortex-A9 processors and provides an engine that offers both the performance and functionality of the Cortex-A9 FPU plus an implementation of the ARM NEON Advanced SIMD instruction set that was first introduced with the ARM Cortex-A8 processor for further acceleration of media and signal processing functions. The MPE extends the Cortex-A9 processor's FPU to provide a quad-MAC and additional 64-bit and 128-bit register set supporting an enhanced set of SIMD operations over 8, 16 and 32-bit integer and 32bit floating point data quantities every cycle. Further enhancing the SIMD capability, the MPE also supports fused data types to remove packing/unpacking overheads and structured load/store capabilities to eliminate shuffling data between algorithm-format to machine-formats. Utilizing the MPE also enlarges the register file available to FPU and increases the design to support 32 double-precision registers while retaining the Cortex-A9 processor's 32/64-bit scalar floating-point and core integer performance.

3.3 Intel's Embedded Processor - Atom

Atom is a clean sheet micro-architecture; one of the only things it has in common with other Intel processors is that it is fully x86 compatible (10). Thus, out of the gate Atom architects must overcome a serious power handicap in the embedded space. Binary compatibility introduces many challenges to designing a low power embedded processor; however, Intel has learned in the past with the Itanium processor that this legacy binary compatibility is crucial for commercial success. Atom targets low-cost subnotebook computers, low-cost desktop PCs, and mobile Internet devices (MIDs). Intel refers to the subnotebooks as netbooks and mini desktop PCs are nettops to emphasize their integrated wireless Internet connectivity. Atom designers decided to discard x86 instruction transformation from CISC instructions to RISC like micro-ops in most cases to save energy. Thus, Atom will execute most x86 instructions in their original form, but they will be diverted into a "microcode sequencer" for special decoding into RISC like operations at the penalty of two clock cycles. In a few cases x86 instructions combine multiple operations; for example, ADD-Store instructions are issued in pairs through the dual pipelines as if they were separate micro-ops. They execute simultaneously, in lockstep without the ability to bypass each other as traditional micro-ops can do in other x86 pipelines (10). However, the instruction pair

occupies only one slot in the 32-entry instruction dispatch table, thus, retaining the code density benefit of CISC instructions.

The Atom core has two instruction decoders that can decode complex types of x86 instructions that other Intel x86 processors would transform into three micro-ops. Atom pre-decodes x86 instructions by tagging them with a one-bit marker, indicating the end of each instruction which could be from 8 to 120 bits. In two predecode stages of the pipeline, the control logic marks the instruction boundaries and stores the instructions in the L1 cache. As a result, instructions that are fetched from the cache are already marked and can bypass those two stages. The instruction cache is 36KB, but the effective size is about 32KB because the boundary tags occupy about 4KB. Except for this tagging, cached instructions are un-decoded; they must pass through three additional pipe stages for full decoding.

The Atom has a 16-stage basic integer pipeline. Note that the two pre-decode stages are excluded from the count. In practice, the pipeline varies from 16 stages to 19 stages, depending on circumstances of the instruction stream. Sometimes, a cache miss forces the processor to spend three clock cycles finding the end of an instruction whose boundary hasn't been tagged yet. In some of those cases, the processor can simultaneously decode a preceding instruction without a penalty, preserving the

nominal 16-stage pipeline shown in Figure 3-5. And in any case, after a cache hit, pre-decoded instructions pay no penalty. For most purposes, it's a 16-stage pipeline.

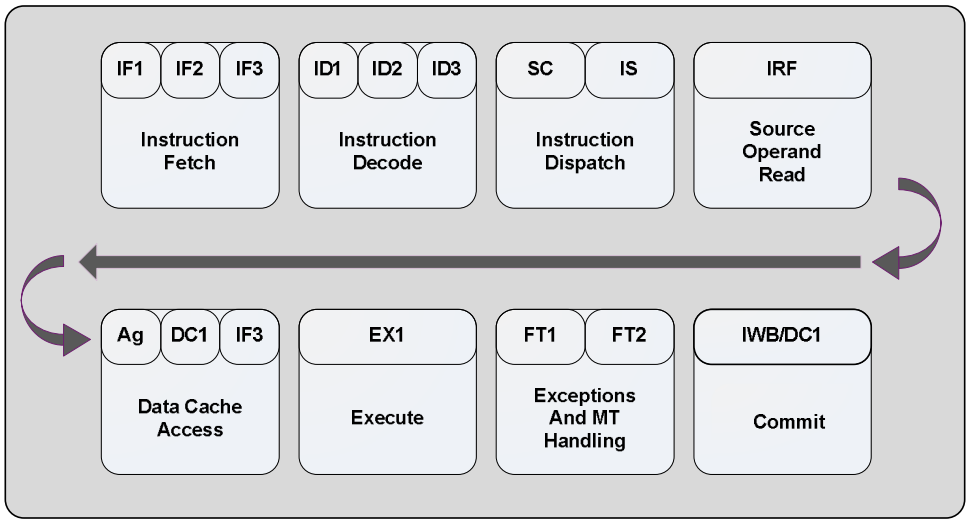

Figure 3-5 - Atom Pipeline (10)

Atom is a two-way super-scalar in-order execution processor with simultaneous multi-threading (SMT) which can increase pipeline utilization and performance which consecutively reduces the energy consumed by toggling the execution of the programs during stall states. This effectively increases instruction throughput without much additional logic which, in turn, helps keep leakage under control as leakage is a large portion of power consumption at deep submicron process nodes. The core maintains duplicate register files and other resources for storing state of each execution thread, so a context switch by the operating system doesn't require flushing

and restoring the registers, status flags, and stack pointers. Both threads share the same 64-entry TLB, which is two-way set-associative. Atom is also equipped with a 16 entry for each thread in a supplemental micro-TLB. Intel doubled the size of the instruction cache's prefetch buffer to reduce other resource contentions during multi-threading, so each thread has three 16-byte buffers. According to Intel, the dual threading boosts Atom's performance by 36% to 47% across several unknown but popular benchmarks with a power penalty of 17% to 19% across the same benchmarks while only adding 8% more area to the die size (10). As we will see in chapter 6, Atom performed much better than this estimate while exploiting concurrency of the SETI WU with its multithreading capabilities.

As seen in the block diagram of Figure 3-6, Atom is equipped with dual ALUs, dual FPUs, dual address-generation units (AGUs), branch predictors, caches, and TLBs.

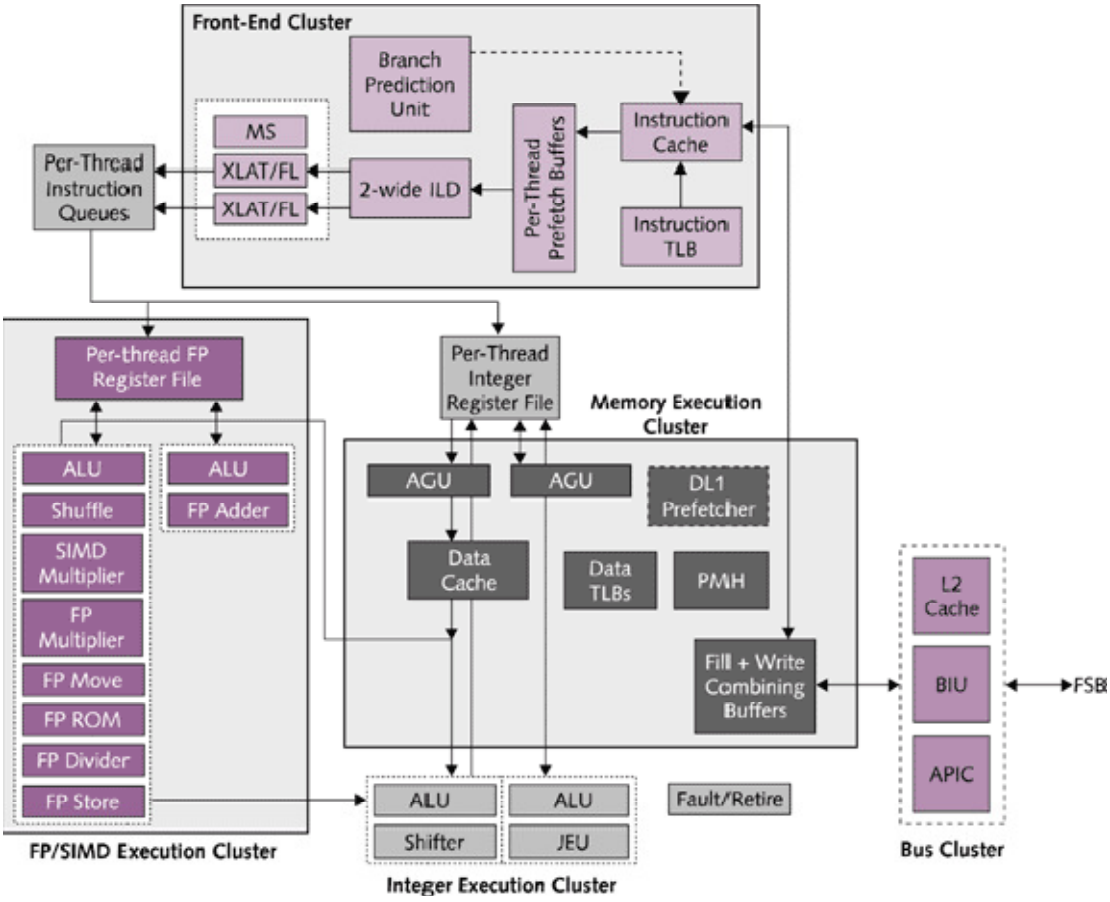

Figure 3-6 - Atom Block Diagram (10)

Like its ARM counter parts, Atom employs multiple strategies and methods to effectively nickel and dime energy consumption to a lower value. Parts of the L2 can be shut down to reduce both dynamic power and static leakage. Atom also uses its version of dynamic voltage frequency scaling (DVFS) to lower the core voltage and frequency depending on CPU load. The L1 I/D caches are built with eight-transistor cells and are larger than standard 6T memory arrays built with 6 transistors. However,

they operate at lower voltage and use less dynamic power. The 512KB L2 uses the standard 6T cells, and has a programmable set-associativety from two ways to eight ways to provide additional power savings. As seen in Figure 6-10, Atom supports multiple power modes that range from full power C0 to deep sleep mode C6. C6 represents 1.6% of max power (C0), C4 is 12%, C1 is 40%, and, thus, C0 is 100%. When Atom enters into the C6 state, the processor saves the values in the registers, status flags, stack pointers, and program counters into an internal memory location and then stops the clocks, shuts down the FSB, and goes to sleep (10).

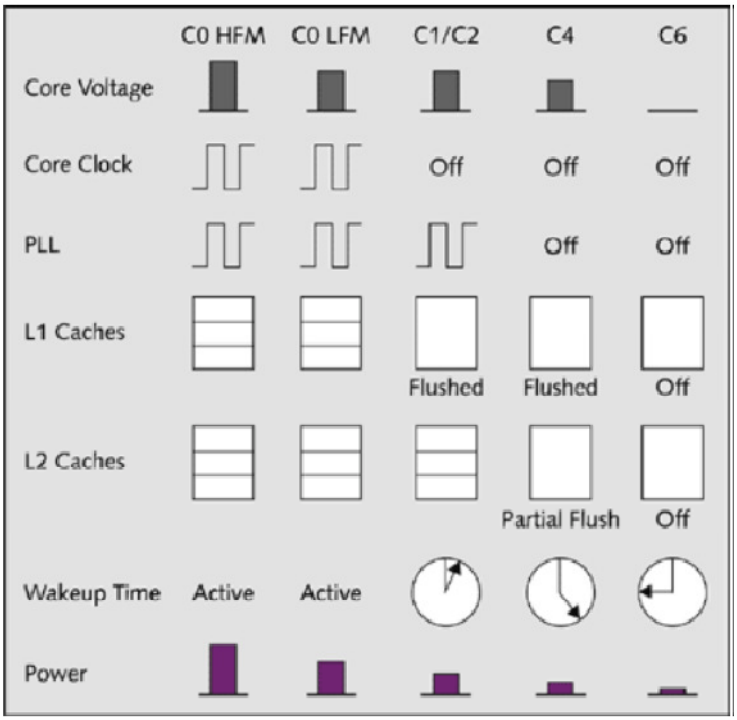

Figure 3-7 - Atom Power States (10)

3.4 Intel's Server Architecture – Nehalem

This evaluation includes a modern x86 based server architecture codenamed Nehalem to provide additional perspective into the performance of embedded processors for scientific computing. The intent is to provide an upper baseline with regard to computational performance. The Nehalem micro-architecture is used in modern Xeon processors and Intel's line of Core-i series processors. The "Core i3 330M" dual core processor is a low power Westmere processor, which is a 32nm die shrink of Nehalem developed for laptops (11). It features dual multithreaded cores with three levels of cache memory with the last level (L3) shared between the two cores seen in Figure 3-8. Nehalem is the successor to the "Core" architecture with the intent to improve core-to-core communication by establishing a point-to-point topology which allows direct communication between microprocessor cores (12). The multi-core "Core" processor implementation used the memory bus for core-to-core communication which introduced serious bandwidth issues due to overhead, thus, limiting scalability. By implementing the point-to-point communications topology, the Nehalem architecture frees up the memory system for additional bandwidth. Although some high end eight core server implementations of Nehalem utilize up to six 64bit memory channels, the dual core implementation in the "core i3" processor in only equipped with two channels labeled DDR3-A and DDR3-B as seen in Figure 3-8. Each core of the "Core i3" 330M processor has a 32kb Instruction/Data L1

cache, and a 256kb L2 with access latencies of 4ns and 11ns respectively. It also has a 3MB L3 cache that is unified or shared between each core and has an access latency of 36ns.

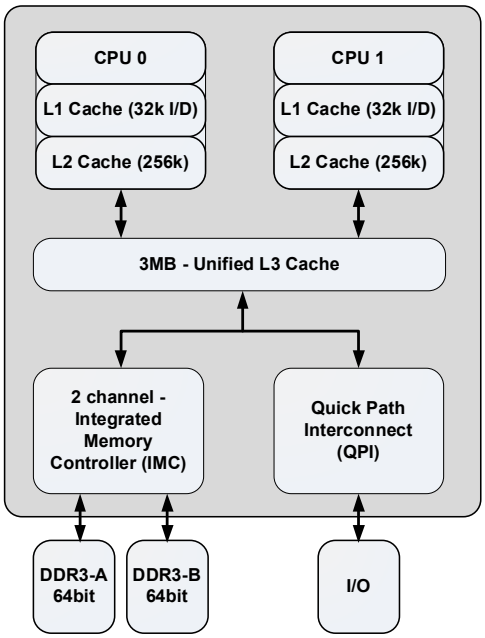

Figure 3-8 - Dual Core Nehalem Processor (Core i3 330M)

Another enhancement to the Nehalem architecture is with the branch prediction. On the "Core" architecture, Intel designed a logic block that they called the loop stream detector which essentially detected loops in execution and saved the instructions to a dedicated buffer, so that they would not have to be continually fetched from cache memory. With Nehalem they took this a bit further and placed the loop stream detector after the instruction decode stage, thus, eliminating that pipeline stages from the loop iteration as seen in Figure 3-9. Nehalem also increased the size of the re-

order buffer to allow more instructions to be ready for immediate execution. However, the main improvements with the Nehalem architecture are with the cache and memory system. The Northridge chipset has been often cited as a bottleneck in previous Intel architectures. On Nehalem, the DRAM controller, the L3 cache and the quick path interconnect (QPI) ports are all housed on the same die as the other cores. This saves a significant amount of off-chip communication and allows the multiprocessor system to be high bandwidth, tightly coupled and with low-latency (13). The TLB, which is a high speed buffer that maps a virtual address to a physical address in the cache or memory, has been significantly increased with the Nehalem architecture (12). When TLBs are too small, TLB misses are more frequent and carry a high performance penalty as the operating system needs go get involved to look up the physical address for the virtual address. This penalty can be in excess of hundreds of clock cycles. Thus, TLB size and performance are very important with applications that work on large data sets. Because of this, Intel designers implemented an additional TLB level to the micro-architecture which is identified as L2-TLB in Figure 3-9 and can hold 512 entries. In addition to larger and multilevel TLBs, Intel developed the QPI bus as a way to connect processors to each other in a point-to-point connection as well as the bridge to the IO subsystem.

