ABSTRACT

PAL, POULOMI. Scheduling to Consolidate Idle Periods for Energy Efficiency in Multicore Systems. (Under the direction of Dr. Gregory T. Byrd.)

Power-efficiency and energy savings are the major drivers in the CPU design space. Design at all levels needs to be energy-aware in order to be considered seriously. With the emergence of mobile devices as a large market, energy efficiency has become of prime importance, as the emphasis now lies on making the battery last longer. Most modern cell phones are required to do much more than making and receiving calls. With 4G approaching quickly, the cell phone is required to be almost as good as any general purpose computer with respect to application complexities. This gives rise to the need for adaptive systems that can handle applications with high performance, and conserve energy as well.

The research presented here provides a scheduling scheme which aims to consolidate CPU idle times, and to introduce some amount of inertia and determinism in the system, so as to provide the opportunity to switch CPUs into low-power modes for conservation of energy. The baseline used here is the Linux kernel-2.6.28, which implements distributed control for load balancing for different CPUs. Changes are made to this kernel, and CPU activity is used as the metric for comparison. It is found that the CPUs that are numbered higher get much longer idle periods, which can be used for switching to energy-efficiency mode, while still maintaining comparable performance.

Scheduling to Consolidate Idle Periods for Energy-Efficiency in Multicore Systems

by
Poulomi Pal

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2009

APPROVED BY:

_______________________	_______________________
Dr. Eric Rotenberg	Dr. Rhett Davis

____
Dr. Gregory T. Byrd
Chair of Advisory Committee

BIOGRAPHY

Poulomi Pal was born in India on the 28th of May, 1984. She received her Bachelor of Engineering degree in Electronics and Telecommunication Engineering in 2006 from Rashtreeya Vidyalaya College of Engineering, Visvesvaraya Technological University, India. In fall of 2007, she joined North Carolina State University as a graduate student. She has been working under the guidance of Dr. Greg T. Byrd at the Center for Efficient Scalable and Reliable Computing since Fall, 2008.

Her primary research interests lie in energy-efficient systems and architectures. Currently she is working on a project involving scheduling for idle time consolidation for energy-efficiency on multicore processors at the Center for Efficient Scalable and Reliable Computing, NCSU. With the defense of this thesis, she is receiving the degree of Master of Science in Computer Engineering from NCSU.

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my thanks to Dr. Gregory T. Byrd for his invaluable guidance throughout my thesis research. I am thankful to him for his valuable suggestions and insights that have shaped the course of my thesis. He has been a source of inspiration and working under his guidance, I have also learned the importance of having well defined goals and objectives, planning, and constantly striving ahead to bring out the best in oneself.

I am grateful to Dr. Eric Rotenberg and Dr. Rhett Davis for their insightful remarks and for kindly agreeing to be on my thesis committee.

I am thankful to Mike Morrow and Kevin Sapp from Qualcomm, for giving me an opportunity to work on this project and their guidance for the same.

I also acknowledge the help of Sabina Grover and other members of the lab, who were always ready to help me out whenever I faced a problem.

None of this would have been possible without my family. They have been a source of tremendous strength and inspiration. Last, but not the least, I would like to thank all my friends who have always been there when I needed them.

TABLE OF CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

Chapter I – Introduction 1

I. RELATED WORK 4

A. DYNAMIC POWER MANAGEMENT 5

B. WORK IN THE HIGH PERFORMANCE COMPUTING SPACE 5

C. OTHER WORK ON DVFS SCHEDULING 6

D. MULTICORE AND THE OPERATING SYSTEM 8

II. THESIS LAYOUT 9

Chapter II - Algorithms and Implementation 11

I. SYSTEM MODEL 12

A. COST ANALYSIS FOR MaxRunQLength 14

II. CPU IDLE ALGORITHM 17

A. IMPLEMENTATION 18

B. FURTHER MODIFICATION 19

Chapter III - Experiments 21

I. EXPERIMENTAL SETUP 21

A. PLATFORM 21

B. OPERATING SYSTEM 24

C. BOOTING UP THE BOARD 24

D. LOGGING INTO THE BOARD REMOTELY 26

E. CPU ACTIVITY MEASUREMENTS 26

II. BENCHMARKS 28

III. RESULTS 30

Chapter IV – Conclusion and Future Work 36

REFERENCES 39

LIST OF TABLES

Table 1: Applications from the MIBench 29

Table 2: Applications from the PARSEC Benchmark 29

Table 3: PARSEC Results 30

Table 4: MIBench Results 32

LIST OF FIGURES

Figure 1: Desired CPU schedule 13

Figure 2: Random CPU schedule 13

Figure 3: ARM11MPcore (courtesy ARM) 22

Figure 4: CPU activity graphs 28

Figure 5: CPU3 idle times for PARSEC 31

Figure 6: CPU2 idle times for PARSEC 31

Figure 7: CPU1 idle times for PARSEC 31

Figure 8: CPU0 idle times for PARSEC 31

Figure 9: CPU3 idle times for MIBench 33

Figure 10: CPU2 idle times for MIBench 33

Figure 11: CPU1 idle times for MIBench 33

Figure 12: CPU0 idle times for MIBench 33

Figure 13: Runtime comparisons for PARSEC 34

Figure 14: Runtime Comparisons for MIBench 35

Chapter I – Introduction

Power consumption has been the factor driving many innovations in the CPU design and utilization space during recent years. Efforts are being made at different levels, from the device level up to the application level, to bring down the power consumption of the systems. At the architecture level, with the uniprocessor being higher in power consumption with diminishing performance returns, multiprocessor systems have gained more importance. It can be more energy-efficient to distribute the processing over multiple simpler processors than on a large complex processor. Multicores, a class of multiprocessors with multiple processor cores on the same chip, provide many additional features such as shared cache and low latencies, thus providing multiple processing units while reducing the associated penalties of communication and synchronization.

With the emergence of mobile devices as a large market, energy efficiency has become of prime importance, as the emphasis now lies on making the battery last longer. Most modern cell phones are required to do much more than making and receiving calls. With 4G approaching quickly, the cell phone is required to be almost as good as any general purpose computer with respect to application complexities. It should be able to support complex graphics, media conversion, web browsing and so on. This necessitates that the device be power efficient while not sacrificing performance to achieve this goal.

A method to achieve energy efficiency is to vary the power state of a machine dynamically with the workload that it is processing. Power state here refers to the level of power that is consumed by the system. Varying the number of active CPUs is a way of changing the active state of a system. Inactive CPUs can go into idle modes in order to conserve energy in cases where the load is light. There are various idle modes in modern-day CPUs - Wait For Interrupt (clock suspended), dormant (but cache state maintained), and completely switched off.

Each idle level, while associated with an energy advantage, is also associated with a proportional transition penalty from and to the active state. Hence, there should be sufficient motivation to change from one state to another. In other words, the advantage gained from going into the idle mode should be greater than the overhead introduced due to the transition. Hence it should be ensured that there is greater inertia in CPU states; that is, CPUs that are already loaded should keep accepting loads as long as they can maintain the performance requirement, before scheduling to a new CPU.

The current Linux scheduler, kernel 2.6.28, achieves load balancing using distributed control of load balancing routines on each core. This leads to a highly random system since there is no control over the scheduling decisions. Ensuring inertia in such a system in an unrestricted system as this is not possible. Hence, we need to apply a simple rule: Do not load a higher CPU when there is an inactive or lightly-loaded lower CPU that can handle the

load with the required performance. This thesis deals with a simple heuristic to make the proper decision with the change in the system load.

