EVALUATING POWER MANAGEMENT CAPABILITIES OF LOW-POWER CLOUD PLATFORMS

Jens Smeds

Master of Science Thesis
Supervisor: Prof. Johan Lilus
Advisor: Dr. Sébastien Lafond
Embedded Systems Laboratory
Department of Information Technologies
Åbo Akademi University
January 2011
ABSTRACT

This thesis addresses the issues of the increasing energy consumption of data centers. Motivations for improving the energy efficiency of cloud infrastructure and methods to reduce the power dissipation of modern computing systems are presented. The idea of performing power management on a system level in many-core systems is discussed. A platform containing an ARM Cortex A-8 processor is studied to support the research on building a server for cloud computing based on low-power processors. The power dissipation and latencies of transitions between power states are measured to evaluate the power scalability of the platform. The presented measurements can be used for future development of a system level power management policy.

Keywords: Power Scaling, Energy Proportional, Cloud Computing, Power Management, BeagleBoard
I owe my deepest gratitude to the Embedded Systems Laboratory at Åbo Akademi University and in particular Professor Johan Lilius. I would not only like to thank you for granting me the opportunity to write this thesis, but also for all the guidance along the way. I am sincerely thankful to Dr. Sébastien Lafond for all the assistance during the past six months, especially for the feedback during the writing process. Special thanks also go to Professor Jan Westerholm for introducing me to the research at Åbo Akademi University in 2009.

Furthermore, I want to thank my co-workers and friends Olle Svanfeldt-Winter and Simon Holmbacka for helping me on a daily basis with different research related issues, and moreover, for making my workday more enjoyable. Finally, I would like to thank my good friend Johan Selänniemi, not only for all his work related input, but also, more importantly, for his continuous encouragement both on and off work.

Jens Smeds
Turku, January 2011
1 Introduction
   1.1 The Cloud Software Program ......................................... 2
   1.2 Purpose of this thesis .................................................. 2
   1.3 Thesis structure .......................................................... 3

2 Energy Efficient Cloud Infrastructure .................................. 4
   2.1 Energy efficiency .......................................................... 4
   2.2 Power dissipation in modern computer systems .................... 5
   2.3 Power management ......................................................... 7
      2.3.1 Advanced Configuration and Power Interface ................. 8
      2.3.2 Dynamic voltage and frequency scaling ....................... 10
      2.3.3 Low-power states ................................................... 11
   2.4 Energy in the cloud ...................................................... 14
   2.5 Clouds based on low-power nodes .................................... 17

3 Power Management in the Cloud ........................................... 19
   3.1 System level power management ...................................... 20
      3.1.1 Introduction ......................................................... 20
      3.1.2 Power scaling ....................................................... 20
      3.1.3 Power management granularity .................................. 21
      3.1.4 Task migration ..................................................... 22
# Table of Contents