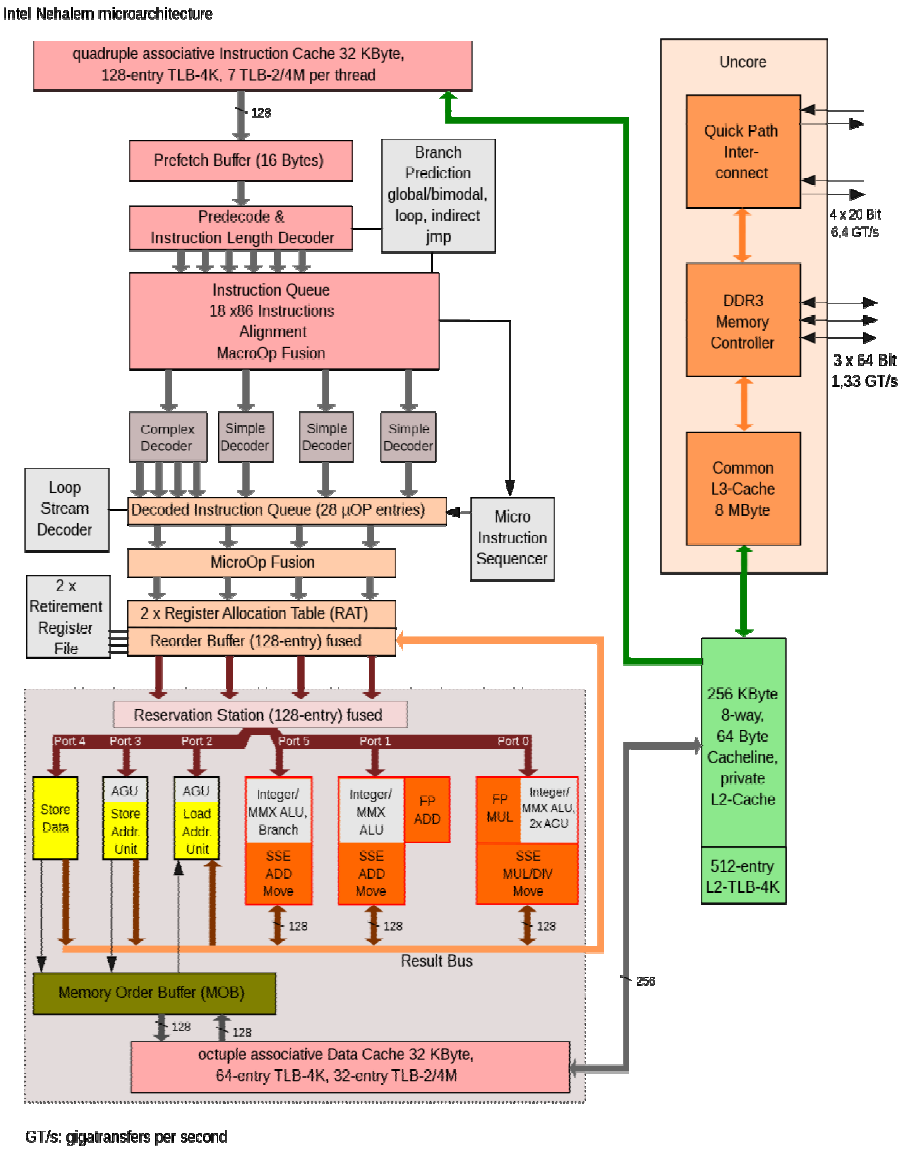

Figure 3-9 - Nehalem Block Diagram (13)

The Nehalem architecture supports modern microprocessors with in-order instruction issues, superscalar pipes, and out-of-order execution data paths that are coupled to a

multilevel memory hierarchy. It has extensive support for branch prediction, speculation, pre-fetching and multiple pipelined functional units (FUs). Nehalem has direct hardware support for integer and floating point SIMD instructions. It also uses an in-order Front-End Pipeline (FEP) shown in Figure 3-10 which retrieves Intel64 instructions from memory, uses four decoders to decode them into micro-ops and buffers them for the downstream stages (13). The branch prediction unit is part of the FEP and like other branch prediction units allows the processor to begin fetching and processing instructions before the outcome of a branch instruction is determined. The BPU makes predictions for direct calls and jumps, indirect calls and jumps, and conditional branches. The branch target buffer (BTB) has been increased in size to improve the accuracy of branch predictions. The BPU also includes a Return Stack Buffer (RSB).

The Execution Engine (EE) can dynamically schedule and dispatch up to six micro-ops per cycle to the execution units as soon as the source operands and resources are available. The in-order Retirement Unit (RU) ensures the result of the micro-op execution is updated to the original program order. The maximum pipeline depth is 16 stages and can be shortened during loop instructions with excluding stages of the FEP by executing instructions that have already been decoded and stored in the loop stream buffer.

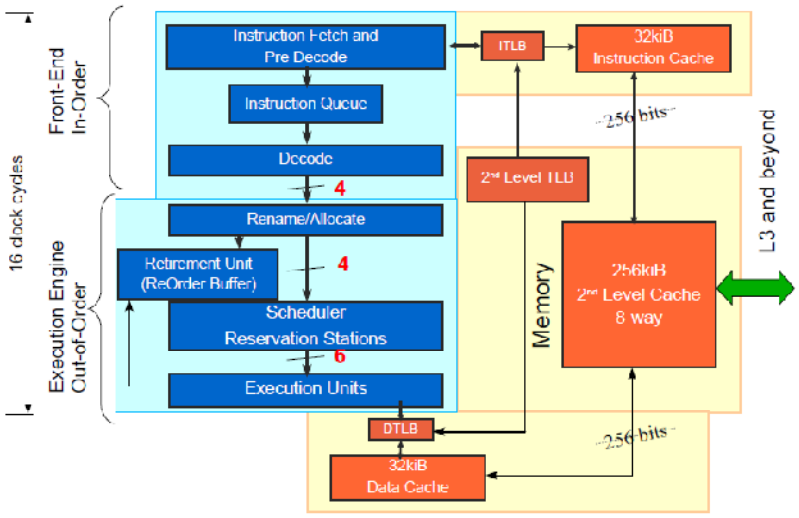

Figure 3-10 - Nehalem Pipeline Overview (13)

4 EVALUATION SYSTEMS

This thesis project uses several evaluation systems that are based on the processors discussed in the previous section.

4.1 Cortex-A8 - Freescale's i.MX53 SoC

Freescale's i.MX53 applications' processor was used for the evaluation of the ARM Cortex-A8. As seen in Figure 6-3, the SoC is highly integrated with dozens of standard connectivity protocols, multimedia logic, etc. For the purpose of this project, the focus is on the processing core, NEON and VFP coprocessors, the memory hierarchy (including the TLB), and connectivity hardware. As seen in Figure 4-1, this particular Cortex-A8 based SoC is equipped with 32KB for the Instruction and Data

L1 cache memory, and 256KB of L2 memory. It also has the standard NEON and VFP units as shown in figure Figure 4-1.

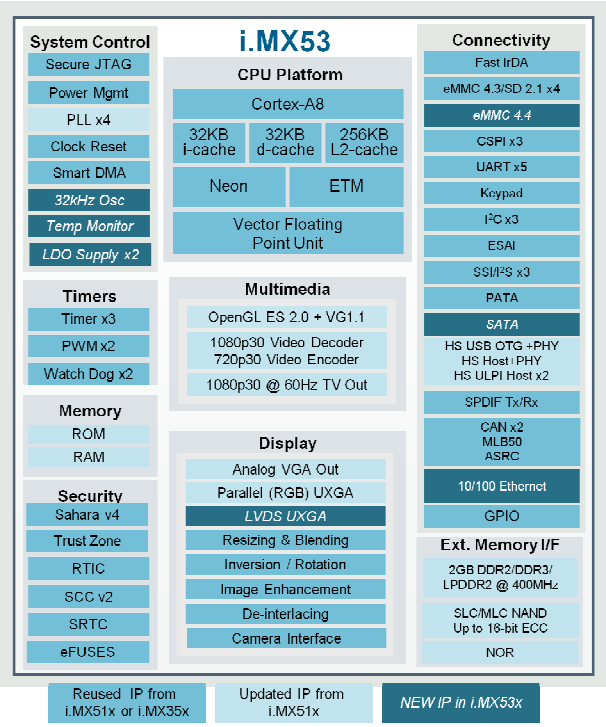

Figure 4-1 - i.MX53 Cortex-A8 based SoC (2)

At the system level the i.MX53 Quick start board (QSB) is running at 1GHz and is equipped with 1GB of DDR3, an Ethernet port, and various other IO such as USB and UART. The i.MX53 QSB is fed by a 5V power supply at the board level which is distributed down in several voltage levels by an off chip power management IC to the various power domains on i.MX53 and the QSB. The bootloader, Linux kernel

(version 2.6.38), and root file system are located and ran from the micro SD slot shown in Figure 4-2.

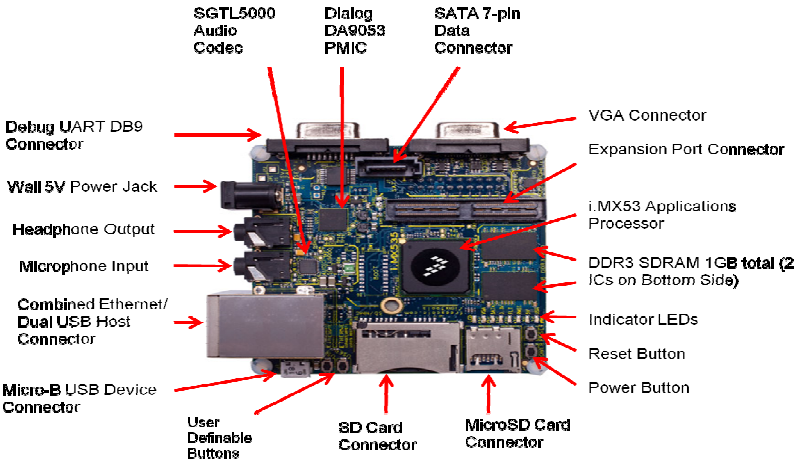

Figure 4-2 - Freescales i.MX53 Quick Start Board (2)

4.2 Cortex-A9 - ARM's Versatile Express FPGA

The versatile express is an FPGA based platform developed and supported by ARM as an evaluation platform to profile code execution and enable pre-silicon code development for a Cortex-A9 platform. It features an early quad core implementation of the Cortex-A9 codenamed Falcon and is revision r0p1. As the logic real-estate on an FPGA is limited and the Cortex-A9 is flexible; the implementation is realized with a 32kB L1 I/D cache, and a unified 512KB L2 shared by all cores. The Falcon Testchip on the Versatile Express is the r0p1revision of the Cortex-A9 and is

equipped with a multilevel TLB hierarchy with the first level as the micro-TLB or (uTLB) with both instruction side and data side with 32-entries each and are implemented as registers. The larger or main second level TLB has 64 entries per CPU and is configurable 64/128-entry RAM based TLB lookup into L1 D-cache (enabled by the operating system). The Versatile Express has 1GB of DDR2 that is connected to a 32bit bus, and running at 250MHz. Because any FPGA implementation comes with multiple layers of EDA bloat, the Cortex-A9 implementation on the Versatile board can only be run at a max frequency of 400MHz. As a general rule of thumb, architectures implemented on FPGA can run half as fast, and use twice the power. As a result, the power data discussed in future sections excludes the versatile board from the power benchmarks. However, the quad core Cortex-A9 implementation in the Versatile Express is architecturally accurate when running a workload and, thus, useful in this thesis when performance events are evaluated in section 6.2.

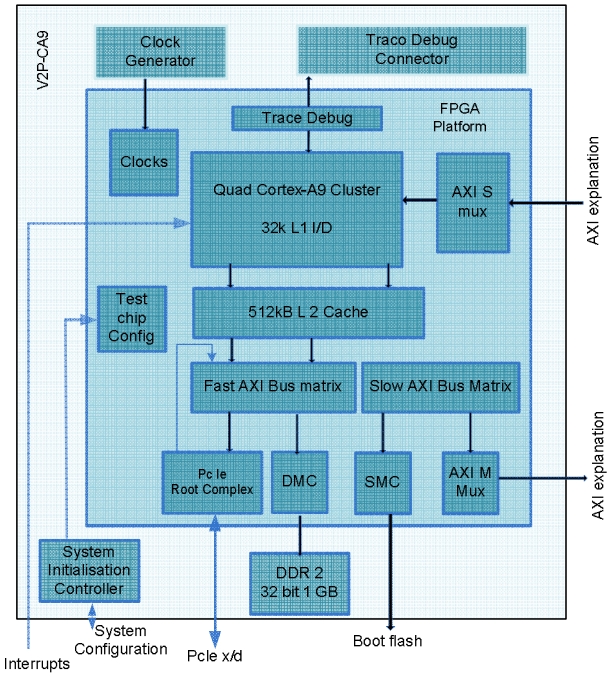

Figure 4-3 - ARM Quad Core Versatile Express (14)

4.3 Cortex-A9 - TI's OMAP4430 SoC

The Pandaboard is an evaluation platform developed by Texas Instruments based on the OMAP4430 which is a dual core Cortex-A9 based SoC. The OMAP4430 is realized on TSMC's 45nm process technology and is capable of running at 1GHz. Additionally, the OMAP4430 has a 32kB I/D L1 cache and a 1MB L2 that is shared between the two cores. As well, each core is equipped with dedicated NEON and FPU co-processors (MPE) (15). The panda board is designed with 1GB of low power DDR2 connected to a 32 bit memory bus running at 400MHz. The OMAP4430 and

panda board support several peripherals, including an Ethernet port, wifi and several other IO standards as seen in Figure 4-4 and Figure 4-5.