The work presented here is particularly beneficial for high-end cell phones, however not requiring any change in the application-level code that already exists. Unlike most other works that deal with preset, pre-analyzed loads, this work deals with a more consumer-relevant environment, where the load is non-deterministic. The technique devised here is more reactive in nature and varies with the load running on the system. The method suggested here can also be combined with circuit-level power features, such as Dynamic Voltage and Frequency Scaling (DVFS).

Dynamic Voltage and Frequency Scaling is a technique which changes the clocking frequency and source voltage of the CPU according to the load it is processing. The CPU is scaled up to accommodate a larger load, and scaled down in case of a very sparse load. This technique is orthogonal to the method of turning off CPUs in order to conserve power. DVFS is instrumental in conserving dynamic power, while switching off CPUs conserves static power as well

Presented here is a scheme where the desired load on an individual CPU is initially fixed. This value is varied and compared to the original Linux scheduler behavior to see if longer idle times and a more deterministic behavior are achieved. The scheme is to limit the number of processors that can be used, depending on the load. The load is represented by the

number of the processes ready to run on each CPU. It is to be noted that use of such simple, easily available data makes the scheme extremely lightweight and with very little overhead.

Through experiments using a multicore development board, we show that the scheduling algorithm increases the length of idle times on higher-numbered CPUs dramatically. It is also seen to compare very well with the static schemes, where the number of CPUs is fixed. In terms of the time consumption the basic algorithm is comparable to the original Linux kernel. We therefore have a scheme that creates opportunity for energy-efficiency, while maintaining performance levels.

I. RELATED WORK

A substantial amount of work has been done in the field of power management and how it can be used in Uniprocessors and Multiple processor systems for energy and power efficiency. There is work dealing with all aspects of this, including hardware support and suggested code changes. However in this paper, the system-level aspects will be concentrated on. These include different scheduling algorithms and heuristics as well as a discussion on what changes in OS are required for Multi-core support.

A. DYNAMIC POWER MANAGEMENT

Benini et al [18] [19] [20] talk of energy savings by putting idle components of a system in sleep mode. They talk about the penalties associated with waking up the processor after sleep state, and if the idle period is not long enough, there might be higher energy consumption owing to the switching penalty. There is a discussion on how the processing element is put in sleep mode only after having been in idle state for a while. This scenario serves as the basis for a system on which idle-time consolidation leads to energy saving, by ensuring that lightly loaded processors go to sleep.

Golding et al [22] talk about advantages of idle mode in a system. These times can be used for speculative work. They also point out that the system can go into a low-power state in these times to conserve battery power.

B. WORK IN THE HIGH PERFORMANCE COMPUTING SPACE

A lot of work done in HPC can be applied to multicore mobile systems as well. Freeh et al [1] investigate the energy consumption and execution time of applications from a standard benchmark suite (NAS) on a power-scalable cluster. They study via direct measurement and simulation both intra-node and inter-node effects of memory and communication bottlenecks, respectively. Additionally, they compare energy consumption

and execution time across different numbers of nodes. Their results demonstrate the potential of power-scalable clusters to save energy. Also on certain cases, there is found to be a speedup due to parallel execution, in addition to energy savings. They further develop a model for Energy-Time trade-off.

Kapiah et al [2] explore the possibility of lowering the frequency of lightly-loaded processors in a power scalable cluster. The slack time is utilized to spread the execution of the faster task, at a lower frequency. This approach provides just-in-time response for the task instead of leaving it idle. This would be very effective in clusters where perfect load-balancing does not occur.

Lim et al [3] devise an algorithm to adaptively scale the frequency and voltage in MPI applications to give minimum EDP. No prior profiling is required for this. All the training and scaling occurs during the MPI period, hence the application remains unaffected.

C. OTHER WORK ON DVFS SCHEDULING

Qu [9] deals with the scheduling algorithm from a mathematical perspective. Though the problem of multicore embedded systems is NP-hard, in real life applications, a few cases can be solved. From the theoretical point of view, this paper serves as a foundation for

power-management of multi-core multi-voltage systems while practically it studies real life applications for optimal solutions.

Luo and Jha [8] examine systems with heterogeneous processing elements with voltage and frequency scaling to find an optimal scheduling technique for multirate periodic tasks. Execution order optimization is done on such task graphs to come up with power-efficient schedules.

DVFS-related work has been done specifically for real time applications as well. Bautista et al [10] present a scheduling algorithm which is ideal for multi-core systems with homogeneous scaling across cores. Initially all cores start from the lowest frequency. When a new task T needs to be scheduled, it is checked if it can be accommodated at the current frequency on any processor. If not, the frequency is raised. When a task finishes, it is taken off the schedule. Now if all processors are found to have slack, the frequency is reduced so that it just delivers in time the tightest schedule.

An additional constraint while scheduling tasks on multicore processors is the cache accesses. Since most multi-cores have shared L2, if care is not taken for cache-awareness, the actual execution time of the schedule will be far worse than expected. Anderson et al look into this aspect [17].

In a system, where periodic tasks are scheduled e.g. Rate-monotonic schedules, these repeating tasks are similar to a loop in a program. Thus, loop scheduling algorithms can be

looked into for possible solutions for scheduling periodic tasks. These algorithms are based on both rotation scheduling and DVS. Dynamic loop scheduling algorithms are also presented in the work [5] [6].

D. MULTICORE AND THE OPERATING SYSTEM

Some of the work [14] deals with the enhancements in the 2.6 version of Linux, especially the better SMP scalability, which makes it more suitable to use with Multicores than earlier versions. Siddha also points out the need to have explicit Multicore capability, rather than club it with the SMP, the most important being the need to support a heterogeneous system, where each CPU can be in a different P and C state. In current Intel architectures, P states are the different operational states and C states are the different idle states.

Knauerhase et al [13] talk about how current OS are inadequate to handle the architectural complexities of Multicores. They suggest that dynamic runtime data collected by the OS can be used to improve performance and more effectively use multicore resources. The authors establish the effectiveness of an observation-based approach.

In Linux-specific work, Shi et al [17] modify the load-balance algorithm and scheduling domains for asymmetric multi-processing systems. All systems with shared L2

caches are put in the same scheduling domain. This ensures that migration occurs between CPUs that share the L2 cache, hence minimizing the migration penalties.

II. THESIS LAYOUT

In this present work, a lightweight algorithm has been used to schedule on the ARM11mpcore platform. The algorithms consider total runqueue length as the indicator of the e load on the system. This approach though extremely simple, introduces very low overhead as it uses data easily accessible by the scheduler.

The implemented scheme involves the scheduler deciding how many CPUs the execution needs to be limited to, depending on the load. The desired load per CPU is statically set in the beginning and a restraint is put on the system that if there are lower CPUs in the system that can handle the load, higher CPUs should not be assigned any tasks. This scheme is discussed in detail in chapter II. Also discussed are the implementation details.

Chapter III deals with the experimental setup, the platform and OS used. The setup explains the power measurement techniques and the benchmarks used. The implemented kernel is compared against the original to show the determinism of the former. Also it is

compared to static schemes to compare utilization. This is followed by the results of the experiments, followed by the conclusion and future work discussion in chapter IV.

Chapter II - Algorithms and Implementation

In this chapter, we present the scheduling algorithm that has been used to consolidate idle periods on the ARM11mpcore platform. The algorithm considers total runqueue length as the indicator of the load on the system. This indicator, though extremely simple, was chosen by the virtue of the ease of access and the timeliness of the data available.