1 Introduction ................................................................. 7
2 System level power management ....................................... 11
  2.1 Load balancing ......................................................... 11
  2.1.1 Task migration costs ...................................... 15
  2.1.2 Transition latencies ...................................... 16
  2.1.3 Cost modeling .............................................. 18
  2.2 Power management costs ................................................. 20
  2.2.1 Transition latencies ...................................... 21
  2.2.2 Task migration costs ...................................... 22
  2.3 A system level power management simulation framework ........ 24
  2.3.1 Approach ..................................................... 24
  2.3.2 Results ...................................................... 25
  2.4 Conclusion ............................................................. 26
3 Date-level power management ............................................ 28
  3.1 Load balancing ......................................................... 28
  3.1.1 Task migration costs ...................................... 30
  3.1.2 Transition latencies ...................................... 30
  3.2 Power management costs ................................................. 31
  3.2.1 Transition latencies ...................................... 31
  3.2.2 Task migration costs ...................................... 32
  3.3 A system level power management simulation framework ........ 33
  3.3.1 Approach ..................................................... 33
  3.3.2 Results ...................................................... 34
  3.4 Conclusion ............................................................. 35
4 BeagleBoard Power Dissipation ........................................... 36
  4.1 Introduction .......................................................... 36
  4.1.1 The BeagleBoard test system ................................ 36
  4.1.2 Power measurement on the BeagleBoard ...................... 38
  4.2 Power dissipation measurements ........................................ 38
  4.3 Analysis ............................................................. 39
  4.4 Conclusions of measurements ....................................... 40
5 BeagleBoard Transition Latency ............................................ 41
  5.1 Introduction .......................................................... 41
  5.2 Voltage and frequency scaling ...................................... 42
  5.2.1 Theory .......................................................... 43
  5.2.2 Implementation .............................................. 44
  5.2.3 Results ...................................................... 46
  5.2.4 Analysis ...................................................... 47
  5.3 Sleep transitions ...................................................... 47
  5.3.1 Theory .......................................................... 48
  5.3.2 Implementation .............................................. 49
  5.3.3 Results ...................................................... 50
  5.3.4 Analysis ...................................................... 51
  5.4 Conclusions of measurements ....................................... 52
6 Conclusions and Future Work ............................................. 53
  6.1 Conclusions .......................................................... 53
  6.2 Future work .......................................................... 54

Bibliography ................................................................. 57

Swedish Summary ............................................................ 61

A Code Used for Transition Latency Measurements ................... 70
  A.1 Code used to create an alternating signal .......................... 70
  A.2 Code used to change clock frequency .............................. 72
  A.3 Data extracting Python parser ....................................... 76
LIST OF FIGURES

2.1 ACPI states and transitions [16]. ..................................... 9
2.2 ACPI state grouping and power dissipation. ....................... 10
2.3 Workload enabling power management. ............................ 12
2.4 Microprocessor core idle states supported by Core i7 processors
[18]. .................................................................................. 13
2.5 Monthly costs of a data center [14]. ................................ 16
3.1 Example of load balancing with task migration. .................... 24
3.2 Example of task migration enabling sleep. ........................ 24
3.3 Power dissipation during sleep and wake-up transitions [6]. .. 27
3.4 Simulation results. A: incoming requests, B: QoS value, C:
number of cores in use, D: energy consumption. ................... 32
4.1 Block diagrams of the test system hardware. ...................... 36
4.2 BeagleBoard full load power dissipation versus frequency. .... 39
4.3 BeagleBoard idle power dissipation versus frequency. .......... 40
4.4 BeagleBoard power difference for idle and full load. .......... 41
5.1 Schematic of voltage and frequency scaling tests. ............... 46
5.2 Unavailability period caused by a change from OPP2 to OPP5
on a BeagleBoard visualized by PicoScope 6. ....................... 47
5.3 Measured OPP transitions. ............................................. 48
5.4 OPP transition latencies. .............................................. 50
5.5 Domain sleep and wake-up transitions for OMAP3530 [29]. . .. 52
5.6 Schematic of wake-up tests. .......................................... 53
5.7 Transition latency of a change from S3 to OPP6 on a BeagleBoard
visualized by PicoScope 6. ........................................ 54
CHAPTER
ONE

INTRODUCTION

Energy consumption has become an important issue for IT infrastructure. The growing market for large distributed computing platforms, so-called clouds, has put an increasing demand on servers and data centers. The consumption of electricity caused by servers, including cooling and other supporting infrastructure, in 2005 was calculated [20] to account for about 1.2 % of the total US electricity consumption. This number has been estimated to be doubled by the year 2010. The high consumption is not only a burden for the environment, but the cost for running servers is becoming a considerable factor from an economical standpoint. As a result, the development of more energy efficient solutions has become an even more popular research topic within the industry.

Recent research [3, 14, 21, 26] shows that building computing clusters with architecture based on low-power nodes instead of the classical architecture based on typical server nodes, has potential to be more energy efficient. The energy consumption of a typical server processor used in a modern data center is an order of magnitude higher than the energy consumption of a low-power processor [21, 32]. Another goal of using low-power processors is to drastically reduce the costs created by cooling infrastructure. Cluster
architecture based on low-power nodes introduces new possibilities for system level power management. The power and performance scaling abilities of an individual low-power node is thus an important research topic for the development of new cloud infrastructure.