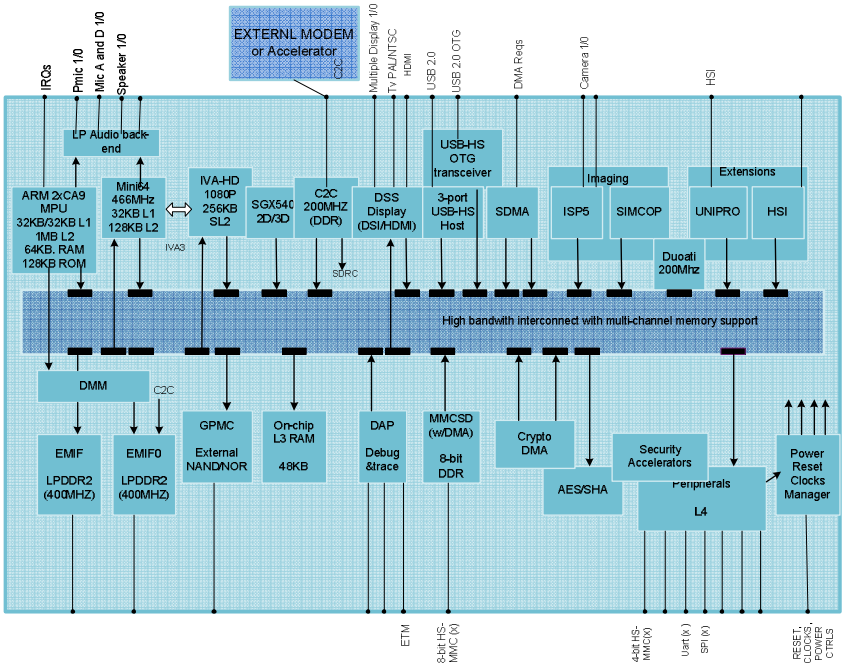

Figure 4-4- TI OMAP 4430 2xCortex--A9 SoC Block Diagram (16)

At the system level, the panda board requires a 5V input that feeds the dialog power management integrated circuit (PMIC). The PMIC then distributes power to the various voltage domains on the OMAP4430 SoC and panda board peripherals. The bootloader, Linux kernel (version 2.6.35) and root file system are stored and run from the SD port. For this thesis most of the peripherals were ignored except the Ethernet and USB ports; the rest were deactivated at the software level.

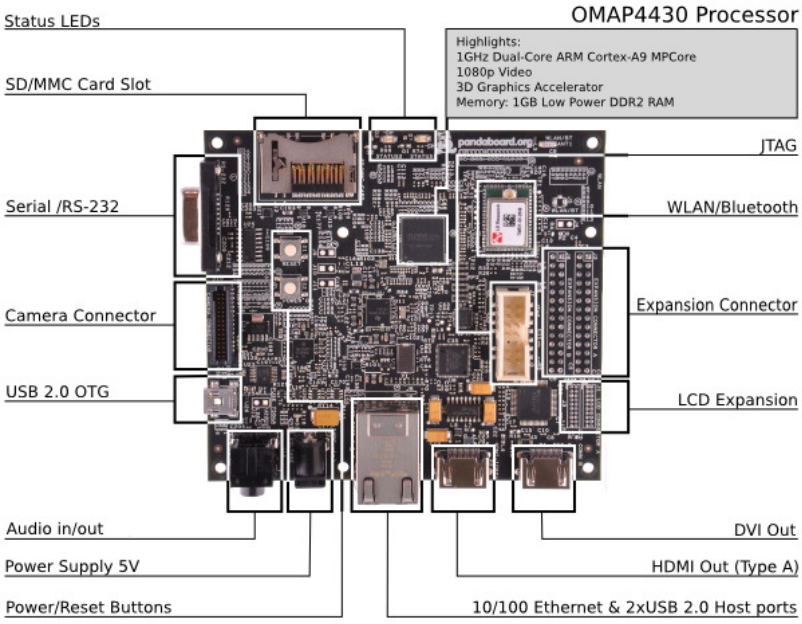

Figure 4-5 - TI OMAP4430 Based Pandaboard (17)

4.4 Cortex-A9 - Freescale's i.MX6x SoC

The Freescale i.MX6Q is a quad core Cortex-A9 based SoC that was realized on a 40nm process technology; it includes ARM MPEs for each core, is capable of running at 1.2GHz, and has integrated on chip power management circuitry. The SoC has 32kB L1 cache for instruction and data and a 1MB shared L2. The main memory bus is 64 bits wide, supporting both DDR2 and DDR3, and capable of running at 533MHz. At the time of this evaluation, the i.MX6Q processor used was a production prototype; thus, the Linux Board Support Package (BSP) was not yet optimized or fully supported. The core clock was set to 1GHz and not the maximum of 1.2GHz. Additionally, the minimum core voltage value was not yet optimized. Thus, the

power and performance results disclosed in future sections reflect an approximate 20% degradation from the capabilities of the SoC. For the purpose of this thesis, the author was given special access to the i.MX6Q evaluation board. The bootloader, Linux kernel (version 2.6.38) and root file system are stored and ran from an SD port.

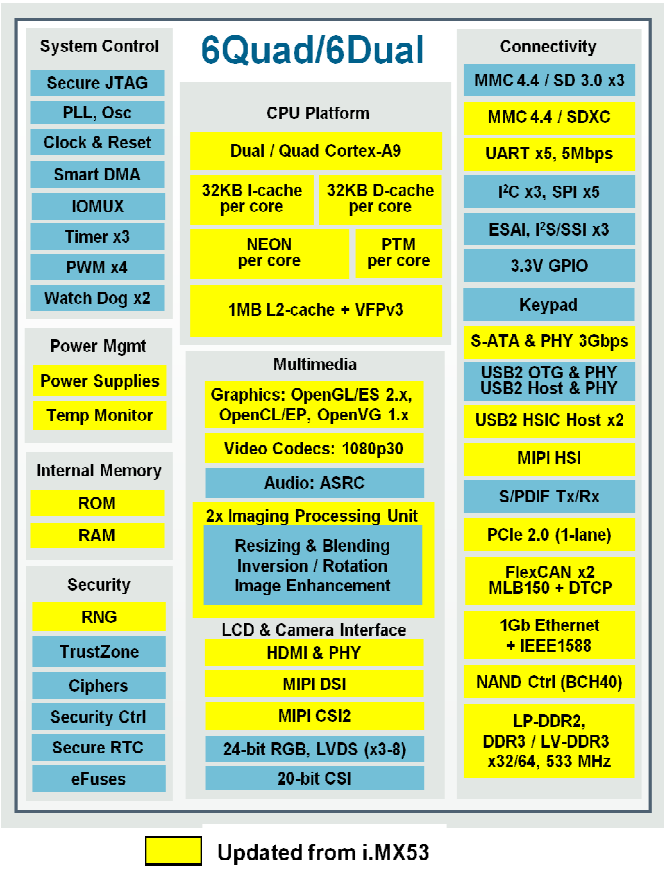

Figure 4-6 - i.MX6Q - Block Diagram (2)

4.5 Intel Atom - The Zotac ZBOX dual core Atom based Nettop

The D525 is an Atom based Pine Trail-D SoC platform developed for the Nettop market (18) and is manufactured on a 45nm CMOS process node. It was chosen for this project as its implementation into the Zotac Zbox HD-ID41-U is similar to the ARM based development kits in terms of multicore SMP, IO resources, and a low power design theme. The Zbox includes the dual core 1.8GHz Atom D525 with integrated memory control and a south-bridge chip, with 2 GB of DDR3 running at 800MHz on a 64bit data bus. The D525 has a 32KB L1 instruction cache, a 24kB L1 data cache, a 1MB shared L2 cache and a 64 entry micro-TLB.

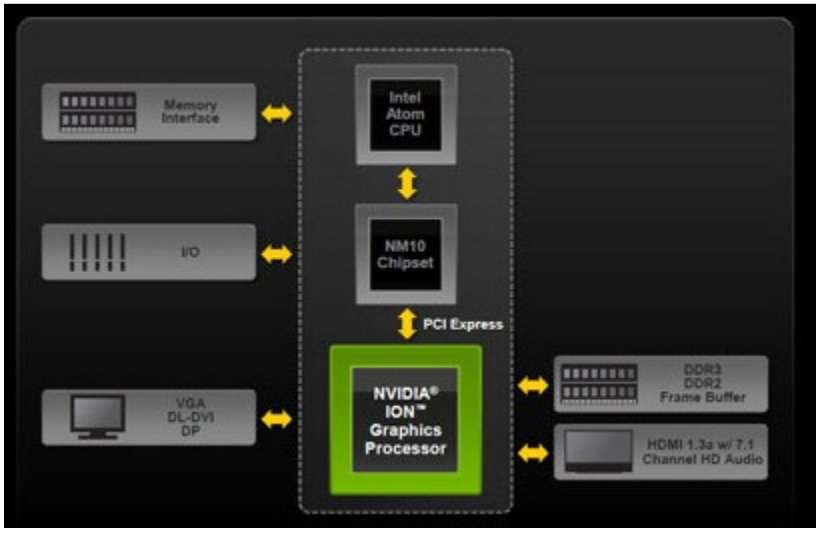

Figure 4-7 - Atom D525 based Zotac Nettop platform (19)

The ZBOX has an Nvidia ION GPU, shown in Figure 4-7, which has been disabled while running BOINC and SETI. For this thesis, the Zbox ran an installation of Ubuntu 10.10 which used the Linux 2.6.35 kernel. The root file system was installed on a 2.5 inch 7200 RPM SATA drive on an EXT4 partition.

4.6 Intel Nehalem - Gateway NV59 Core-i3 based notebook

A Nehalem based Gateway Notebook, the NV59, was used to evaluate the processing performance of Nehalem against the mobile embedded processors of this thesis. The NV59 uses a dual core implementation of the multithreaded Nehalem architecture as seen in Figure 3-9 running at 2.13GHz. It has 4GB of DDR3 with 2x64bit channels running at 1033MHz . The L1, L2, and L3 caches were set to 32kb, 256kb, and 3MB respectively with the L3 cache as unified and share between both cores. It has a two level translation look aside buffer with 64 entries in the first level and 512 in the second. The test system ran the Ubuntu 10.10 version of Linux with kernel version 2.6.35 and GCC version 4.4.3. During power benchmarking, the LCD on the NV59 was disabled and the battery was removed to isolate the power consumption at the system level for a fair comparison.

5 SYNTHETIC BENCHMARKS

In this chapter three synthetic benchmarks are used to evaluate the performance of the project processors in addition to the SETI WU discussed in the next chapter. In this section, performance values for floating point operations, integer operations, and memory bandwidth are investigated as these three metrics are most critical to timely execution of scientific work loads.

The BOINC middleware client includes an integer benchmark (Dhrystone) and a floating point benchmark (Whetstone) to assess the performance of the volunteers' computing system, and are both integrated into the BOINC client. The results are sent back to the server and the server then determines how much work to give the volunteers system based on the benchmark results. The benchmarks are compiled as part of the BOINC client as described in appendix C. At the time of this thesis, the GCC compiler version (4.4.3) used did not support the architectural specifics of Nehalem architecture; however, the -march flag was set to "core2" and provided fairly good results for the Core i3. However, performance improvements are likely as Nehalem expands on SIMD floating point support from its predecessor, the core architecture. The -mtune flag for the Cortex-A8, Cortex-A9, and Atom were set to their exact names, -mtune=cortex-a8, -mtune =cortex-a9, and -mtune=atom. The ARM Cortex A series processors also required an additional flag -mfpu=neon to make the compiler aware that the SIMD neon co-processor and floating point units are available on the silicon.

An important consideration is that BOINC benchmarks Whetstone and Dhrystone may not predict workload performance accurately as these particular benchmarks are small enough in size to fit entirely inside the L1 cache on all test systems of this thesis, and, thus, has little coverage on the memory performance of the test systems. However, for the purpose of characterizing the performance of a system for a loosely coupled BOINC application, Dhrystone and whetstone are sufficient. As with any

single benchmark, none of them reflect the overall performance of the machine. The best benchmark is the application that is intended to run on the machine, as done with SETI in section 6 where the memory performance is also examined.

5.1 BOINC Whetstone

Whetstone measures primarily floating point performance in floating point operations per second (FLOPS). Whetstone is a single threaded application that is programmed and compiled to run on only one core. However, BOINC exploits concurrency rather than parallelism as it will schedule WUs on all available cores or quasi-cores (multithreading). Therefore, on a quad core system or dual core system with 2-way multithreading, BOINC will schedule four WUs for execution by default. However, with a multithreaded processor like the Atom D525 and Core i3, the core's pipelines are shared by two separate execution threads which share the pipeline during stall cycles. Thus, concurrently executing threads introduce additional overhead when run on multithreaded processors versus dedicated single threaded cores. Depending on the application, the overhead with multithreading could be negligible during stall periods due to frequent memory accesses, but with whetstone the entire binary image is small enough to fit into the L1 cache on all systems. This means that whetstone will be tightly coupled to the core architecture even on fine grained multithreaded processors. Therefore, the data in Figure 5-1 shows a maximum of two instances of whetstone running on the Atom D525 and Core i3 because these multiprocessors have only two

dedicated cores. The multicore MFLOP values were calculated from the values that resulted in the single core execution of Whetsone, multiplied by the number of dedicated cores. For clarity, the Cortex-A8 single core whetstone performance is multiplied by one, the OMAP 4430, Atom 525, and core i3 single core performance is multiplied by two, and the Versatile Express and i.MX6Q values are multiplied by four. These calculations ignored overhead because it is possible to execute instances of Whetstone on each core without accessing the shared regions of the memory hierarchy.

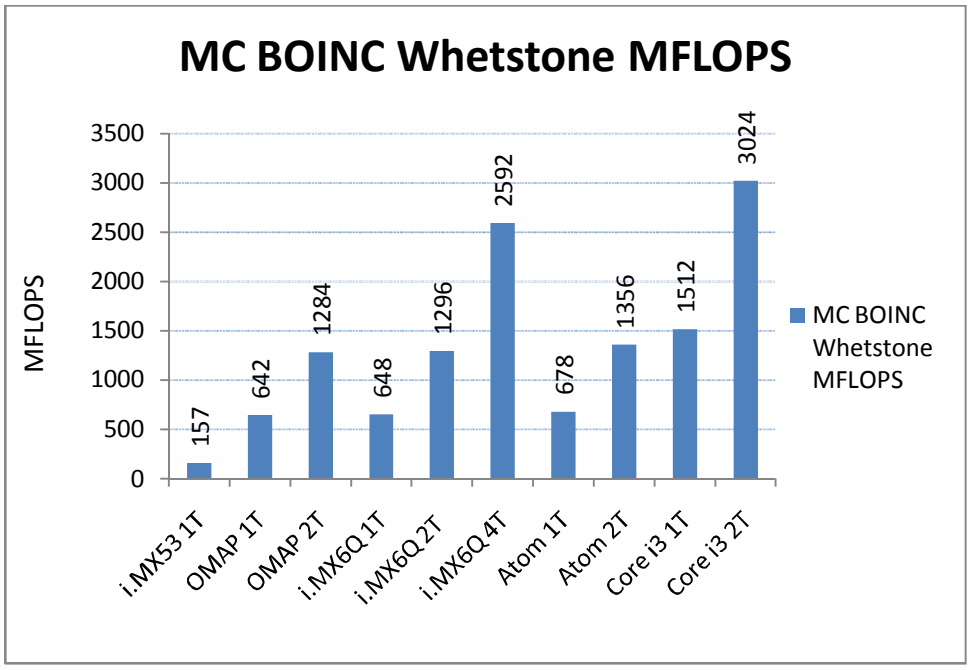

Figure 5-1 - BOINC multicore MFLOPS

To determine the BOINC whetstone MFLOPS/Watt shown in Figure 5-2, the data from Figure 5-1was used in conjunction with the power logged while running multiple instances of SETI in Figure 6-10. The completion of Whetstone and Dhrystone only takes a few seconds. Thus, sampling power every second only provides a few data points. As a result, the SETI WU was used as the power baseline due to the fact that several thousand samples could be taken to during the execution to provide the best average. These power values for SETI are seen in Figure 6-10, and talked about in section 6.3.