Other alternative indicators of the load were considered as well, such as profiling data generated by oprofile or the data present in the /proc file system. However, both those options require file accesses which would be an excessive overhead at the frequency at which the data needs to be accessed. Also since the data present in the files is sampled and updated at intervals, it is probable that the data in the file will be obsolete. Runqueue length that is available to the scheduler at any point of time is instantaneous data and therefore more representative of the load of the system at the given time. Also since it is present in variables accessible to the scheduler, it can be accessed with maximum ease.

Presented here is a scheme where the desired load on an individual CPU is set statically. This value (MaxRunQLength) is experimented to build different kernels which are compared to the original Linux kernel to see if longer idle times and a more deterministic

behavior are achieved. The scheme is to limit the number of processors that can be used for thread scheduling, depending on the load. Threads will be assigned to lower-numbered CPUs as long as their loads do not exceed MaxRunQLength.

I. SYSTEM MODEL

Let us consider a system with N processors. For sake of simplicity, let us assume that each of the CPUs can run in two states – normal execution state and idle state. Using a system-wide value for desired threads per CPU (MaxRunQLength), we can ensure that a new CPU is brought up from idle state only if the load (in our case, number of ready-to-run threads) is greater than the product of number of active CPUs and MaxRunQLength.

Consider a case with eight threads (T1 through T8). T1 through T5 are scheduled initially, and T6 and T7 become ready to run after completion of T5. Finally T8 is introduced to the system one time unit after the completion of T6 and T7. If MaxRunQLength=1, we observe the schedule as shown in figure 1.

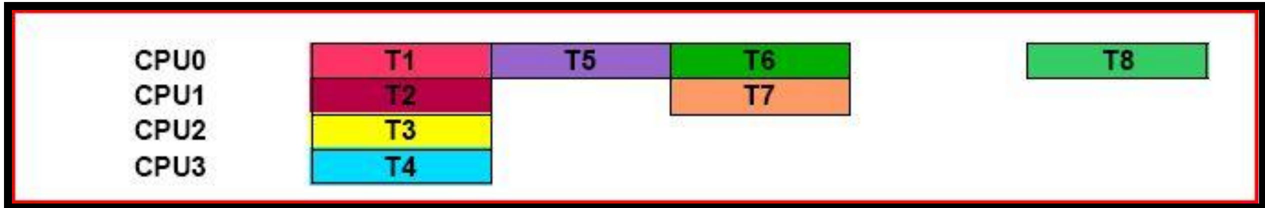

Figure 1: Desired CPU schedule

Note that the lowest-numbered CPU is chosen when possible. This leaves the higher numbered CPUs such as CPU3, idle for a longer period.

In a regular Linux scheduler, the above tasks may run in a random manner such as given in Figure 2.

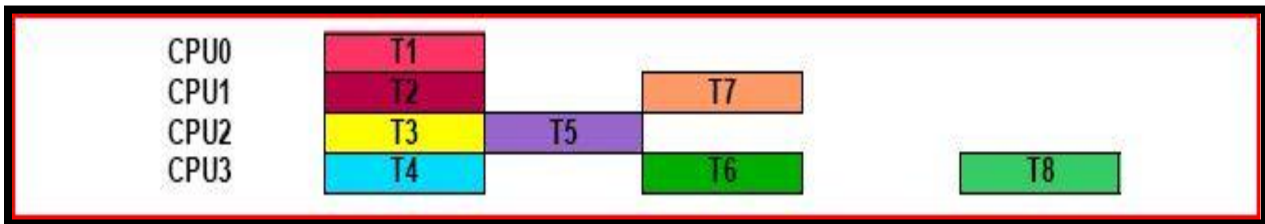

Figure 2: Random CPU schedule

Though the above shown schedule also succeeds in completing the task, it is impossible to predict which CPU shall start executing a given thread. It might even happen that though there might be no more than one thread on the system, the CPUs may be active in a cycle (e.g. CPU0, CPU1, CPU2, CPU3, CPU0,.....) with none of them getting sufficient idle time to go to a lower power mode.

We can ensure a more deterministic behavior by arranging the CPUs in order of preference to get the execution of a thread. For example, if there is only one thread, schedule it on CPU0. If there are two, balance between CPU0 and CPU1, and so on. This will effectively consolidate the system's idle periods onto the higher-numbered CPUs. Larger idle periods allow these CPUs to enter into power-saving idle modes and to minimize the transitions between active and idle modes.

A. COST ANALYSIS FOR MaxRunQLength

The value for MaxRunQLength is an important factor that decides the energy efficiency and the performance of the system. A lower MaxRunQLength results in a higher number of active CPUs needed for a given load and therefore higher power. Also, the MaxRunQLength value is proportional to the response time of a task, hence inversely proportional to the performance. A detailed cost analysis is required to set this value.

The energy consumed by the system is directly proportional to two factors, the power and the time required finishing the job. Power of a system is of two kinds – static and dynamic. Dynamic Power depends on the supply voltage and the frequency at which the system is clocked. If the load on the system is low, it can be switched to a low voltage-frequency mode, as is done in Dynamic Voltage and Frequency Scaling (see Chapter I for

discussion). Static power is due to leakage currents whenever any component is switched on, hence the only way to counteract that would be to switch the component off. As we see here, rise in power is either due to higher frequency, or due to an additional processing element, and represents the increase of processing capability of the system. This in turn makes power rise inversely proportional to the time that is required for the completion of a task.

Here is an analysis of the factors that determine statically what the MaxRunQLength value should be. The two major factors for the energy restraints on MaxRunQLength are the power saved in idle state, and the penalty associated with bringing an idle CPU up to the running state. The MaxRunQLength value should be such that, if a decision is made to bring the idle CPU up, the time saved to run the threads should be able to save the energy spent in making the transition as well as compensate for the time penalty of the transition.

Let, Pon = Power dissipation of element in ON state

Poff = Power dissipation of element in OFF state

Pother= Power dissipation of other elements

Etrans= Energy lost in transition

Ttask= Time taken for task without additional element

Ttrans = Time taken for transition

Tgain = Time gained by bringing up an additional processing element

So **Tgain>= Ttrans**, to make the transition time-efficient.

For energy–efficiency,

[(Pother+Poff) x Ttask]>=[Etrans + Pother x Ttrans + (Pother+Pon) x (Ttask-Tgain)]

Similarly, a normal state element should be sent into low-power mode, only if the length of idle time can justify the transitional energy loss.

(Pon-Poff) x Tidle >= 2 x (Etrans + Pother x Ttrans) [two transitions – beginning and end]

The above equation shows that the length of idle time is an important factor in the decision to power down a CPU to its idle state.

The following section describes the method of consolidation of idle times with a statically set MaxRunQLength value. There are factors that may require the MaxRunQLength to be changed dynamically as well. These shall be discussed in a later chapter.

II. CPU IDLE ALGORITHM

This algorithm restricts the number of CPUs that can be used to schedule the processes depending on the total number of processes in the system that are ready to run. The basic algorithm is very simple with MaxRunQLength = desired number of threads to be run per CPU. If the number of threads to active CPU ratio increases beyond the set threshold, another CPU is activated. The major components of this algorithm are:

a. level - This is the system level depending on the number of running processes. This has been made equivalent to the number of active CPUs to reduce extra calculations.

b. MaxRunQLength - This is the optimal number of processes that can be assigned per cpu.