1.1 The Cloud Software Program

The Cloud Software Program is a research program of Tivit plc. The purpose of the program is to aid the software industry in Finland and to improve the position of the Finnish software industry on a global scale [1]. With the combined efforts of companies, universities and research institutes, the program aims at forming a new agile ecosystem for Finnish software development [30].

The objective of the research done in the Embedded Systems Laboratory at the Department of Information Technologies at Åbo Akademi University is to explore the possibilities to develop a server, which can be used for cloud computing, based on low-cost low-power mobile processors. Central aspects of the research are energy efficiency and total cost of ownership (TCO) of cloud infrastructure.

1.2 Purpose of this thesis

This thesis is done to support the research on building an energy efficient cluster for cloud services. The purpose of this thesis is to analyze the power management features and energy consumption of a low-power embedded microprocessor. One of the goals is to establish parameters that affect the performance and energy consumption of a node in a server cluster. The established parameters can be used in future research to develop a power management policy that increases the energy efficiency of clouds. The power and performance scalability of a low-power microprocessor is analyzed by studying the relationship between power and clock frequency on a commonly
available low-power single-board computer. Furthermore, values for the established parameters are measured on the board.

### 1.3 Thesis structure

Chapter 2 begins by introducing the concept of *energy efficiency* and describing some common techniques to reduce energy consumption. Chapter 2 continues with an introduction to *cloud computing* and *energy consumption in the cloud* before discussing the new approach to use low-power nodes. *System level power management* and parameters that affect power management in many-core systems are presented in Chapter 3. We argue why these parameters must be considered when creating a power management policy for cloud infrastructure.

In Chapter 4 we present measurements of the power dissipation of a BeagleBoard. Values for some of the previously defined parameters are measured on this particular platform in Chapter 5. The thesis concludes with a summary of the results and suggestions for future work.
ENERGY EFFICIENT CLOUD INFRASTRUCTURE

2.1 Energy efficiency

The concepts energy and power, and their meanings in regard to computer systems, are first explained to form an understanding of energy efficiency and energy efficient computing. Energy, which is commonly presented as the ability to do work, exists in many forms (e.g. kinetic, potential, heat, electromagnetic and electric). Energy cannot vanish or cease to exist; it can only be transformed into other forms of energy. In computing systems, energy is needed to perform computations as well as to store and move data. The energy is supplied as electricity and, when used, it is transformed into heat. Power is the physical quantity that represents the amount of energy transformed into another form per time unit. Power describes the rate of energy consumption, i.e. how fast energy is used. This thesis uses the SI derived units Joules (J) and Watts (W) for energy and power, respectively.

Energy efficiency can, based on the definition of energy and power, be established as the amount of work that can be performed with a certain
amount of energy. Depending on the type of the performed tasks, the unit of measurements for energy efficiency varies. In this thesis, we use the concept energy efficiency in the context of computer systems without much detail to the actual units. Energy efficiency, as used in this thesis, rather refers to the more ambiguous definitions presented by Harizopoulos, Sahah, Meza and Ranganathan [15):

\[
\text{Energy Efficiency} = \frac{\text{Work Done}}{\text{Energy}} = \frac{\text{Work Done}}{\text{Power \times Time}} = \frac{\text{Performance}}{\text{Power}}
\]

2.2 Power dissipation in modern computer systems

Today’s computer systems consist of electronic circuits with millions of transistors. Every transistor consumes energy when powered. As technology develops, the number of transistors grows. This trend has followed Moore’s Law, according to which the number of transistors per silicon die area doubles approximately every two years, for decades [24]. The drastic growth of the number of transistors used in circuits that has occurred the last decades has resulted in a radical increase in the total amount of energy consumed by computer systems. To realize the origins of the energy consumption, we examine the CMOS circuits that modern computer systems consist of. The power dissipation of a CMOS circuit is analyzed to form an understanding of how the energy consumption can be reduced. The power dissipated in a CMOS circuit can be divided into two parts: dynamic power dissipation and static power dissipation.