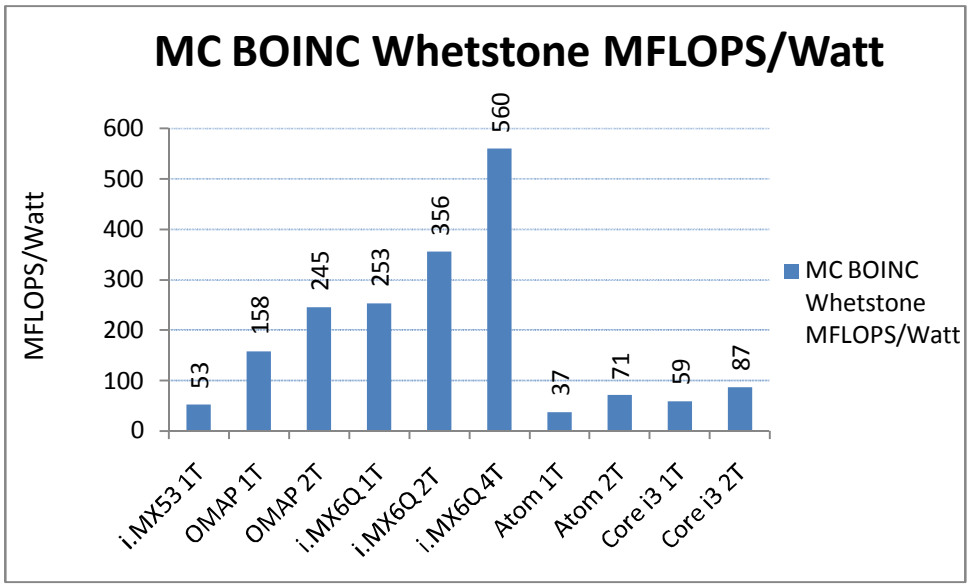

Figure 5-2 - BOINC multi-core MFLOPS/Watt

5.2 BOINC Dhrystone

Dhrystone is included in the BOINC client to measure integer and string performance. The performance value is expressed in number of iterations, or Dhrystones MIPS per second. Like Whetstone, Dhrystone is a single threaded application that is programmed and compiled to run on only one core. Thus, the data for multi-core Dhrystones MIPS and Dhrystone MIPS/Watt in Figure 5-3 and Figure 5-4 respectively are calculated in the same way as multi-core Whetstone MFLOPS and Whetstone MFLOPS/Watt as described in the previous section.

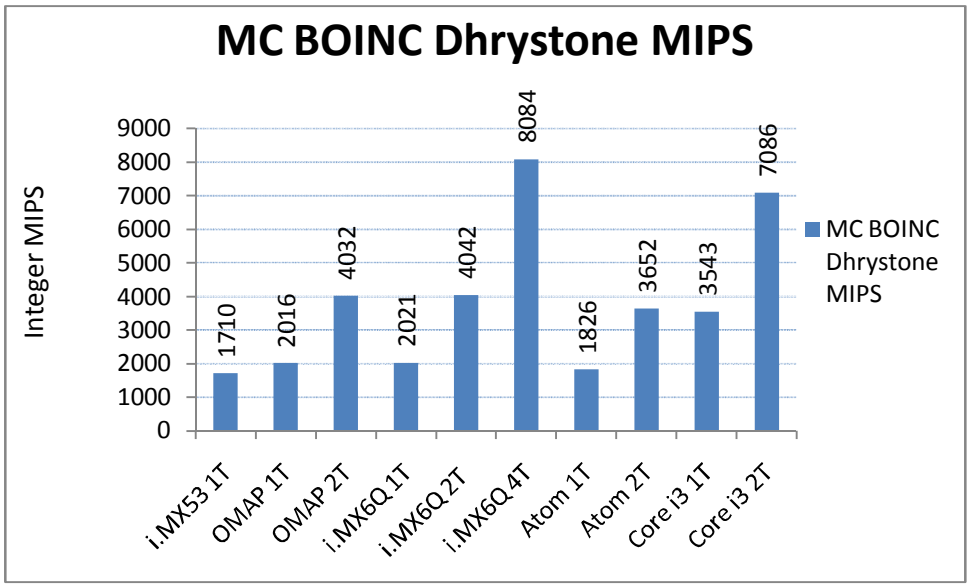

Figure 5-3 - BOINC multi-core Dhrystone MIPS

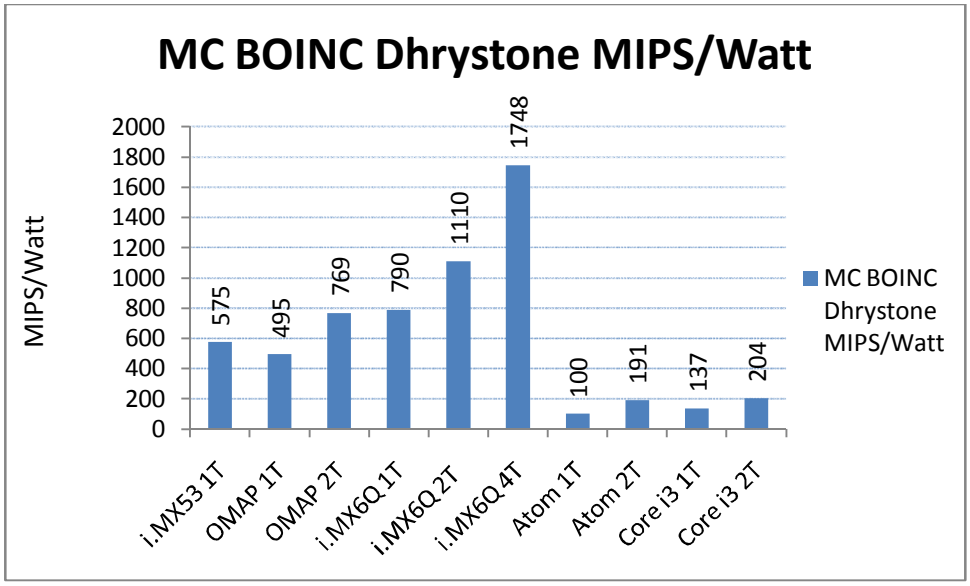

Figure 5-4 - BOINC multi-core Dhrystone MIPS/Watt

5.3 STREAM

The STREAM benchmark is a synthetic benchmark program that measures memory bandwidth and the corresponding computation rate for simple vector kernels such as Add or Triad. As the memory wall continues to grow, CPUs are approaching the point where it is pointless to increase the clock rate as they will frequently be in a state of waiting on memory. As this continues, programs will be limited by the memory bandwidth of the system rather than by the computational performance of the CPUs. As an extreme example, several current high-end machines run simple arithmetic kernels for out-of-cache operands at 4-5% of their rated peak speeds, which means that they are spending 95-96% of their time idle, waiting for cache

misses to be satisfied (20). STREAM benchmark is specifically coded to work with datasets much larger than the available cache on any given system to ensure that the entire memory hierarchy up to the DRAM is being accessed. STREAM uses the OpenMP library to utilize multiple cores on an SMP system and reflect the cache coherence overhead of the system. Memory throughput is extremely important with several scientific applications as many of them work on large data sets. In addition, the memory wall is aggravated by multi-core systems, so understanding limitations to the system memory hierarchy can provide insight into the potential performance limiters of embedded processors. The unmodified STREAM benchmarks implement read/write tests, using four different memory access patterns. These tests perform calculations on a one-dimensional array of double precision floating point numbers. The authors made the following modifications to the original STREAM benchmarks to better compliment the assembler benchmarks. Every thread is bound to a specific core and allocates its own memory. The time stamp counter is used for time measurement which excludes overhead caused by the spawning of new threads. The author's preprocessor commands to the code to optimize memory accesses and to perform explicit writes of data to main memory without affecting cache (12).

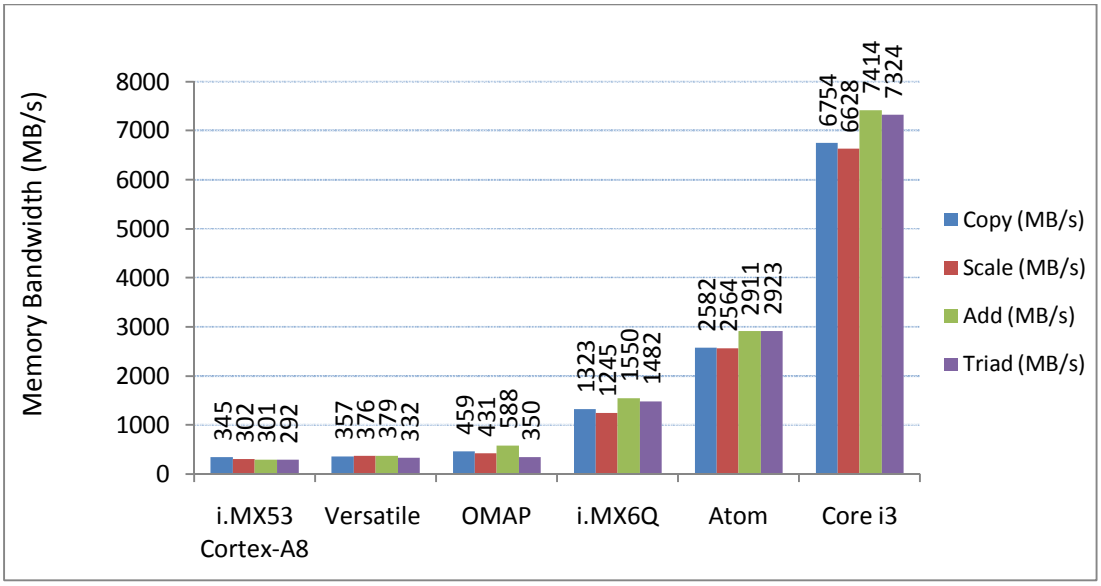

Figure 5-5 - STREAM Memory Benchmark Results

As expected, the Nehalem based Core i3 system delivered bandwidth that was several times that of the other systems. It has two 64 bit DDR3 memory channels running at 1066MHz; whereas, on the low end i.MX53 has one 32 bit DDR2 channel running at only 400MHz.

6 RUNNING A BOINC APPLICATION - SETI@HOME

The SETI test workload is a static work load that is included with the source code to test the execution time on various machines. On some systems it takes only a few hours to run; whereas, on others it takes a few dozen hours. Unlike synthetic benchmarks, this test is an actual WU from the SETI@Home volunteer computing project. As a result, this benchmark test will provide a direct reflection of the

performance of these test systems if executing a WU that was scheduled by the SETI server. The WU is included in the subversion download from the SETI website and will be uniform across all platforms. In addition, it enables us to establish a power profile for the test systems because the execution time is long enough to get thousands of data points for current draw during WU execution.

The SETI application was ported to the various platforms and architectures and the procedure is described in APPENDIX A. The values in Figure 6-1 represent the actual execution times and are not normalized to a reference clock frequency.

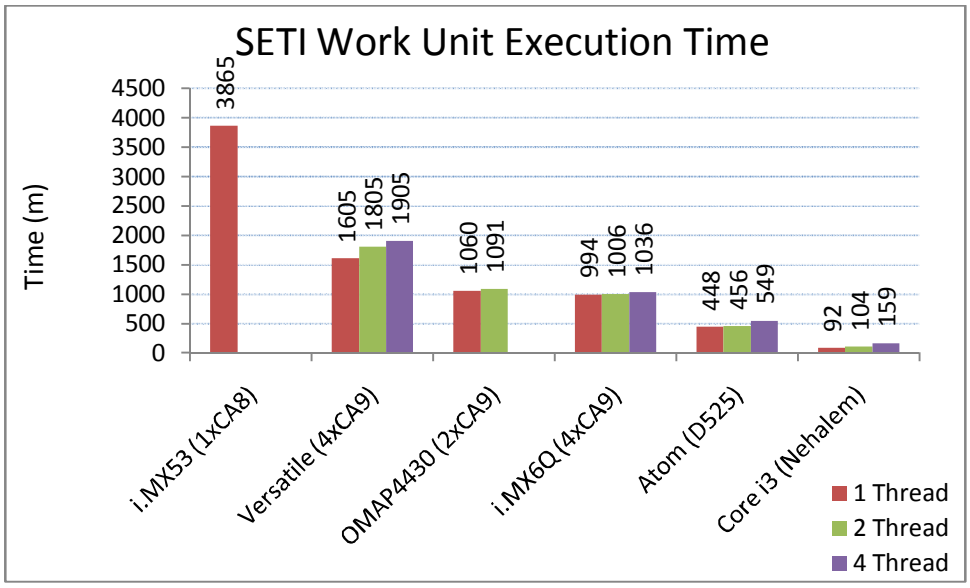

Figure 6-1 - SETI execution for multiple threads

BOINC provides the middleware framework for parallel volunteer computing; however, on the client system BOINC schedules multiple WU instances concurrently,

depending on the number of processors of the SMP system. This is different from parallelism as each SETI WU instance has no knowledge of each other. However, each SETI WU instance running on an SMP client system must share the memory hierarchy with the other instances. This introduces concurrency overhead during periods of memory access to the levels of the memory hierarchy that are shared by each processor as seen inversely from the speed up in Figure 6-2 These values were determined by measuring the execution time of running one, two and four instances of SETI concurrently on the SMP system and measuring the execution time difference between one instance versus two and four instances. By using Amdahl's Law, the overall speed up from running concurrent instances of the SETI WU can be calculated by dividing the execution time of single threaded runs by the adjusted execution time of the multithreaded runs (i.e. multithreaded time / threads).

The Intel processors showed a reduction in speed up from running 2 threads to 4 threads versus 1 to 2 threads as shown in Figure 6-2, but this is because these are dual core processors that support multithreading; thus, this overhead is due to sharing the processor pipelines in addition to memory resources. This overhead is not as visible on the Atom processor versus the Core i3 which is likely due to the fact that the program execution is in-order per the pipeline organization, and the memory hierarchy is lower performance which introduces pipeline stall gaps that can be exploited with Atom’s multithreading capabilities. The added overhead from 2

threads to 4 threads on the core-i3 shows that the processor has tighter program coupling in the processor pipeline which is expected as exploits out of order execution and the memory hierarchy is higher performance. In either case, multithreading shows a substantial power and performance benefit as concurrency is able to utilize the processor's pipes during stall states of the other thread. Overall, the concurrency overhead on ARM core was very low on all processors as seen by the nearly linear speedup as high as 3.84 (linear is 4) on i.MX6Q which make running concurrent WU on SMP systems ideal for BOINC applications in terms of throughput and energy efficiency.