Level = (number of ready-to-run processes -1)/ MaxRunQLength

A. IMPLEMENTATION

The implementation of the algorithm requires changes in all the routines where load balance occurs. The routines are as follows:

self_balance: This routine is called once a new process is created and scheduled. The change made here is that if the calculated CPU is greater than the current level, the process is scheduled onto the CPU whose number is equal to the current level.

idle_balance: This routine is called once a CPU is about to become idle. The change required in this case is to check whether the CPU calling this routine is greater than the

current level. If so, the routine returns to caller, without pulling a task onto the CPU for execution.

load_balance: This routine is called by an idle CPU to take over execution of processes from overloaded CPUs. The change required in this case is to check whether the CPU calling this routine is greater than the current level. If so, the routine returns to caller, without pulling a task onto the CPU for execution.

B. FURTHER MODIFICATION

Further modification can be made in the given algorithm. Instead of reacting to a change in the level of the system immediately, it may check to see if the level has been maintained for a while before making the change. This is done to ensure that the system follows stable trends rather than getting caught in transients.

Instead of having local variables for the level, as is sufficient in above algorithm, a global value is kept so that it can be changed by all the three routines listed earlier. At every instance of level-calculation, it is checked if the current value continues the trend beyond a threshold value. If yes then the level is changed to the calculated value.

Different values of MaxRunQLength and threshold count have been experimented with to come up with an effective configuration for systems and workloads used in this thesis.

Chapter III - Experiments

I. EXPERIMENTAL SETUP

A. PLATFORM

The platform used for the experiments is the ARM11MPCore on a Realview Emulation Baseboard.

The baseboard is a highly integrated development board. The baseboard has two slots for core tiles that need to be tested. The *Field Programmable Gate-Array* (FPGA) implements a bus matrix, configuration interface, peripheral controllers, and interface logic. An 8MB configuration flash holds FPGA images. Other features available on board such as 256MB of 32-bit wide DDR SDRAM and Ethernet interface controller IC and connector. More details of the baseboard can be found in the Emulation Baseboard User Guide [22].

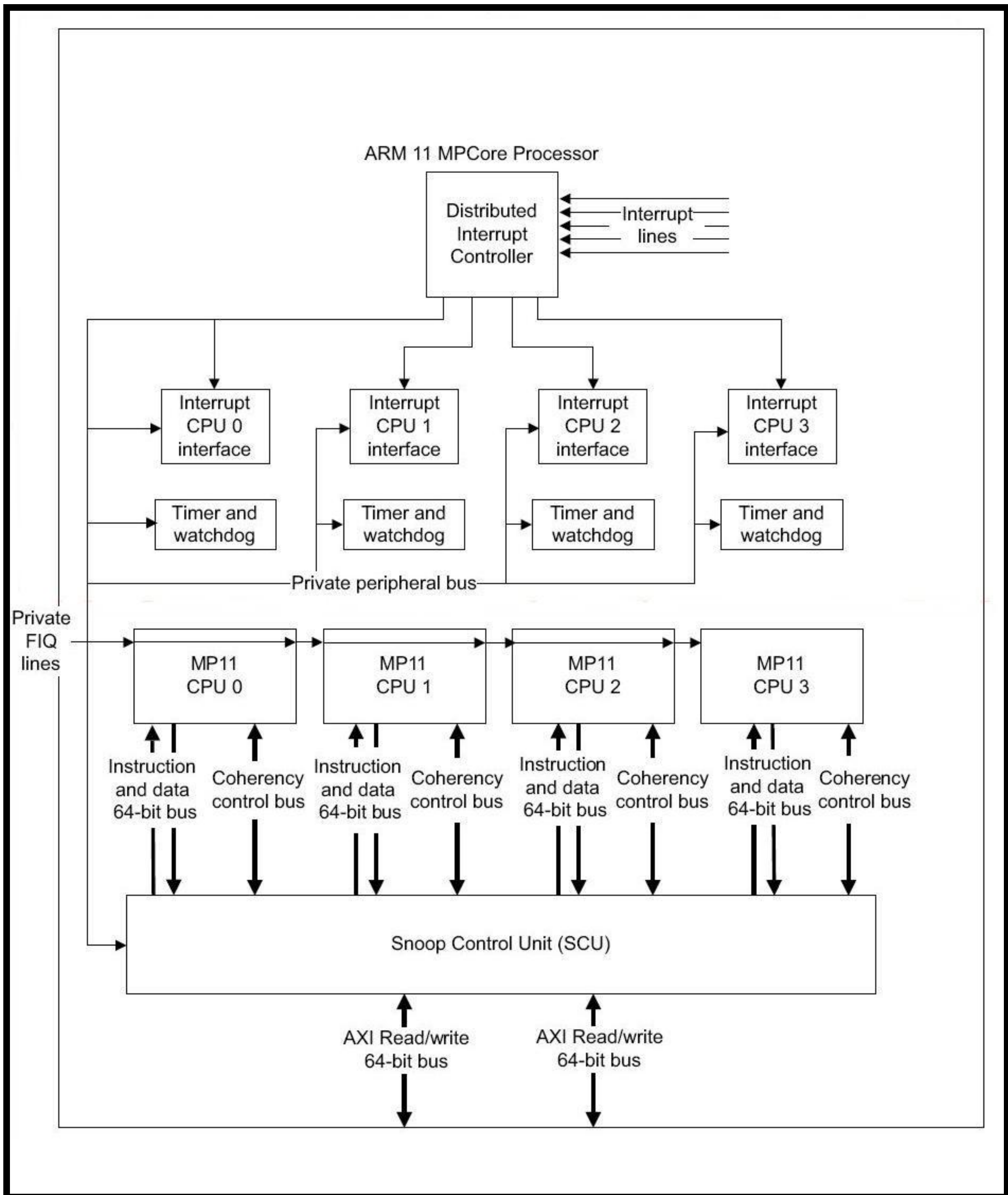

Figure 3: ARM11MPcore (courtesy ARM)

The processor has four MP11 CPUs that implement the ARM architecture v6K. It is a high-performance and low-power ARM cached multiprocessor macrocell with full virtual memory capabilities. The processor features has up to four MP11 CPUs, a *Snoop Control Unit* (SCU) responsible for maintaining coherency among MP11 CPUs level 1 data caches, a Distributed Interrupt Controller with support for legacy ARM interrupts, among other features. The shared L2 cache between the 4 CPUs ensures that there is minimum penalty in case of migration.

Each individual CPU features an 8-stage pipeline, branch prediction with return stack, Instruction and Data *Memory Management Units* (MMUs), managed using MicroTLB structures backed by a unified main TLB, instruction and data caches, including a non-blocking data cache with *Hit-Under-Miss* (HUM), a data cache that is physically indexed, physically tagged, write back, write allocate only, an instruction cache that is virtually indexed, physically tagged, 32-bit interface to the instruction cache and 64-bit interface to the data cache, hardware support for data cache coherency, *Vector Floating-Point* (VFP) coprocessor support and JTAG-based debug. More such details can be found in the ARM11 MPCore™ Processor Technical Reference Manual [23].

B. OPERATING SYSTEM

The base OS used for this thesis is Linux 2.6.28 with SMP capabilities and the latest ARM patch. The system boots through tftp and uses a network file system, hosted by a server (billie.ece.ncsu.edu). The base file system is a cramfs; however the file system used for the experimental work is the underlying Debian file system.

Currently billie.ece.ncsu.edu acts as a dhcp server and is connected to the ARM PB11MPCore, giving it access to the Internet as well. This facilitates the installation of new packages.

C. BOOTING UP THE BOARD

The ARM PB11MPCore is turned on by a regular Power button on the front panel. On powering on, it boots up and gives a prompt. The stdin, stdout are directed to the SER0 port of the board. This is connected to ttyS0 of billie.ece.ncsu.edu. A terminal emulator like minicom can be used to access the prompt by listening to the serial port at 38400baud with no parity and 1 stop bit.