The dynamic energy consumption consists of the power dissipated during a transition from logic ‘0’ to logic ‘1’ or vice versa. While switching from one logic state to another, the capacitive loads in the transistors are charged and discharged causing so called switching current. Further, there is a short period of time when the transistor is short-circuited. The short-circuiting causes a current through the transistor. The two currents, out of which the switching current generally is the dominant one, originate dynamic power dissipation [23]. As described by Weste and Eshragian [34], the dynamic power
dissipation, \( P_d \), of a CMOS circuit can be obtained from the formula

\[
P_d = C_l V_{dd} f
\]  

(2.1)

where \( C_l \) is the load capacitance, \( V_{dd} \) is the supply voltage and \( f \) is the transition frequency. Equation (2.1) shows that the dynamic power dissipation depends on three products. The load capacitance is completely determined by the design of the hardware and cannot be adjusted after the hardware has been manufactured. Both the voltage and the frequency are, however, generally adjustable in modern circuits. Regulation of voltage and frequency is used for dynamic power management (DPM). Such DPM methods are discussed later in this chapter.

Static power dissipation accounts for the rest of the power dissipated in CMOS circuits. The static power dissipation is continuously dissipated regardless of the state of the transistor or any state transitions. The static dissipation originates from leakage currents. Leakage currents are generally small flows of current that are not intended to, and ideally would not, exist in the circuit. In practice, leakage currents cannot be avoided completely.

As the technology used in the semiconductor manufacturing process shrinks, the amount of leakage current increases [22]. As a result, the static power dissipation has over time become an increasingly important factor of the total energy consumption in modern circuits. The total static power dissipation, \( P_s \), in a CMOS circuit is defined [34] as

\[
P_s = \sum_{1}^{n} \text{leakage current} \times \text{supply voltage}
\]  

(2.2)

where \( n \) is the number of transistors. The leakage current is determined by the hardware and how it is built, e.g., by the used manufacturing process technology. Equation (2.2) shows that the static power dissipation can only be controlled by adjusting the supply voltage, or by shutting down parts the circuit and thus reducing the number of powered transistors.
2.3 Power management

High power dissipation results in both high energy consumption and high temperatures. Minimizing the power dissipation, and thus avoiding overheating and keeping the energy consumption low, is important for modern computer systems. Power management is essential for the battery life and physical size of laptop computers and embedded systems, but also for server clusters in data centers. By studying the power dissipation of CMOS circuits, one can note that there are several factors that affect the power dissipation in modern computers. These factors should be managed in a manner that minimizes the power dissipation, while still being able to offer functionality and performance according to requirements set on the system. The goal is to maximize the energy efficiency (see definition in equation (2.1)) by minimizing the power.

Techniques to avoid unnecessary energy consumption have been developed [19]. Hardware features to reduce power dissipation have been controlled by BIOS-based power management [19]. To achieve more effective power management there is, however, a need of more comprehensive interaction between software and hardware. Software, e.g. the operating system, can make more intelligent decisions and control the power management features supplied by the hardware. Specifications and standards how power management can be implemented have been formed [16]. The purpose of these specifications is to make it possible to apply different power management techniques as effectively as possible.

For a typical server, the CPUs can contribute to over 50 % of the total server energy consumption [7]. In contrast to many other components, in which the power dissipation only can be controlled by turning the component on or off, CPUs offer more advanced abilities to dynamically manage the energy consumption. Two common power management methods are Dynamic Voltage and Frequency Scaling (DVFS), which enables power reduction on the cost of performance, and power management using low-power states, which reduces power on the cost of functionality. In this section we first give a short summary of a well established specification for Operating System-directed configuration and
2.3.1 Advanced Configuration and Power Interface

The Advanced Configuration and Power Interface (ACPI) [16] is a collective specification that forms a standard of interfaces for power management. It has been developed by five companies within the industry: HP, Intel, Microsoft, Pheonix and Toshiba. The ACPI acts as an interface between hardware and operating system. The purpose of the specification forming this interface is to create a standard for how the interaction between the operating system and the hardware of a computer system is implemented.