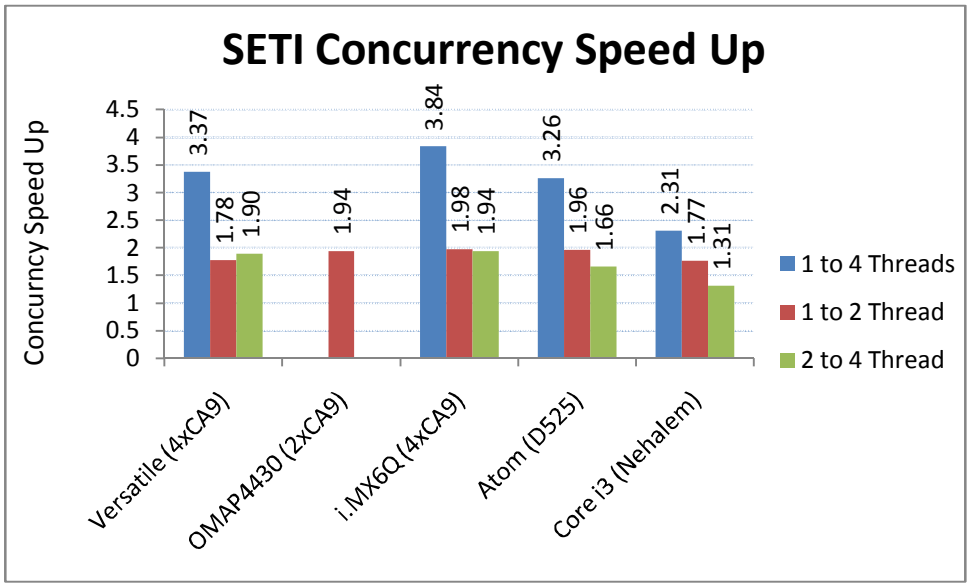

Figure 6-2 - SETI Concurrency Speed Up

6.1 SETI Performance profiling

Many CPUs including ARM and Intel provide performance counters, which are hardware registers that can count events within the CPU. Several performance events exist on all architectures, most of which have little correlation to each other. For this thesis the performance events were chosen if they provided a direct comparison across the different architectures. As such and regarding this thesis, they are restricted to, 1) The CPU clock cycles which are the number of clock cycles during the sample period. 2) Instructions retired or executed, which represent the number of instructions that are committed. 3) last level cache misses, which are the misses to the last level of on chip cache memory that caused an access to main memory. 4) last level cache hits which are the accesses to the last level of cache memory that prevented an access to main memory. 5) Instruction TLB misses, which are instructions that missed the TLB and caused the OS to look up the physical address from the virtual address. 6) Data TLB misses, which is a data location that missed the TLB and caused the OS to look up the physical address from the virtual address. With the statistical information provided by these performance events, the cache miss rate, branch prediction accuracy, and clock cycles per instruction can be calculated in addition to the actual performance events.

To access these performance counters and the popular Linux based profiling package, oprofile was used to profile the performance events with regard to the SETI application. Oprofile has been ported to a number of different architectures and provides profiles of code based on the number of these occurring events. Every time a certain (configurable) number of events has occurred, the PC value is recorded. This information is aggregated into profiles for each binary image. It is capable of profiling all parts of a running system, from the kernel (including modules and interrupt handlers) to shared libraries to binaries. It runs transparently in the background, collecting information at a low overhead which has been measured to be around 3% or less in this thesis project. These features make it ideal for profiling entire systems to determine the bottlenecks of a system with regard to the applications proposed in the thesis. The profiling was done while running the SETI WU while oprofile collects the performance data. Because the evaluation processor architectures completed the static WU at different times, a sample period for each architecture needed to be determined. The idea is have the sample period "sample" across the same section of the SETI WU. On the Core i3, 10% of the WU is done in 9.1 minutes, and on the Cortex-A8 10% of the same WU is done in 6.4 hours. Thus, the sample period needs to be normalized across the different architecture completion times, so that the end of the sample period on one architecture is approximately the same end point on the other architectures with respect to the WU completion percentage. This was done by choosing the sample period for the Core i3 and then scaling the sample

period across the lower performing architectures in order to provide the best possible comparison. The profiling was done running only one execution thread of SETI on all architectures. With the initialization line below the value 200000 is the sample rate at which oprofile will increment the counter. So after 200000 accesses to the L2 cache, the sample value will increment once. The "0x0" is the flag value that inputs specific detail into what to be counted, but in this case it is the default. The next two values "0:1" specify to profile the kernel or user space. The kernel value is set to '0' because the kernel is of little interest to this project; thus, the focus remains on the SETI WU with the user space flag set to "1".

```
sudo opcontrol --event=L2_ACCESS:200000:0x0:0:1
```

Due to the complexity of adding oprofile support on different architectures, only 4 of the 6 systems were evaluated with performance counters. The evaluation includes at least one representative from each processor family; thus, only one of the three Cortex-A9 processors (the FPGA based Versatile Express) will be evaluated with Oprofile running SETI.

6.2 Performance events

Figure 6-3 shows the raw data dump of the data collected from the registers of the performance counters. Several caveats exist when using this type of data to evaluate the wide variance of architectures in addition to the sample scaling that was

previously discussed. First, the clock frequency ranges from 400Mhz to 2.13GHz on the evaluation systems; thus, the sample period was normalized from execution wall time and scaled from the core-i3. Second, Intel processors use a Complex Instruction Set Architecture (CISC) which has the characteristic of code density. Thus, the number of executed instructions should be slightly less than a Reduced Instruction Set Architecture (RISC) such as the ARM processors. Third, the instructions per clock cycle (IPC) of the evaluation processors ranges from 0.16 to 1.28 which can cause additional differences with the frequency of events across the different architectures due to more or less instruction level parallelism, causing more events to occur per cycle. That said, this evaluation methodology is not perfect and using the comparisons between the number of instructions executed, it shows a margin of error as high as 18% within the two classes of instruction sets. However, the margin of error is negligible when using data from a specific platform to make calculations regarding that same platform such as branch prediction accuracy or instructions per clock cycle. For example, data regarding branch instructions executed and branch instructions missed were both collected from the same execution sample period from the same processor and doesn't require any normalization or scaling to produce the value. However, the 18% error margin regarding the number of instructions executed is undoubtedly a worst case value as the number of clock cycles executed is a very frequent event that is close to the number of clock cycles. In the case of the memory

events, the error margin will be much lower as these events are very infrequent with respect to the number of instructions executed as seen in Figure 6-3.

Event	Cortex-A8	Cortex-A9	Atom	Core-i3
CPU clock cycles	11116868	2603703	2533910	612285
Number of instructions executed	1774661	1874861	924637	783789
Requests to memory	14413	25046	4802	503
Hits in on chip cache	52955	24	21250	1920
Branch instuctins executed	105364	124715	82646	76263
Branch instuctins mispredicted	4249	4741	1685	678
Instruction TLB miss	84	8	60	1
Data TLB miss	8554	2803	2549	2733

Figure 6-3 - SETI events counted by performance counters

The branch prediction accuracy was determine by counting the number of branch instructions that were executed versus the number of branch predictions that missed from the same sample run using different event counters. All architectures performed well with this metric in the high ninety percentile as seen in Figure 6-4. The lowest is the Cortex-A8 at 96%, which has 13 cycle penalty for any branch prediction miss. The Core i3, Atom and Cortex-A9 carry with it respectively, a 16, 19, and 8 cycle penalty for the pipeline flush due to a branch prediction miss.

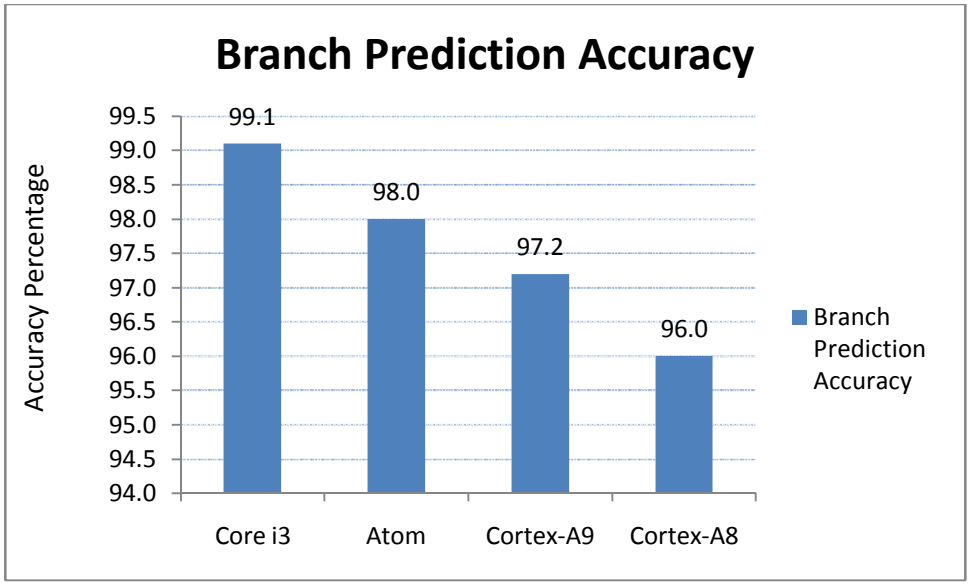

Figure 6-4 - SETI Branch Prediction Accuracy

Memory access for SETI were evaluated by counting the numbers of last level cache misses. A miss in the last level of cache memory results in a high penalty access to memory. On the Atom, Cortex-A9, and Cortex-A8 the last level cache is the L2 which is 1MB, 512kB, and 256kB respectively. On the Core i3 Nehalem architecture, the last level of cache is the unified/shared 3MB L3. Thus, the results are very close to what is expected from the larger caches having the least number of misses. During approximately the same sample period of the SETI WU, the Core i3 L3 cache miss event sampled only ~500 times; whereas, the Cortex-A9 sampled ~16k times.

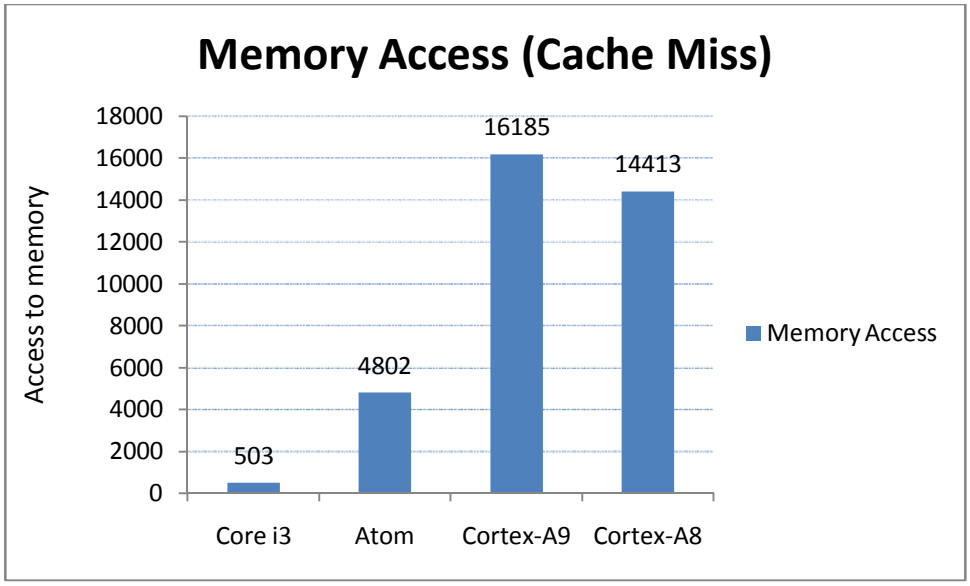

Figure 6-5 -SETI Last Level Cache Misses (memory access)

To investigate the effectiveness of the on chip cache memory, the rate of misses to the last level of on chip memory was calculated. The ARM Cortex-A9 based versatile express has a staggering miss rate of nearly 62k percent; whereas, the Core i3, Atom, and Cortex-A8 measured a miss rate of roughly 22, 23, and 27 percent respectively as seen in Figure 6-6.

Because of this high miss rate, other programs running on the FPGA based Cortex-A9 system were examined. It was discovered that 82% of the last level cache access on the ARM Cortex-A9 were caused by accesses from the Linux kernel. This high level of kernel overhead could be caused by the low L2 density due to the FPGA realization and the kernel is a prototype release and might not be optimized.

However, even though the miss rate is staggering compared to the other architectures,

the total number of misses during the sample period only represent 0.86% of the number of instructions executed.

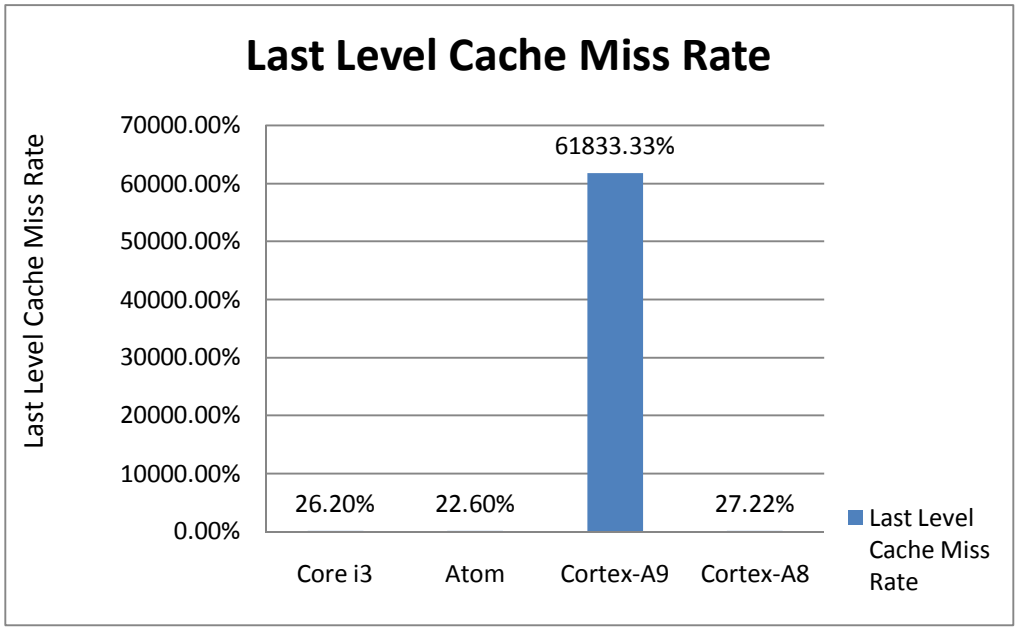

Figure 6-6 - SETI - Last Level Cache Miss Rate

The instruction TLB performed well on the Core i3 as only one sample was taken during the sample period. The samples on the embedded processors varied from 60 on Atom, 79 on the Cortex-A9, and 84 on the Cortex-A8 as seen in Figure 6-7.