At the terminal:

```
> flash run uboot
```

This should start the booting up process. Currently, the default boot command is set to the one that is required by us, hence it directly proceeds to boot.

The default boot arguments are as follows:

```
bootcmd=tftpboot ; bootm

bootdelay=2

baudrate=38400

bootfile=uImage

stdin=serial

stdout=serial

stderr=serial

verify=n

bootargs=root=/dev/nfs ip=dhcp console=ttyAMA0 video=vc:1-2clcdfb: nfsroot=192.168.1.42:/local/home/ppal/nfs mem=128M
```

D. LOGGING INTO THE BOARD REMOTELY

1. Ssh into billie.ece.ncsu.edu.
2. The board has been supplied with an ip address of 192.168.1.149. Telnet into this machine. (Do not ssh.) Log in as root.
3. At the prompt type in the following:

```
~ #cd /

~ #chroot Debfs /bin/bash
```

4. The above should transfer control to the Debian file system. Beyond this point any Debian commands available can be used. Newer packages can be downloaded onto the system using an apt-get install <package-name> .All experiments are run in this environment.

E. CPU ACTIVITY MEASUREMENTS

CPU activity of the system, during the benchmark runs, is logged by the kernel. Profiling code inserted into the kernel, causes the load-balance function of CPU0 to log the

CPU activity state along with the time in jiffies[1]. The routine is called every 200ms, so that is the profiling frequency as well. This method has a drawback because CPU0, at the time of profiling, is in the state of transition between tasks. To get better CPU0 state values, a version of the kernel with CPU3 doing the profiling is built.

A program is written to read the dump of the kernel profiling messages, to analyze the values, and calculate the following: idle periods, percentage utilization and Total Run Time. Figure 4 shows an example of CPU activity plot after the kernel is modified.

[1] A jiffy is the duration of one tick of the system timer interrupt. It is not an absolute time interval unit, since its duration depends on the clock interrupt frequency of the particular hardware platform. For ARM, this value is 10ms.

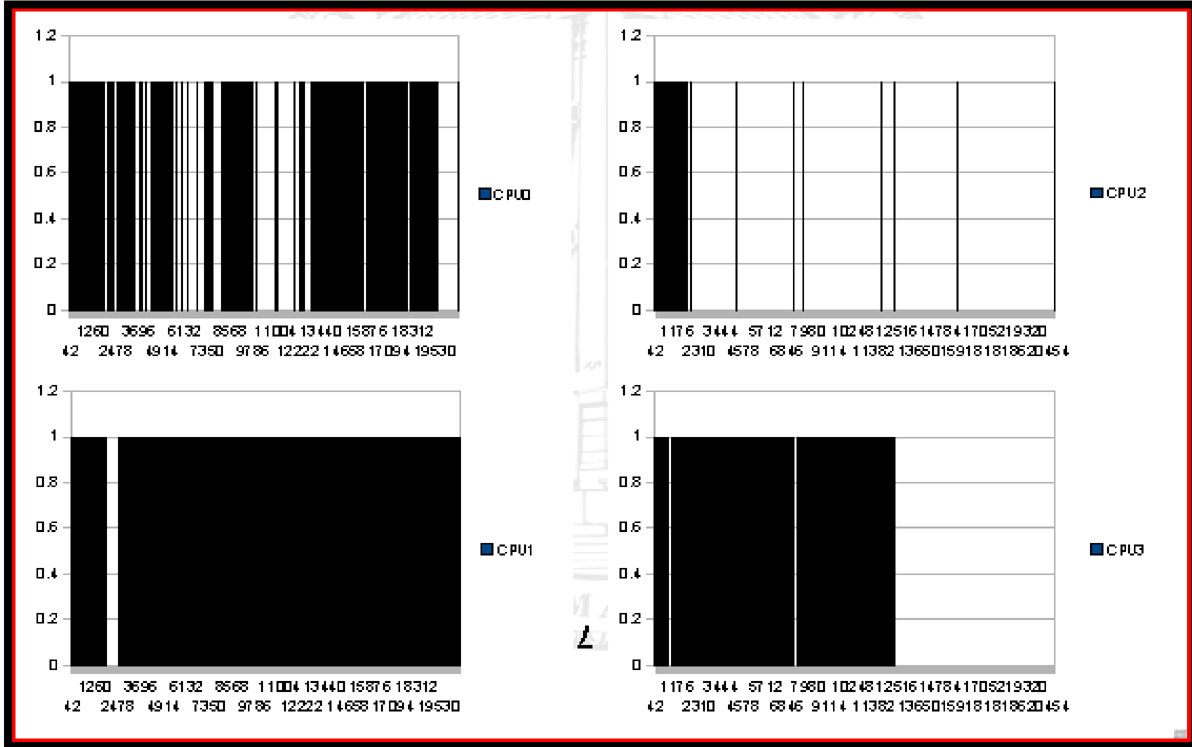

Figure 4: CPU activity graphs

II. BENCHMARKS

For running the experiments, two sets of benchmarks have been used. The first set is a subset of the MIBench telecommunication suite. These applications are representatives of common low-end applications that are run on most cellular systems. These applications are ones that utilize the CPU substantially, but run for a short time. The applications with a small description are tabulated in Table 1.

Table 1: Applications from the MIBench

Sl. No.	Name	Description
1	CRC32	This benchmark performs a 32-bit Cyclic Redundancy Check (CRC) on a file.
2	FFT	This benchmark performs a Fast Fourier Transform and its inverse transform on an array of data.
3	ADPCM	Adaptive Differential Pulse Code Modulation (ADPCM) is a variation of the standard Pulse Code Modulation (PCM).
4	GSM	The Global Standard for Mobile (GSM) communications is the standard for voice encoding/decoding.

The second set is a subset of the PARSEC benchmark suite. These applications represent the high-end applications that may be run on mobile phones. These applications are built in the single threaded mode, so as to properly represent legacy software that is not multi-threaded. These applications have almost 100% CPU utilization over a long period of time. Details of the applications used are listed in Table 2.

Table 2: Applications from the PARSEC Benchmark

Sl. No.	Name	Description
1	fluidanimate	This is an animation application used to represent different high-end game software that might be played on hand-held devices.
2	bodytrack	This is an application to track a 3D pose of humans. It is representative of computer vision/image processing applications used in cell phone cameras.
3	x264	This is a video encoding application that is part of MPEG-4 standard.

III. RESULTS

We shall first take a look at how the modified kernel, with a MaxRunQLength=1, fares with respect to the Original Linux kernel in the 2 benchmarks. The following show the lengths of idle times by the two different kernels under PARSEC benchmark. Tabulated below and plotted later are the mean values of the ten longest idle periods in decreasing order. The error bars indicate the standard deviation of these values.