The interface enables a common way to perform configuration and power management from the operating systems and is fundamental for OSPM [18]. Compared to the older BIOS-based power management model, OSPM offers a better view of the system as a whole [19]. The ACPI is purely a specification and only acts as an interface. Hence, the ACPI can be applied on all different types of computer systems, including desktop, mobile and server machines.

In a system where the ACPI has been implemented, the operating system is usually responsible for conserving energy by making power management decisions. These decisions determine how the power management methods specified by the ACPI are used to minimize energy consumption. The ACPI also specifies a namespace to access the hardware. The namespace includes information about the type of the hardware as well as what power management features are available on the hardware. By including this information, the namespace gives the operating system the ability to easily handle power management decisions.

The ACPI specification includes definitions of different power states, not only for the system as a whole, but also for individual hardware devices. These states and the transitions between them are illustrated in Figure 2.1. The system states consist of the global system states $G0$ to $G3$, the sleeping states $S1$ to $S4$ and a state called Legacy. The $G3$ state represents the system when no
power is supplied, i.e. the system is completely powered down. The G2 state (also called S5 or Soft Off) is a state where no user mode or system mode code is executed and the system must be restarted to return to the normal working state. The global sleep state, G1, is divided into four different sleep states. The sleep states are defined based on how much of the system is turned off. These states range from the most energy consuming state S1, to the deepest sleep state, S4, where the largest part of the system is disabled. The working state is called G0 (also S0). In the Legacy state, the power management decisions are not made by an operating system but by hardware or firmware.

Power states for pieces of hardware include the device power states D0 to D3 and the processor power states C0 to Cn. When the system is in the working state, the states of devices, such as hard disk drives, modems, etc., are described by the device power states ranging from D0, which is the fully-on state, to D3, where the power of the device is removed. The C0 to Cn states represent corresponding states for the CPU. Further, there are a varying number of performance states available for a processor or other device in C0 or D0. These performance states, P0 to Pn, represent different performance
and energy consumption levels of the device or processor. An overview of the state grouping and their individual order relative to the power dissipation is displayed in Figure 2.2.

Figure 2.2: ACPI state grouping and power dissipation.

Today, the ACPI is widely supported by operating systems, e.g. Linux and modern versions of Windows, as well as hardware manufacturers. The power states of the ACPI enable a common way to apply power management methods such as DVFS and sleep states for both the whole system and individual devices.

2.3.2 Dynamic voltage and frequency scaling

Studying the origins of power dissipation in modern CMOS circuits by analyzing equations (2.1) and (2.2), one can note that the power dissipation, both the dynamic and static part, is dependent on the voltage. Furthermore, the dynamic dissipation is also affected by the clock frequency. As a result, the energy consumed by a modern CPU can be reduced by running it using a lower voltage level and on a slower clock frequency.
There is dependence between the voltage and the clock frequency of a processor [9]. Switching the state of a transistor requires charging and discharging of the capacitance in the transistor. The charging and discharging is, however, not instant. How fast the logic state can be switched is therefore determined by the time required to charge or discharge the capacitive load. As the charge and discharge time depends on the current, which is determined by the voltage, the maximum frequency is dependent on the applied voltage. A higher voltage charges and discharges the capacitance faster and thereby allows a higher maximum frequency. The clock frequency and supply voltage should thus be adjusted simultaneously according to the demands between them. The voltage should, from an energy efficiency perspective, always be as low as possible for any chosen clock frequency.