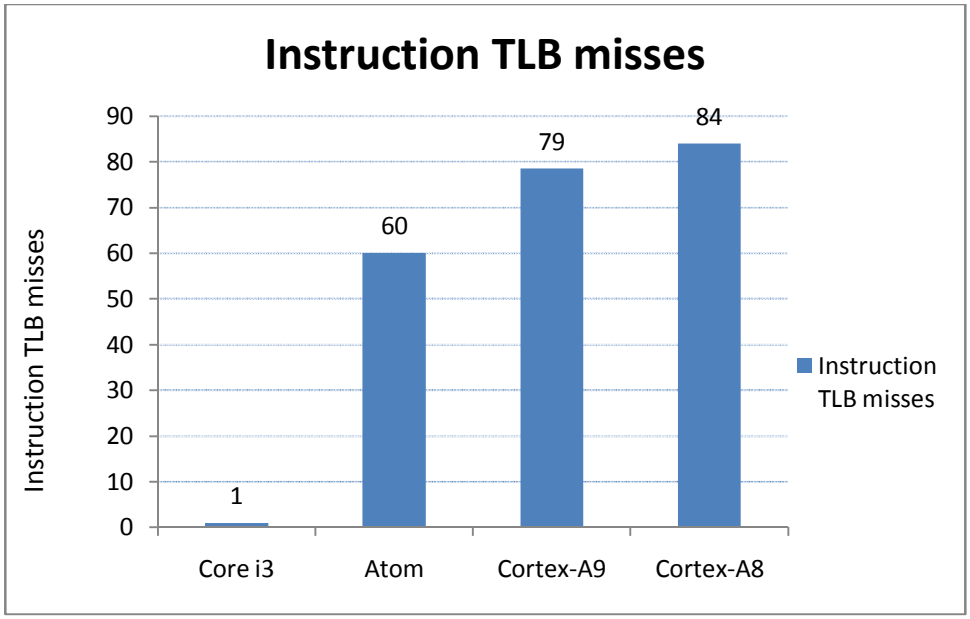

Figure 6-7 - SETI ITLB misses

Data TLB misses occurred much more frequently than instruction TLB misses while running SETI. The Cortex-A8 and Cortex-A9 have 32 entry TLBs, which are reflected by the increased number of TLB misses by comparison with the Intel processors that have 64 entry L1 TLBs. The penalty of a TLB miss could be as high as 100s of clock cycles on embedded processors. The Core i3 had more data TLB misses than the Atom architecture which is likely due to the fact that the Nehalem based Core i3 has a higher much higher IPC than the Atom processor.

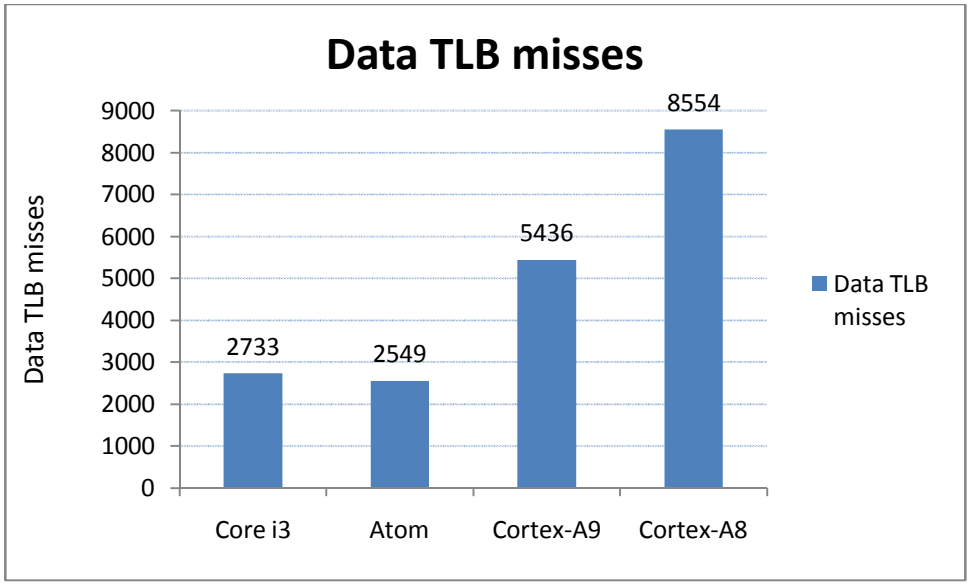

Figure 6-8 - SETI Data TLB misses

Lastly, the number of instructions per clock (IPC) was calculated from the number of instructions that executed or retired versus the number of clocks. With superscalar processors that have multiple pipelines with multiple issue and decode, the number of instructions per clock cycle can be greater than the number of clocks as seen on the Core i3 shown in Figure 6-9. The Cortex-A9 has a very respectable IPC of 0.72 even with the slimmed down FPGA limited L2 cache that had the highest miss rate.

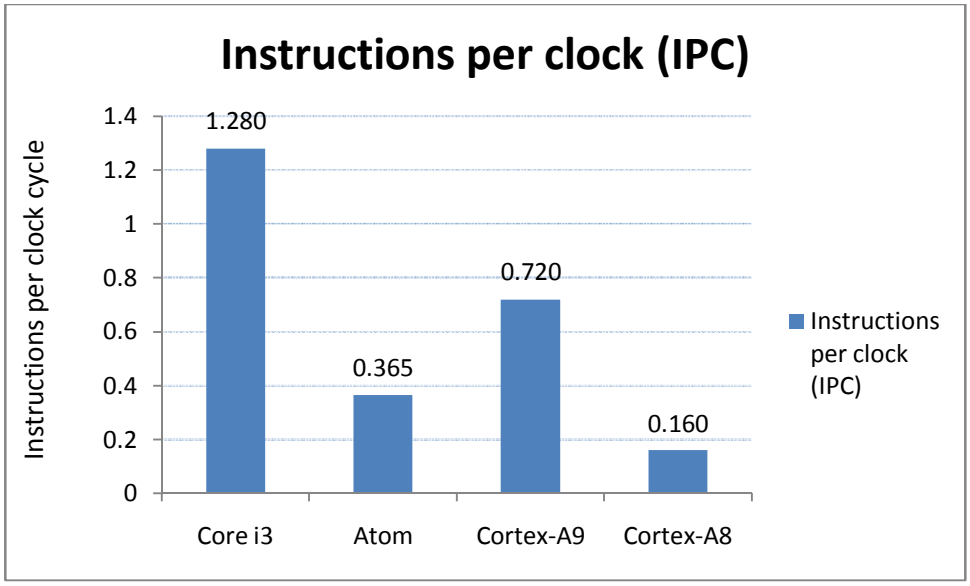

Figure 6-9 -SETI Instructions per clock (IPC)

6.3 SETI Energy Consumption

One of the main value propositions for running a scientific work load on an embedded processor is the lower power consumption. Power at the board level power was logged because isolating the power domains of the internal cores and memory infrastructures would be a major undertaking and would not be representative of the typical scenario running volunteer computing on any platform. Off chip peripherals such as USB, Audio, and HDMI ports are not used while running volunteer computing projects; however, they are powered on at the chip level and, thus, drawing at least some energy. However, all evaluation systems have a very similar mix of board level peripherals and, thus, the differences would be negligible. System power was logged while running SETI during the entire execution of the SETI test

WU. Current was sampled during every second and then averaged across the entire execution period. Increases in dynamic power consumption from running multiple SETI instances or threads can be seen in Figure 6-10. The Gateway NV59 (Core i3) pulled 39.2W while running 4 instances or threads of the SETI WU; whereas, the ARM processors like the quad core Cortex-A9 based i.MX6Q pulled power values as low as 4.62W.

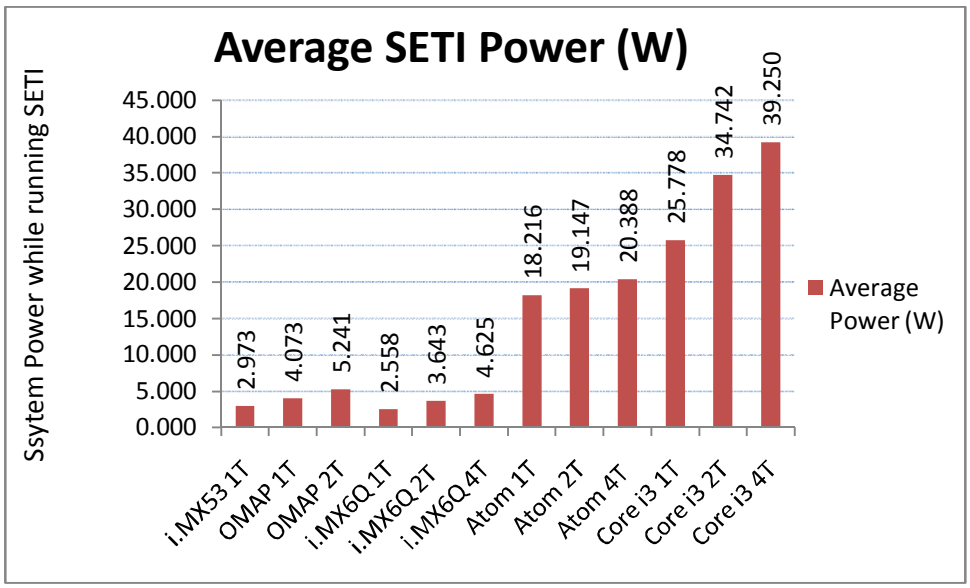

Figure 6-10 - System Power while running SETI instances or threads

However, the slower execution time of workloads on embedded processors could potentially nullify the power efficiency since the energy consumed is a function of power and time. For example, i.MX6Q pulled only 4.6W while executing 4 instances of the SETI WU; however, it took more than 17 hours to complete all 4 WUs. The Core i3 processor pulled 39.4W while running the same 4 instances of the SETI WU

which took less than 3 hours to complete. With the execution time and the average power values logged by the power supply script, the energy was calculated in terms of Joules per SETI WU completed as seen in Figure 6-11. The values in Figure 6-11 are a reflection of the number of threads that can be serviced by processor or multicore processor system. A maximum of one instance of SETI was used to calculate energy for i.MX53, two instances for the OMAP4430, four instances for i.MX6Q, Atom D525, and Core i3. In contrast to the ARM cores, the Intel processors, the Core i3 and Atom D525, both support simultaneous multithreading, so two additional thread executions were added to make a total of four during the maximum execution run. By maxing out the execution threads in the core pipelines, the Intel processors were able to capitalize on the memory latency, hiding effects of multithreading which directly translates to energy savings in deep submicron processing nodes where leakage power dissipation is greater than dynamic power.

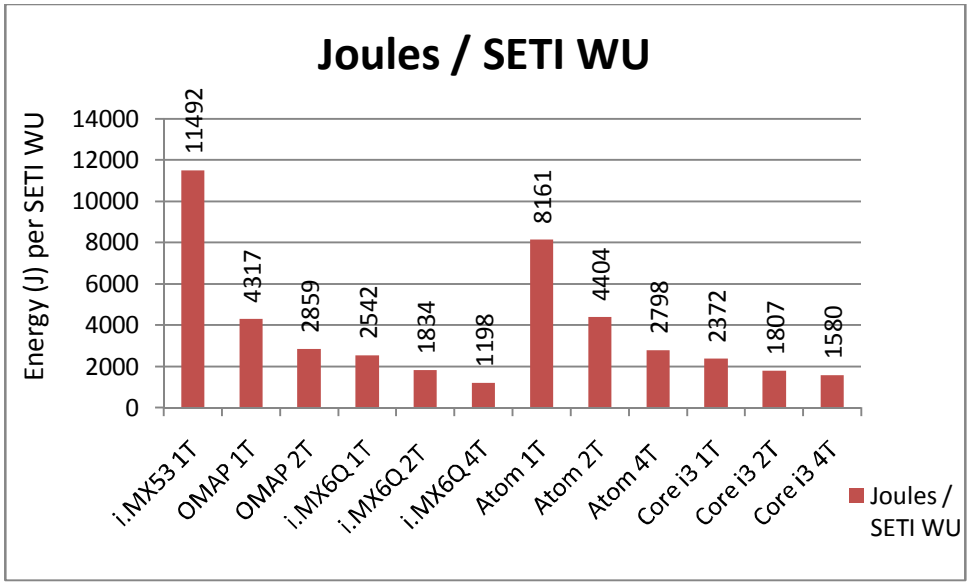

Figure 6-11 - Energy used to complete the SETI WU

7 PROJECT SUMMARY AND CONCLUSION

7.1 Project Summary

The Cortex-A8 was designed primarily for low power integer performance with the twin superscalar integer pipelines, and showed impressive results on the Dhrystone benchmark. The Cortex-A8, however, suffered from low throughput while executing floating point operations during Whetstone and the SETI WU. The vector floating point unit on the Cortex-A8 processor does not support pipelined IEEE floating point operations. As such, no floating point instruction level parallelism was exploited during the execution of the SETI WU and, thus, took over 64 hours to complete.

Although the power was significantly lower on the Cortex-A8 than the Intel x86 processors, the energy consumed was much higher due to the extended duration of the SETI WU execution and resulted in the highest energy consumption as seen in Figure 6-11. That said, even with the lower MFLOPS of the Cortex-A8, the contributions are still viable for most applications such as SETI with little or no system level data dependencies regarding the other processing nodes. However, BOINC supports applications such as Enigma@home which does primarily integer based computation and would be ideal for the Cortex-A8. In either case, with the aggressive time tables of mobile consumer products, Cortex-A8 based SoCs are currently being phased out to make way for Cortex-A9 and next generation Cortex-A based devices with multi-core implementations.

The ARM Cortex-A9 processor is the first ARM core capable of computing sustained high throughput IEEE 754 floating point code and, thus, it enabled the high SETI IPC. This high IPC, however, could be higher with Cortex-A9 realizations that have larger on chip cache memory and a larger TLB network. The NEON co-processor coupled with the pipelined floating point unit are able to produce floating point performance that is around six times greater than the Cortex-A8 as shown by the BOINC Whetstone benchmark seen in Figure 5-1. In addition, the whetstone results show that a clock cycle normalized value for the Cortex-A9 would be nearly the same as the Core i3 Nehalem based processor as the silicon implemented Cortex-A9s were both running at 1GHz but produced nearly half the results of the Core i3 running at

2.13GHz. This suggests that the IPC on the Cortex-A9 is higher running whetstone than SETI. More importantly, with i.MX6Q the performance/watt ratio is more than 6 times greater than on the Core i3 while running Whetstone. This also translated into the lowest energy consumption by iMX6Q with a value of 1198J per SETI WU while running four instances concurrently as seen in Figure 6-11.

However, with running SETI a few performance limitations were revealed on the Cortex-A9 by the duration of the SETI WU execution and logging performance events with the performance counters. The last level cache miss rate was nearly 240k times higher on the 256kB L2 cache of the Cortex-A9 versus the 3MB L3 cache on the Core-i3 while running SETI. A cache miss at the last level of cache memory results in a memory access that causes the processor pipelines to enter a stall state. Depending on the processor, the duration of this stall state can be in excess of hundreds of clock cycles and, thus, performance suffers significantly on all processors and even worse on embedded processors as described in chapter 2.

The Atom processor showed fairly good performance running Whetstone with 678 MFLOPS , but was showed nearly the same performance value of 648 MFLOPS as the OMAP and i.MX6Q Cortex-A9 based SoCs running at 1GHz which is only 55% of the clock frequency of the Atom. However, during the SETI WU execution the Atom completed the job in less than half the time. The architecture also did an excellent job of exploiting concurrency of the SETI WU with its multithreaded

capabilities. As discussed in the previous chapter, Atom's pipeline was well utilized by multithreading as the overhead of running two SETI instances per core was low enough to enable a speed up of 1.66 as seen in Figure 6-2 with a system power increase of only 6%. Intel has introduced many power saving techniques to the Atom processor; however, most of them are tailored to exploit use cases with power states. Intermittent usage by the user has created a need for multiple power states to conserve battery energy. For example, a netbook user may want to keep the netbook close by incase an email needs to be sent or information needs to be sporadically searched on the internet. The rest of the time the system is able to exploit the Atom's low power states described in Figure 3-7. However, with BOINC running a scientific application such as SETI, the system will need to be in the max power state C0 of Figure 3-7 during the entire BOINC/SETI session. Therefore, these energy saving states in addition to dynamic voltage frequency scaling (DVFS) will not be exploited while running BOINC applications in the background. This, among other reasons, is why Atom had the worst performance per watt ratio of all the processors while running Dhrystone and Whetstone. However, with multiple instances of the SETI WU running concurrently, the in-order multithreaded execution gave Atom a speed up of 3.26 with a power increase of only 11% as seen in Figure 6-2 and Figure 6-10 respectively. Thus, as an embedded processor Atom does very well running a volunteer scientific workload in terms of energy efficiency if concurrency is exploited.