Table 3: PARSEC Results

MEAN CPU3 IDLE TIME (JIFFIES)		**MEAN CPU2 IDLE TIME (JIFFIES)**		**MEAN CPU1 IDLE TIME (JIFFIES)**		**MEAN CPU0 IDLE TIME (JIFFIES)**	
Modified	**Original**	**Modified**	**Original**	**Modified**	**Original**	**Modified**	**Original**
22151	8113	21051	7567	10440	9897	7015	3356
5551	3971	6590	2970	4733	5162	3371	971
3283	1807	2408	1827	2420	4107	2183	566
460	959	1462	982	868	2063	1663	452
283	747	736	830	495	1662	1335	415
256	514	462	445	272	1119	873	414
201	402	428	317	258	923	715	358
166	310	397	220	189	637	668	324
129	249	288	184	174	520	564	303
116	226	256	146	151	401	396	280

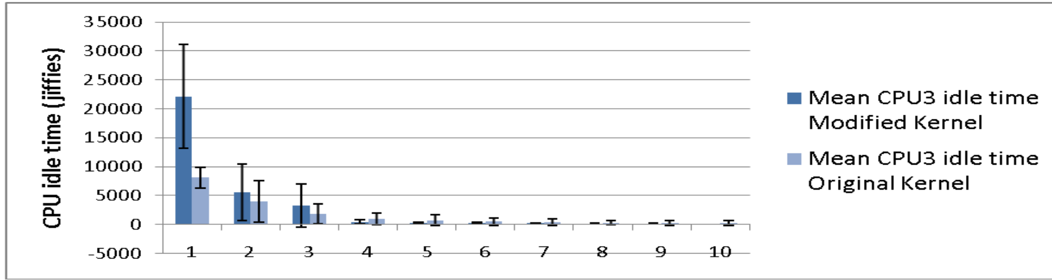

Figure 5: CPU3 idle times for PARSEC

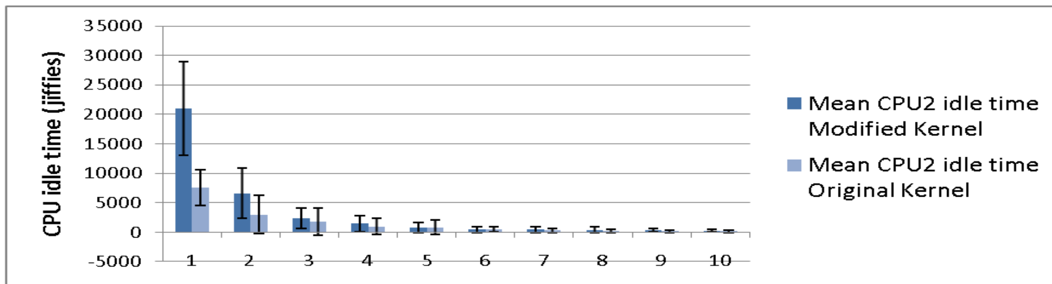

Figure 6: CPU2 idle times for PARSEC

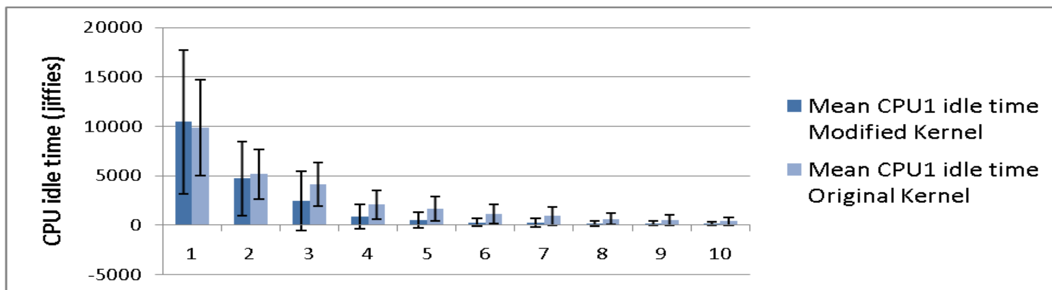

Figure 7: CPU1 idle times for PARSEC

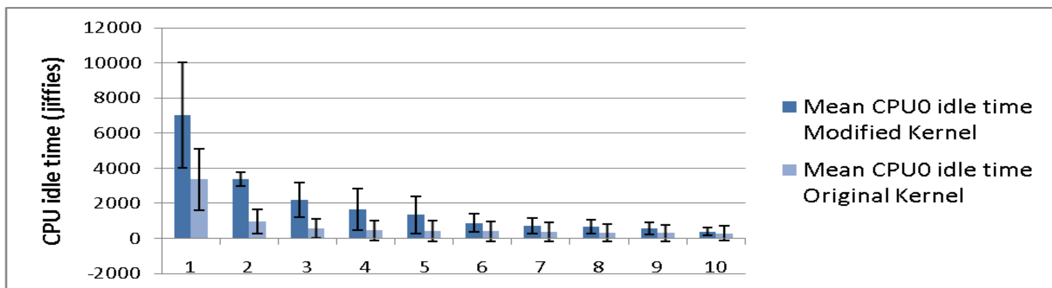

Figure 8: CPU0 idle times for PARSEC

As we see in the results of CPU3 idle times, there is a significant difference brought about by the consolidation, in the modified kernel.

We see that as the CPUs go lower in number, the idle time period advantage decreases, as is expected, for we intend heavily loading the CPUs that are numbered lower. Following are the results from the MIBench Benchmark.

Table 4: MIBench Results

MEAN CPU3 IDLE TIME (JIFFIES)		MEAN CPU2 IDLE TIME (JIFFIES)		MEAN CPU1 IDLE TIME (JIFFIES)		MEAN CPU0 IDLE TIME (JIFFIES)	
Modified	**Original**	**Modified**	**Original**	**Modified**	**Original**	**Modified**	**Original**
2517	1613	2521	1540	2391	1699	287	810
133	195	183	309	461	337	196	311
60	77	90	211	60	152	140	234
57	48	52	144	57	57	91	124
55	47	52	74	54	56	91	106
51	47	49	56	45	54	85	97
50	47	46	54	45	46	62	78
48	46	44	44	44	46	59	73
45	45	43	44	43	45	55	69
45	44	43	43	43	44	51	67