The Nehalem architecture is a representative of a typical BOINC client machine as well as an architecture used in many supercomputers that do scientific computing. Thus, no conclusion is needed, and as previously stated Nehalem was added to provide perspective to this thesis and to provide a normalization reference point.

7.2 Project Conclusion

The purpose of the project was to determine if using mobile embedded processor for volunteer computing was feasible in terms of processing power and energy consumption. The term feasible is subjective and may vary depending on the granularity of the application shown in Figure 2-2. Feasibility, determined by the author of this thesis has two metrics 1) if the performance/watt ratio is similar to that of a standard BOINC client compute node such as a Nehalem based Desktop, and 2) if the WU execution duration of the embedded processors completes its executions in a timely manner such that it does not introduce bottlenecks in the form of starvation on other processing nodes. Thus, with this loose definition of feasibility, and with examining the data for WU execution time, floating point and integer performance, energy consumption, and massive number of embedded mobile devices, the evidence of feasibility is overwhelming. By extending volunteer computing to embedded processors, scientists developing volunteer applications could tap into a maximum

theoretical value of 141 ExaFLOPS of processing power while using much less energy. However, this value does assume a few ideal conditions that are unlikely in practice. The first assumption is ideal concurrency and parallelism with regard to the concurrent execution of WU on a SMP BOINC client node, and parallelism at the distributed application level. The second assumption is that all 6 billion mobile devices employ a processor such as i.MX6Q running a BOINC application that exploits both the Cortex-A9 FPU and the Vivante GPU. The third is that they would all have to be running this application at the same time. In practice, a more realistic value would be 5% of all mobile devices which divides 141 ExaFLOPS down to 7 ExaFLOPS. In addition, the number of active users is a fraction of the number of total users. For example, SETI has over 5.2M total users with average of 286K active users. As such, the average SETI computational contributions from users is only 461TFLOPS when a peak average could be approximately 8.38 PFLOPS if all users were active at once. That said, if the second assumption were true, the peak adjusted value of 7 EXAFLOPS would be more like 1/18th of that value which is around 388 Peta FLOPS of average computational contributions by using the same ratio. This is still a massive amount of processing power that is around 48 times the processing power of the world's fastest supercomputer, (K computer) which is a Japanese supercomputer capable of performing more than 8 quadrillion calculations per second or 8 PetaFLOPS.

This aggressive time table for consumer products introduces a major benefit to the proposed use of said devices for volunteer computing. On average consumers replace their cell phones every 6-18 months, which has the effect of a built in refresh mechanism for compute nodes (cell phones) as BOINC volunteers update their devices. This, unlike other types of parallel computing introduces a characteristic of self maintenance and upgrading and will ensure that computational contributions from mobile devices will scale with Moore's Law.

7.3 BOINC modifications needed

Even with the energy efficient computation of ARM processors, 4.6W of continuous power draw will drain the mobile device battery quickly. Thus running BOINC and volunteer computing applications like SETI will be limited to the time period when the mobile device is connected to a battery charger or external power supply. Additionally, the current standard for high end smart phone chargers are 5V at 1A which equates to a maximum wattage rating of 5W. At the circuit level, the typical power management IC is designed to route current from the battery charger to the embedded system as needed through a network of diodes. These diodes will route current overflow (if any) to the device battery. An average power draw of 4.6W on i.MX6Q, running four SETI instances doesn't leave much power to charge the battery, and, as a result, charging the battery will take longer. However, the industry is

moving toward 10W and 15W chargers for mobile devices at the high end to meet these increased demands and will, thus, nullify this problem on most future devices. Either way the mobile BOINC client will need to incorporate mechanisms to detect the presence of a battery charger as a condition for scheduling a WU. The BOINC client may also have to include preferences for running at night during the time the user sleeps since battery charge time will take longer. Since some mobile devices take a significant amount of time executing WU, another modification that might be needed is dispatching smaller WU from the BOINC application servers so that WU completion doesn't take as long.

7.4 Future work

1. Porting BOINC to mobile operating systems

Additional work on this effort is already under way by the BOINC development team at UC Berkeley. In addition to other features, the BOINC client will be ported to an ARM based Android platform using the Android Native Development Kit (NDK). The NDK improves on the Java only SDK features to allow native C/C++ compilation which enables code to exploit the co-processors such as NEON, FPU and GPUs. A variation of the needed modifications discussed in the previous section will be included with this port.

2. Exploiting graphics co-processors on embedded SoCs

This thesis only explores the computational performance of the processor cores; however, exploiting mobile GPUs could provide performance several times that of using only the FPUs on mobile processors as discussed in chapter 1 and chapter 7.2.

APPENDIX A - PORTING BOINC TO TEST ARCHITECTURES

Instructions to build and optimize BOINC and SETI for the test system architectures.

Cortex-A8

```
#!/bin/bash

sudo apt-get install autoconf m4 openssl libcurl4-openssl-dev libtool subversion
libfftw3-dev

svn co http://boinc.berkeley.edu/svn/trunk/boinc
cd boinc/
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=cortex-a8 -mfpu=neon" CFLAGS="-O3 -
mtune=cortex-a9 -mfpu=neon"
make

svn checkout https://setisvn.ssl.berkeley.edu/svn/seti_boinc
cd ../seti_boinc
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=cortex-a8 -mfpu=neon" CFLAGS="-O3 -
mtune=cortex-a9 -mfpu=neon"
make
```

Cortex-A9

```
#!/bin/bash

sudo apt-get install autoconf m4 openssl libcurl4-openssl-dev libtool subversion
libfftw3-dev

svn co http://boinc.berkeley.edu/svn/trunk/boinc
cd boinc/
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=cortex-a9 -mfpu=neon" CFLAGS="-O3 -
mtune=cortex-a9 -mfpu=neon"
make

svn checkout https://setisvn.ssl.berkeley.edu/svn/seti_boinc
cd ../seti_boinc
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=cortex-a9 -mfpu=neon" CFLAGS="-O3 -
mtune=cortex-a9 -mfpu=neon"
make
```

Atom

```
#!/bin/bash

sudo apt-get install autoconf m4 openssl libcurl4-openssl-dev libtool subversion
libfftw3-dev
```

```
svn co http://boinc.berkeley.edu/svn/trunk/boinc
cd boinc/
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=atom" CFLAGS="-O3 -mtune=atom"
make

svn checkout https://setisvn.ssl.berkeley.edu/svn/seti_boinc
cd ../seti_boinc
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=atom" CFLAGS="-O3 -mtune=atom"
make
```

Intel Core-i3

```
#!/bin/bash

sudo apt-get install autoconf m4 openssl libcurl4-openssl-dev libtool subversion
libfftw3-dev

svn co http://boinc.berkeley.edu/svn/trunk/boinc
cd boinc/
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=core2" CFLAGS="-O3 -mtune=core2"
make

svn checkout https://setisvn.ssl.berkeley.edu/svn/seti_boinc
cd ../seti_boinc
./_autosetup
./configure --disable-server CXXFLAGS="-O3 -mtune=core2" CFLAGS="-O3 -mtune=core2"
make
```

APPENDIX B – LINUX KERNEL PATCH FOR ARM PMU ACCESS

By default the ARM performance monitoring unit (PMU), is protected in kernel space. It uses the ARM co-processor address CP15 to interface with the ARM core. To access the performance counters from user space the kernel must be modified to allow the oprofile drivers to access the PMU. Using source code from the git repository from kernel.org or Freescale, the following procedure can be used to modify the linux kernel for an ARM Cortex-A8 to enable Oprofile in both "Timer

mode" and "Event mode". Timer mode is useful to determine "hotspots" in the code being executed. It allows the programmer to gain insight into the percentage of time that the processor spends within the different functions and libraries of a program. Event mode is very useful to examine the architectural limitations of the give architecture. Modern processors have performance counters that are capable of incrementing a counter if a specific event occurs such as a cache or TLB miss. These counters give statistical insight to the limitations of an architecture executing a specific application.

7.5 Setting up the Linux kernel for Oprofile in timer mode

a) Selecting the drivers to build into the kernel

```
$ git clone git://sw-git.freescale.net/linux-2.6-imx.git
$ export CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/arm-none-linux-gnueabi-
$ cd linux-2.6-imx/
$ git checkout -b 2.6.38 --track origin/imx_2.6.38
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} distclean
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} imx5_defconfig
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} menuconfig
//described in next step
```

b) Select the kernel modules when GUI based editor appears

```
General Setup ---> Select [*] Profiling Support
General Setup ---> Select <*> OProfile system profiling
Kernel Hacking ---> Select [*] Kernel Debugging
<Exit>
<Exit> and Save will start the build
```

c) Note: Do not select performance counters and monitoring support for timer mode. i.e.:

```
General Setup ---> Kernel Performance Events And Counters  ---> [ ]
Kernel performance events and counters
```

d) Build the new kernel

```
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} uImage
```

e) Find the new kernel

```
// The new kernel image (uImage) is put in ../arch/arm/boot/
```

f) Copy the new kernel to SD

```
$ sudo dd if=uImage of=/dev/sdx bs=512 seek=2048 &sync
```

g) Copy the vmlinux image to the /boot directory of the target rootfs if kernel profiling is desired.

```
$ sudo mount -l /dev/sdx1 /media/sdcard
$ sudo cp vmlinux /media/sdcard/boot/vmlinux
```

h) The follow script can be used to profile programs running on the system for "SleepSeconds" which is set for 100 seconds.

```
SleepSeconds=100
./seti_boinc --standalone &
date
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
```

```
sudo opcontrol --dump
sudo opreport
```

7.6 Setting up the Linux kernel for Oprofile for Event Mode

a) Selecting the drivers to build into the kernel

```
$ git clone git://sw-git.freescale.net/linux-2.6-imx.git
$ export CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/arm-none-linux-gnueabi-
$ cd linux-2.6-imx/
$ git checkout -b 2.6.38 --track origin/imx_2.6.38
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} distclean
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} imx5_defconfig
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} menuconfig
//described in next step
```

b) Select the kernel modules when GUI based editor appears

```
General Setup ---> Kernel Performance Events And Counters  ---> [*]
Kernel performance events and counters
General Setup ---> Select [*] Profiling Support
General Setup ---> Select <*> OProfile system profiling
Kernel Hacking ---> Select [*] Kernel Debugging
<Exit>
<Exit> and Save will start the build
```

c) Modify the kernel source (2.6.38 in this example)

- In /arch/arm/kernel/pmu.c

```
static int __devinit pmu_device_probe(struct platform_device *pdev)
{
        printk("pmu_device_probe called \n" ); //Add this line
        if (pdev->id < 0 || pdev->id >= ARM_NUM_PMU_DEVICES) {
                pr_warning("received registration request for unknown
static int __devinit pmu_device_probe(struct platform_device *pdev)
{
        printk("pmu_device_probe called \n" ); //Add this line
        if (pdev->id < 0 || pdev->id >= ARM_NUM_PMU_DEVICES) {
```

```
static int __init register_pmu_driver(void)
{
	printk("register_pmu_driver called \n"); //Add this line
	return platform_driver_register(&pmu_driver);
}
device_initcall(register_pmu_driver);

struct platform_device *
reserve_pmu(enum arm_pmu_type device)
{
	struct platform_device *pdev;

	if (test_and_set_bit_lock(device, &pmu_lock)) {
		printk("test_and_set_bit_lock() failure \n"); //Add this line
		pdev = ERR_PTR(-EBUSY);
	} else if (pmu_devices[device] == NULL) {
		clear_bit_unlock(device, &pmu_lock);
		printk("pmu_devices[%d] == NULL  \n", device); //Add this line
		pdev = ERR_PTR(-ENODEV);
	} else {
		printk("pmu_devices[%d] reserved !!  \n", device);
		pdev = pmu_devices[device];
	}

	return pdev;
}
int
release_pmu(struct platform_device *pdev)
{
	printk("release_pmu called \n"); //Add this line
	if (WARN_ON(pdev != pmu_devices[pdev->id]))
		return -EINVAL;
	clear_bit_unlock(pdev->id, &pmu_lock);
	return 0;
}
init_cpu_pmu(void)
{
	int i, irqs, err = 0;
	struct platform_device *pdev =
pmu_devices[ARM_PMU_DEVICE_CPU];
```

```
        printk("init_cpu_pmu called \n"); //Add this line
        if (!pdev)
init_pmu(enum arm_pmu_type device)
{
        int err = 0;
        printk("init_pmu called \n"); //Add this line
        switch (device) {
```

- In /arch/arm/mach-mx5/devices.c

```
#include <mach/imx-uart.h>
#include <mach/mx53.h> //Add this line
#include <mach/irqs.h>
#include <asm/pmu.h>
                .dma_mask = &usb_dma_mask,
                .coherent_dma_mask = DMA_BIT_MASK(32),
        },
};
static struct resource pmu_resources[] = { //Add this line
        { //Add this line
                .start = MX53_INT_PMU, //Add this line
                .end = MX53_INT_PMU, //Add this line
                .flags = IORESOURCE_IRQ, //Add this line
        }, //Add this line
}; //Add this line

struct platform_device pmu_device = { //Add this line
        .name = "arm-pmu", //Add this line
        .id = ARM_PMU_DEVICE_CPU, //Add this line
        .num_resources = ARRAY_SIZE(pmu_resources), //Add this line
        .resource = pmu_resources, //Add this line
}; //Add this line

static struct resource usbh2_wakeup_resources[] = {
        {
```

- In /arch/arm/mach-mx5/devices.h

```
extern struct platform_device mxc_usbh1_wakeup_device;
extern struct platform_device mxc_usbh2_wakeup_device;
extern struct platform_device pmu_device; //Add this line
```

- In /arch/arm/mach-mx5/board-mx53_loco.c

```
imx53_add_ipuv3(0, &ipu_data);

mxc_register_device(&pmu_device, NULL); //Add this line

for (i = 0; i < ARRAY_SIZE(loco_fb_data); i++)
```

d) Build the new kernel

```
$ make ARCH=arm CROSS_COMPILE=${CROSS_COMPILE} uImage
```

e) Find the new kernel

```
// The new kernel image (uImage) is put in ../arch/arm/boot/
```

f) Copy the new kernel to SD

```
$ sudo dd if=uImage of=/dev/sdx bs=512 seek=2048 &sync
```

g) Copy the vmlinux image to the /boot directory of the target rootfs if kernel profiling is desired.

```
$ sudo mount -l /dev/sdx1 /media/sdcard
$ sudo cp vmlinux /media/sdcard/boot/vmlinux
```

7.7 Running Oprofile in event mode

a) Setting up RVDS

To use oprofile in events mode you will need to connect the ARM Real View Development Suite (RVDS) debugger to the ARM core via JTAG. In order to run the kernel while connected to the JTAG debugger you must include the kernel argument "jtag=on" in the boot loader arguments. Once this is done you will also need to modify the settings of your ARM Cortex-A8 connection.