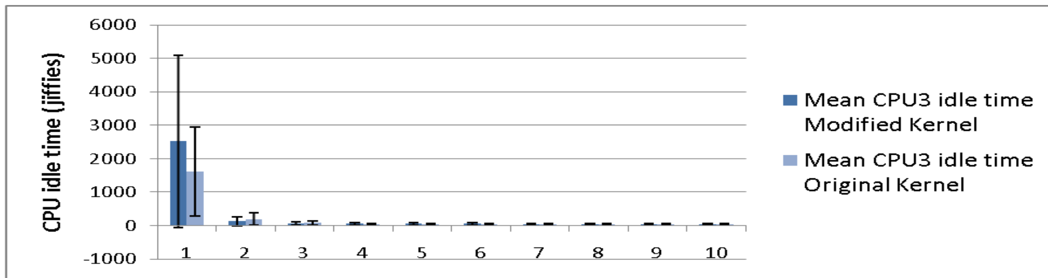

Figure 9: CPU3 idle times for MIBench

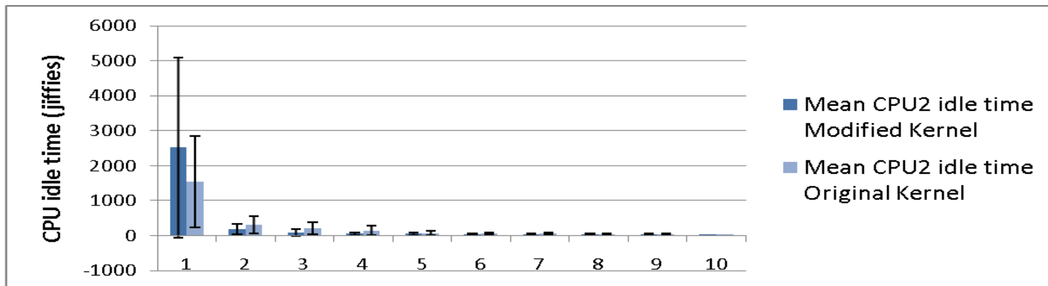

Figure 10: CPU2 idle times for MIBench

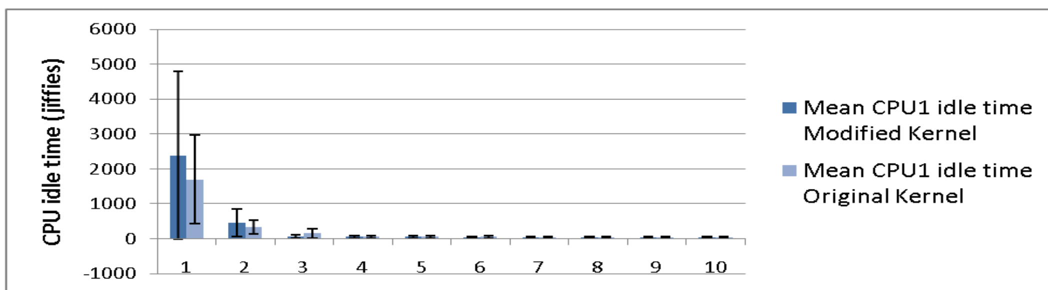

Figure 11: CPU1 idle times for MIBench

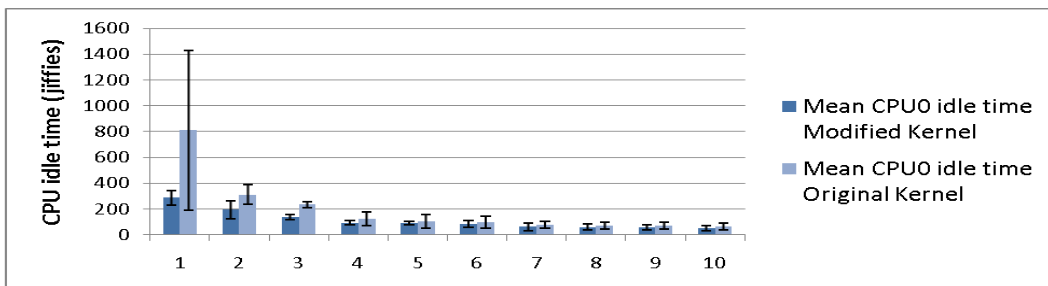

Figure 12: CPU0 idle times for MIBench

The CPU0 results of the MIBench runs indicate that the idle times of CPU0 have been drastically reduced. This is due to the preferential loading of CPU0 that is loaded to free up higher CPUs for better idle-time consolidation. Since CPU0 is never moved to sleep state, this doesn't adversely affect the potential energy savings. Effectively, we have moved these idle periods to higher CPUs, where they can be better exploited.

Further experiments are conducted to see how the dynamic scheme of varying the schedulable CPUs, compares to static schemes of fixing the number of CPUs beforehand. Following are the results of the PARSEC runs.

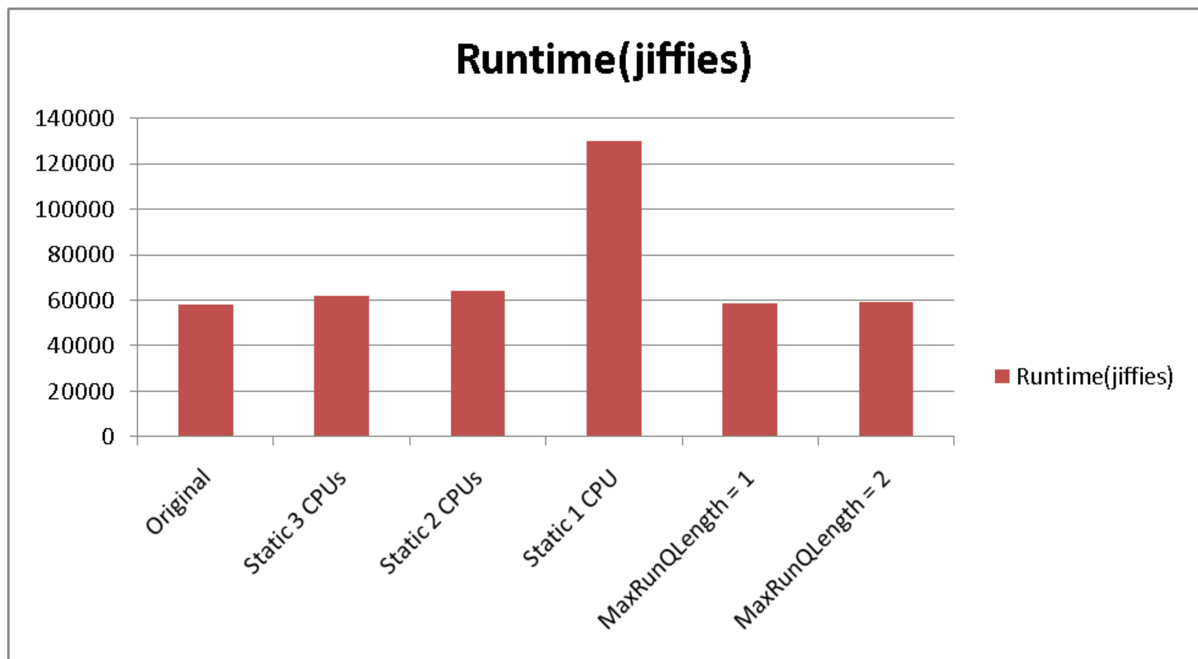

Figure 13: Runtime comparisons for PARSEC

For the Mibench runs, a similar trend is seen.

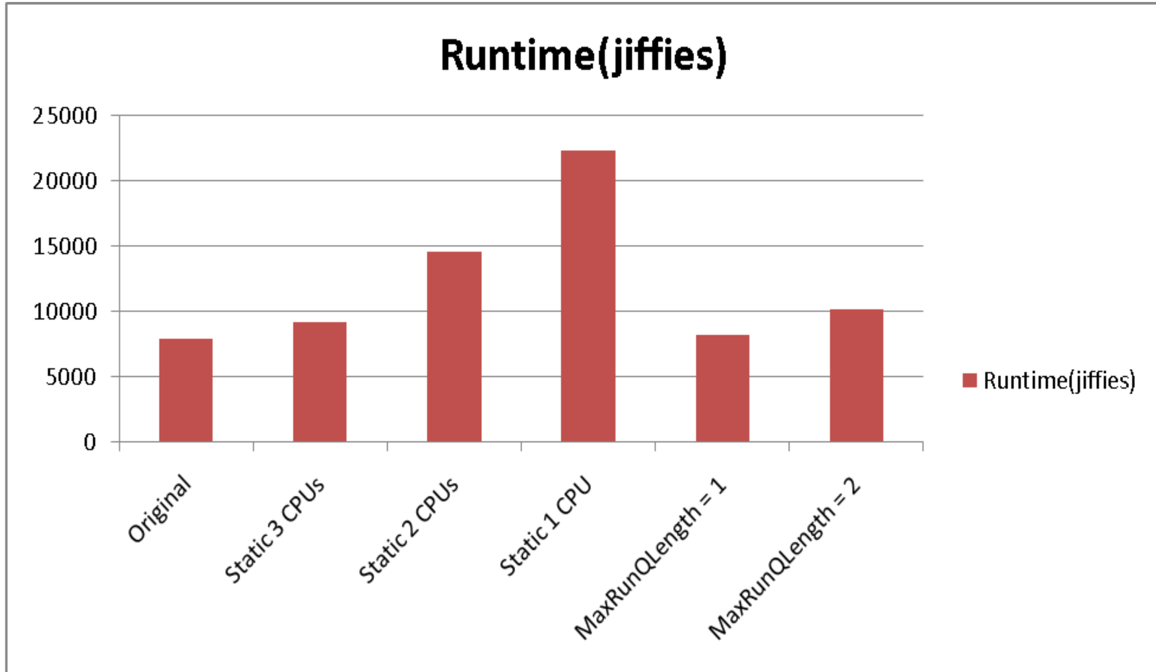

Figure 14: Runtime Comparisons for MIBench

Here we can see that our modified kernel compares well to the original kernel, with respect to runtime, and is better than some of the static schemes implemented. This can be explained by the dynamic nature of the system, which grows to accommodate new tasks.