- Include 'jtag=on kernel argument (from uboot)
- Disable "semihosting" when running linux (from RVDS)
- Disable the vector catch (from RVDS)
- Change configuration "no reset, no stop" (from RVDS)

b) Booting Linux

- Insert SD card
- Reset board
- Boot to command propt
- Connect RVDS

c) Run command sequence (below)

```
$ sudo opcontrol --init
$ sudo opcontrol --vmlinux=/boot/vmlinux-'uname -r'
$ sudo opcontrol --event=DCACHE_ACCESS:1000:0:1:1
$ sudo opcontrol --start
$ sudo opcontrol --status
// do stuff
$ sudo opcontrol --dump
$ sudo opcontrol --stop
```

8 APPENDIX C - SETTING UP OPROFILE FOR DATA GATHERING

This section describes how to setup oprofile scripts to log event data on the various platforms.

8.1 The ARM Cortex-A8 oprofile script

```
#!/bin/bash
# iMX53 - ARM Cortex A8 Oprofile Script
SleepSeconds=180 #on real run
./seti_boinc --standalone &
```

```
sudo rm -r /var/lib/oprofile
sudo rm -r /root/.oprofile/daemonrc

# Running next event
sudo opcontrol --init
sudo opcontrol --vmlinux=/home/lucid/vmlinux-2.6.38-00462-gc38d9c7-dirty
sudo opcontrol --reset
sudo opcontrol --event=CPU_CYCLES:200000:0x0:0:1 --
event=INSTR_EXECUTED:200000:0x0:0:1 --event=L2_ACCESS:200000:0x0:0:1 --
event=L2_CACH_MISS:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
sudo opcontrol --dump
date >> mx53_Oprofile.txt
sudo opreport >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt
sudo opreport -l /home/lucid/Documents/seti_boinc/client/seti_boinc >>
mx53_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt

# Running next event
sudo opcontrol --init
sudo opcontrol --vmlinux=/home/lucid/vmlinux-2.6.38-00462-gc38d9c7-dirty
sudo opcontrol --reset
sudo opcontrol --event=PC_BRANCH_MIS_PRED:200000:0x0:0:1 --
event=PC_BRANCH_MIS_USED:200000:0x0:0:1 --event=PC_IMM_BRANCH:200000:0x0:0:1 --
event=PC_BRANCH_EXECUTED:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
sudo opcontrol --dump
date >> mx53_Oprofile.txt
sudo opreport >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt
sudo opreport -l /home/lucid/Documents/seti_boinc/client/seti_boinc >>
mx53_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt

# Running next event
sudo opcontrol --init
sudo opcontrol --vmlinux=/home/lucid/vmlinux-2.6.38-00462-gc38d9c7-dirty
sudo opcontrol --reset
sudo opcontrol --event=ITLB_MISS:200000:0x0:0:1 --event=DTLB_REFILL:200000:0x0:0:1 --
event=CYCLES_INST_STALL:200000:0x0:0:1 --event=CYCLES_NEON_DATA_STALL:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
sudo opcontrol --dump
date >> mx53_Oprofile.txt
sudo opreport >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt
sudo opreport -l /home/lucid/Documents/seti_boinc/client/seti_boinc >>
mx53_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> mx53_Oprofile.txt
echo " " >> mx53_Oprofile.txt
```

8.2 The Cortex-A9 oprofile script

```
#!/bin/bash
# ARM Cortex A9 Oprofile Script
SleepSeconds=120 #on real run
date
#rm state.sah
#sudo opcontrol --shutdown
./seti_boinc --standalone &

# Running Timer Mode
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=CPU_CYCLES:200000:0x0:0:1 --
event=INSTR_EXECUTED:200000:0x0:0:1 --event=CLK_INT_EN:200000:0x0:0:1 --
event=PC_BRANCH_MIS_PRED:200000:0x0:0:1 --event=PC_BRANCH_MIS_USED:200000:0x0:0:1 --
event=PC_IMM_BRANCH:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> ARM_Versatile_Oprofile.txt
opreport >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt
opreport -l /home/user41/seti_boinc/client/seti_boinc >> ARM_Versatile_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt



# Running next event
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=CO_LF_MISS:200000:0x0:0:1 --event=CO_LF_HIT:200000:0x0:0:1 --
event=INS_FP_RR:200000:0x0:0:1 --event=INS_NEON_RR:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
sudo opcontrol --dump
date >> ARM_Versatile_Oprofile.txt
opreport >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt
opreport -l /home/user41/seti_boinc/client/seti_boinc >> ARM_Versatile_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt



# Running next event
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=ITLB_MISS:200000:0x0:0:1 --event=DTLB_REFILL:200000:0x0:0:1 --
```

```
event=STALL_INS_TLB:200000:0x0:0:1 --event=STALL_DATA_TLB:200000:0x0:0:1 --
event=STALL_INS_UTLB:200000:0x0:0:1 --event=STALL_DATA_ULTB:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
sleep $SleepSeconds
sudo opcontrol --dump
date >> ARM_Versatile_Oprofile.txt
opreport >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt
opreport -l /home/user41/seti_boinc/client/seti_boinc >> ARM_Versatile_Oprofile.txt
sudo opcontrol --shutdown
echo " " >> ARM_Versatile_Oprofile.txt
echo " " >> ARM_Versatile_Oprofile.txt
```

8.3 The Intel Atom oprofile script

```
#!/bin/bash
SleepSeconds=292
# atom processor

./seti_boinc --standalone &

sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=CPU_CLK_UNHALTED:200000:0x0:0:1 --
event=INST_RETIRED:200000:0x1:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Atom.txt
sudo opreport >> Atom.txt
echo " " >> Atom.txt
sudo opreport -l /home/xbmc/Documents/boinc/seti_boinc/client/seti_boinc >> Atom.txt
sudo opcontrol --shutdown



sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=LLC_MISSES:200000:0x41:0:1 --event=LLC_REFS:200000:0x4f:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Atom.txt
sudo opreport >> Atom.txt
echo " " >> Atom.txt
sudo opreport -l /home/xbmc/Documents/boinc/seti_boinc/client/seti_boinc >> Atom.txt
sudo opcontrol --shutdown
```

```
# 1)DTLB - any miss 2)
sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=DATA_TLB_MISSES:200000:0x7:0:1 --event=ITLB:200000:0x02:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Atom.txt
sudo opreport >> Atom.txt
echo " " >> Atom.txt
sudo opreport -l /home/xbmc/Documents/boinc/seti_boinc/client/seti_boinc >> Atom.txt
sudo opcontrol --shutdown

# 1)DTLB - any miss 2)
sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=BR_INST_RETIRED:200000:0x0:0:1 --
event=BR_MISS_PRED_RETIRED:200000:0x0:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Atom.txt
sudo opreport >> Atom.txt
echo " " >> Atom.txt
sudo opreport -l /home/xbmc/Documents/boinc/seti_boinc/client/seti_boinc >> Atom.txt
sudo opcontrol --shutdown
```

The Intel Core-i3 oprofile script

```
#!/bin/bash
SleepSeconds=60

./seti_boinc --standalone &
sleep 10
# Running Timer Mode
sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=CPU_CLK_UNHALTED:200000:0x0:0:1 --
event=INST_RETIRED:200000:0x0:0:1 --event=LLC_MISSES:200000:0x41:0:1 --
event=LLC_REFS:200000:0x4f:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
```

```
sudo opcontrol --dump
date >> Nehalem.txt
sudo opreport >> Nehalem.txt
echo " " >> Nehalem.txt
sudo opreport -l /home/jeff/stuff/seti_boinc/client/seti_boinc >> Nehalem.txt
sudo opcontrol --shutdown

# 1)DTLB - any miss 2)
sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=DTLB_LOAD_MISSES:200000:0x1:0:1 --
event=DTLB_MISSES:200000:0x1:0:1 --event=ITLB_MISSES:200000:0x1:0:1 --
event=ITLB_MISS_RETIRED:200000:0x20:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Nehalem.txt
sudo opreport >> Nehalem.txt
echo " " >> Nehalem.txt
sudo opreport -l /home/jeff/stuff/seti_boinc/client/seti_boinc >> Nehalem.txt
sudo opcontrol --shutdown

# 1)DTLB - any miss 2)
sudo rm -r /var/lib/oprofile/
sudo rm -r /root/.oprofile/daemonrc
sudo opcontrol --init
sudo opcontrol --no-vmlinux
sudo opcontrol --reset
sudo opcontrol --event=BR_INST_RETIRED:10000:0x0:0:1 --
event=BR_MISS_PRED_RETIRED:10000:0x0:0:1 --event=RESOURCE_STALLS:2000000:0x1:0:1 --
event=UOPS_EXECUTED:2000000:0x3f:0:1
sudo opcontrol --start
sudo opcontrol --status
echo "profiling for $SleepSeconds seconds"
sleep $SleepSeconds
sudo opcontrol --dump
date >> Nehalem.txt
sudo opreport >> Nehalem.txt
echo " " >> Nehalem.txt
sudo opreport -l /home/jeff/stuff/seti_boinc/client/seti_boinc >> Nehalem.txt
sudo opcontrol --shutdown
```

9 APPENDIX E – LOGGING POWER WITH AN AGILENT SUPPLY

This script was used to sample system level power of the evaluation platforms.

```
#### Control Script for Agilent/HP E3632A ############
#!/usr/bin/perl

use Device::SerialPort;
```

```
my $port = new Device::SerialPort("/dev/ttyS0") || die "can't open ttyS0\n";

$port->baudrate(9600);
$port->parity("none");
$port->handshake("none");
$port->databits(8);
$port->stopbits(2);
$port->read_char_time(0);
$port->read_const_time(1);

### DEBUG Variables
#  Set DEBUG to 1 to print all of the debug statements
my $HP_DBG = 1;   my $DEBUG = 1;
my $delay;
if ($DEBUG){ print "Power Log Started\nCurrent,\tTime\n";}



####  Control Agilent/HP E3632A ############
&send_cmd('*RST');                # reset
&send_cmd('SYSTem:REMote');       # remote control necessary for rs232 control


#&send_cmd("VOLTage:RANGe P30V"); # Use this supply for larger voltage / lower
current
&send_cmd("VOLTage:RANGe P15V");  # Use this supply for lower voltage / larger
current

#&send_cmd("APPL 19.0, 4.0");  #Intel Atom and Core_i3 Set to 19V rated at 4A
&send_cmd("APPL 5.0, 7.0");  #ARM Cortex-A9 Set to 5V rated at 7A
&send_cmd("OUTP ON");             #turn ON

$current=0;

open DATA_OUT, ">CortexA8_SD2_neon_fftw_current.txt" or die $!;  # open file, erase
previous
close DATA_OUT;

while(1){
        open DATA_OUT, ">>CortexA8_SD2_neon_fftw_current.txt" or die $!;  # open file
APPEND
        &send_cmd("MEASure:CURRent?");
        $delay=time+.1;                 #delay to allow read to finish
        while(time<$delay){};
        $current = $port->lookfor;      # look for data from rs232
        ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);

        printf DATA_OUT "%f, %d:%d:%d\n",$current,$hour, $min, $sec;
        printf "%f, %d:%d:%d\n",$current,$hour, $min, $sec;
        $current=0;
        close DATA_OUT;
}
if ($DEBUG){ print "TEST ENDED\n";}



## fucntion written assuming $RS232 object was already created
```

```
sub send_cmd {
        my $cmd=shift;
        my $delay;
        #$RS232->write_done(0);
        $port->write("$cmd\r\n");
        $delay=time+.05;           #delay to allow command to finish sending
        while(time<$delay){};
}
```

10 REFERENCES

1. **Cisco Systems, Inc.** [Online] June 2011. http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=324003.

2. **Freescale Semiconductor.** Freescale Multimedia Applications Processor Division. [Online] www.freescale.com/imx.

3. **http://mersenne.org.** Great Internet Mersenne Prime Search. [Online] http://mersenne.org/various/history.php.

4. **World Community Grid.** [Online] October 2011. http://www.worldcommunitygrid.org/forums/wcg/viewthread?thread=15715.

5. **Anderson, David P.** *BOINC: A System for Public-Resource Computing and Storage.* Berkeley, CA : University of California at Berkeley, 2004. pp. 1-7.

6. **boinc.** Berkeley Open Infrastructure for Network Computing. *BOINC.* [Online] November 3, 2010. http://boinc.berkeley.edu/.

7. **Sterling, Prof. Thomas.** High Performance Computing. *Lecture 5, Capacity Computing.* s.l. : Louisiana State University, 2011.

8. *Cortex-A8: High Speed, Low Power.* **Baron, Max.** 2005, Microprocessor Report, pp. 1-7.

9. **ARM.** Cortex-A9 Technical Reference Manual. *arm.com.* [Online] 2010.

10. *Intel's Tiny Atom - New Low-Power Microarchitecture Rejuvenates the Embedded x86.* **Halfhill, Tom.** 2009, Microprocessor Report, pp. 1-12.

11. **CPU World.** Intel Core i3 Mobile i3-330M. *CPU World.* [Online] CPU World, 2010. [Cited: October 1, 2011.] http://www.cpu-world.com/CPUs/Core_i3/Intel-Core%20i3%20Mobile%20I3-330M%20CP80617004122AG.html.

12. **Rolf, Trent.** *Cache Organization and Memory Management of the Intel Nehalem Computer Architecture.* s.l. : University of Utah, 2009.

13. **Thomadakis, Machael E.** *The Architecture fo the Nehalem Processor and Nehalem-EP SMP Platforms.* College Station : Texas A&M University, 2011.

14. **ARM Holdings.** CoreTile Express A9x4 - For the Versatile Express Family. *ARM corporate website.* [Online] www.arm.com.

15. *OMAP4430 Architecture and Development.* **Witt, David.** s.l. : Texas Instruments, 2009. Hot Chips Symposium. pp. 1-16.

16. *More Applications for OMAP4.* **Baron, Max.** 2009, Microprocessor Report, pp. 2-6.

17. **Texas Instruments.** Panda Board. *Panda Board.* [Online] 2011. www.pandaboard.com.

18. *CPU-IP Cores Fight for Dominance - Licensable Processors and Architectures Battle the x86 in 2011.* **Halfhill, Tom R.** 2011, Microprocessor Report, p. 1.

19. **ZOTAC USA.** ZOTAC. [Online] 2011. http://www.zotacusa.com.

20. **University of Virginia.** STREAM: Sustainable Memory Bandwidth in High Performance Computers. *STREAM.* [Online] Department of Computer Science . http://www.cs.virginia.edu/stream/.

21. **CBS News.** Number of Cell Phones Worldwide Hits 4.6B. *CBS News Online.* [Online] February 2010. http://www.cbsnews.com/stories/2010/02/15/business/main6209772.shtml.

22. **ClimatePrediction.net.** [Online] 2002-2010. www.climateprediction.net.

23. **Cosmology@Home.** www.cosmologyathome.org. *www.cosmologyathome.org.* [Online] 2011.

24. **Moore, Gordon.** 1965 - "Moore's Law" Predicts the Future of Integrated Circuits. *Computer History Museum.* [Online] 1965. http://www.computerhistory.org/semiconductor/timeline/1965-Moore.html.

25. **MalariaControl.net.** MalariaControl.net. *MalariaControl.net.* [Online] 2011. http://www.malariacontrol.net.