The dynamic scheme(with proper setting of MaxRunQLength) closely matches the runtime performance of the best static scheme, while providing better opportunity for energy savings through consolidated idle times.

Chapter IV – Conclusion and Future Work

The goal of this work is to present a scheduling scheme for consolidation of idle periods for energy efficiency in Multicore Systems. This work is relevant for the high-end cell phones and other mobile platforms, which need to support complex applications now in addition to the regular voice services. Such an environment requires supporting a more non-deterministic workload, similar to general-purpose computers. Previous works in this area deal mostly with predetermined and pre-analyzed loads. In this work we strive to come up with efficient scheduling algorithms for more random workloads, to devise more reactive methods to schedule applications such that energy is conserved and battery life is longer. In addition, these methods would work well with older software as well, and no changes are required in the software to make them work with these schemes. The techniques used are extremely lightweight, as they use data readily available to the scheduler.

In this thesis, the scheduling algorithm introduces determinism compared to the original 2.6.28 Linux kernel, to ensure that CPUs that are numbered higher are allocated threads only when lower-numbered CPUs are unable to schedule the thread entering the system. The scheme essentially limits the number of processing elements that need to be constantly active. This allows the lightly-loaded elements to be idle for longer times, thus allowing them to switch to low-power idle modes. Based on the CPU activity data collected,

it is seen that the scheduling scheme lengthens the idle periods of the higher numbered CPUs (for example, CPU3, CPU2). On lower numbered CPUs, there isn't a great lengthening of idle times; rather, there is a shortening of idle times on CPU0. This can be explained by the scheduling rule which preferentially selects a lower numbered CPU to run a thread, thus causing inertia of activity or idleness in the CPUs.

The scheduling algorithm presented in this thesis works with a statically set value, MaxRunQLength that governs how many threads can be run on a processing element effectively. This value is the determining factor behind the system level at run time. Depending on the system, several factors account for the cost analysis to set the MaxRunQLength value statically as was discussed in Chapter II. In a typical system, there are various factors that might require this value to be set dynamically.

In workloads where the typical execution times of the applications are known, when a new thread is introduced into a system, we can use that value to analyze whether at the current state of the system; the new thread can be completed within the allowed time-frame. If yes, then MaxRunQLength is increased; otherwise a new processing element is brought up from its idle state. Another CPU state that is orthogonal to the CPU idle states is the CPU voltage-frequency processing state. Cost analysis can be done to see if it is more beneficial to switch the existing CPUs to a higher power processing state to accommodate a new task or to bring up a new processing element. This can be an interesting problem to look into.

The scheme presented in this work has been tested with different benchmarks representative of different loads that may be run on the system. These are suitable for consumer-relevant environments and can be used for idle time consolidation for energy efficiency on mobile platforms.

.

REFERENCES

1. V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer. “Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster.” IEEE Intl. Symp. on Parallel and Distributed Processing (IPDPS), April 2005, p. 4a.

2. N. Kappiah, V. W. Freeh, and D. K. Lowenthal. “Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs.” Supercomputing 2005, Nov. 2005, p. 33.

3. M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. “Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs.” Supercomputing 2006, Nov. 2006, p. 14.

4. Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. “Voltage and Frequency Control With Adaptive Reaction Time in Multiple-Clock-Domain Processors.” Intl. Symp. on High-Performance Computer Architecture (HPCA), Feb. 2005, pp. 178-189.

5. Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao, and E. H.-M. Sha. “Minimizing Energy via Loop Scheduling and DVS for Multi-Core Embedded Systems.” Intl. Conf. on Parallel and Distributed Systems, July 2005, pp. 2-6.

6. Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L. T. Yang, and E. H. -M. Sha. “Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems.” *IEEE Trans. on Circuits and Systems II: Express Briefs*, Vol. 54, No. 5, May 2007, pp. 445-449.

7. J. H. Anderson, J. M. Calandrino, and U. C. Devi. “Real-Time Scheduling on Multicore Platforms.” IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS), April 2006, pp. 179-190.

8. J. Luo and N. K. Jha. "Power-Efficient Scheduling for Heterogeneous Distributed Real-Time Embedded Systems." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 6, June 2007, pp. 1161-1170.

9. G. Qu. "Power Management of Multicore Multiple Voltage Embedded Systems by Task Scheduling." Intl. Conf. on Parallel Processing Workshops (ICPPW), Sept. 2007, p. 34.

10. D. Bautista, J. Sahuquillo, H. Hassan, W. Petit, and J. Duato. "A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications." IEEE Intl. Symp. on Parallel and Distributed Processing (IPDPS), April 2008.

11. D. Li, H.-C. Chang, H. K. Pyla, K. W. Cameron. "System-level, Thermal-aware, Fully-loaded Process Scheduling." IEEE Intl. Symp. on Parallel and Distributed Processing (IPDPS), April 2008.

12. E. Seo, J. Jeong, S. Park, and J. Lee. "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors." *IEEE Trans. on Parallel and Distributed Systems*, Vol. 19, No. 11, Nov. 2008, pp. 1540-1552.

13. R. Knauerhase, P. Brett, B. Hohlt, T. Li, S. Hahn. "Using OS Observations to Improve Performance in Multicore Systems." *IEEE Micro*, Vol. 28, No. 3, May-June 2008, pp. 54-66.

14. Suresh Siddha **"**Multi-core and Linux Kernel" http://software.intel.com/sites/oss/pdfs/mclinux.pdf

15. D. Zhu and H. Aydin. "Reliability-Aware Energy Management for Periodic Real-Time Tasks." IEEE Real Time and Embedded Technology and Applications Symp. (RTAS), April 2007, pp. 225-235.

16. D. Zhu, H. Aydin, and J.-J. Chen. "Optimistic Reliability Aware Energy Management for Real-Time Tasks with Probabilistic Execution Times." Real-Time Systems Symp. (RTSS), Nov. 2008, pp. 313-322.

17. Q. Shi, T. Chen, W. Hu, C. Huang. "Load Balance Scheduling Algorithm for CMP Architecture." Intl. Conf. on Electronic Computer Technology, Feb. 2009, pp. 396-400.

18. L. Benini, A. Bogliolo, G. D. Micheli, "A survey of design techniques for system-level dynamic power management." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 8 No. 3, June. 2000, pp. 299-316.

19. Y.-H. Lu, L. Benini, G. D. Micheli, "Low-power task scheduling for multiple devices." Proceedings of the eighth international workshop on Hardware/software codesign, May. 2000, pp. 39-43.

20. Y.-H. Lu, L. Benini and G.D. Micheli, "Power-aware operating systems for interactive systems", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 10 No. 2, Apr. 2002, pp. 119–134.

21. R. Golding, P. Bosch, and J. Wilkes, "Idleness is not sloth." *Proc.USENIX Winter Conf.*, New Orleans, LA, Jan. 1995, pp. 201–212.

22. Emulation Baseboard User Guide http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0303d/

23. ARM11MPcore Technical Reference Manual http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0360f