The workload and usage of a processor is not always constant but varies over time. An effective way to reduce energy consumption is to regulate the voltage and clock frequency according to needs of the processor while the processor is running [11]. Scaling the voltage and frequency has become a commonly used technique for DPM. Figure 2.3 shows an example of a load that presents opportunities to save energy. When the load drops during periods $P_1$ and $P_2$, the processor is not required to run at full capacity. The frequency and voltage can thus be reduced to lower the energy consumption. This technique is called Dynamic Voltage and Frequency Scaling. The ACPI specification supports DVFS by defining the $Px$ states. A $Px$ state can for example define the clock frequency and voltage of the processor. The power dissipation can thus be reduced by dynamically switching between different $Px$ states.

### 2.3.3 Low-power states

Another effective way to reduce power dissipation is to use low-power states [6]. By using DVFS, one can reduce the energy consumption on the expense of performance while still having the whole system running. There are, however, time periods when certain parts of a computer system are not used and do not need to be powered. By using low-power states the dynamic power dissipation is removed if the clocks are cut. Moreover, the rest of the power dissipation,
i.e. the static dissipation, of that part of the system is eliminated if the supplied power is removed. Thus, more energy can be saved by turning off parts of the system compared to only reducing the operating voltage and frequency.

For the example in Figure 2.3, the CPU is not needed during period $P_2$. Powering down the processor during this period gives the ability to further reduce the power dissipation. The usage of such low-powered states, also referred to as sleep states, is commonly used for power management. Generally, the latency for returning to the working state from a sleep state is, however, higher than for only switching between performance states. It is important to predict the time period that the system can remain in a sleep state to estimate the outcome of using that state. For example, the power management needs to decide if the period $P_2$ is long enough to use a sleep state for the load in Figure 2.3. Latencies of transitions and the consequences of these latencies are described in greater detail in Chapter 3.

The ACPI specification includes the $Sx$ states as sleep states for the system as a whole. No user mode code is executed in a system sleep state. The system can, however, resume to the working state and continue work without being rebooted. The $Dx$ and $Cx$ states in the ACPI specification represent low-power states for devices and processors. In these states, parts of the device or processor are disabled to eliminate unnecessary power dissipation.
Examples of techniques used in the hardware implementation of Cx states, i.e. to reduce the power dissipation of a processor, are: clock gating, turning off phase-locked-loops (PLLS) and flushing caches [18]. Clock gating can be implemented in a circuit by making a logical ‘AND’ operation on the clock signal and a control signal. The system is thus able to disable the clock signal by turning the control signal to a logical ‘0’. By disabling the clock signal, the dynamic power dissipation can be eliminated. Figure 2.4 shows the differences between four core-specific Cx states supported by Intel Core i7 microprocessors. The figure shows not only the state of the clock, PLL and caches, but also the differences in both the idle power dissipation and the wake up latencies.

Figure 2.4: Microprocessor core idle states supported by Core i7 processors [18].

Techniques to reduce the dynamic power dissipation on ARM Cortex A-8 processors include “function gating” (e.g. NEON, ETM and integer core gating) and “state element gating” (e.g. local clock gating) [4]. The idea behind these gating techniques is similar to the idea behind clock gating, i.e. to remove dynamic dissipation by removing functionality. The implementation of leakage current management, i.e. methods to reduce the static power dissipation, on the ARM Cortex A-8 is based on power domains. The power domains allow different parts of the processor to be completely powered down. The static power dissipation of that domain is thus eliminated.
An example of a system sleep (Sx) state is the widely used S3 state (also known as Suspend to RAM) [16]. Power is no longer supplied for a microprocessor in this system state. The context of not only the processor, but also the cache and chipset is lost. The context of the main memory is retained. The processor context and the L2 cache configuration context are restored when returning to an active state.

2.4 Energy in the cloud

Cloud computing is a term described [35] as the approach to provide computing as a service. Cloud computing is based on the idea that hardware, software and information can be globally provided as a service on an on-demand basis over the Internet. The Platform as a Service (PaaS) and Software as a Service (SaaS) models offer the users an alternative to their own computing systems and applications [10]. By having a shared supply of services accessible over the Internet, multiple clients can be offered the same services. The need for every individual client to have own personal hardware and software is thereby removed.

In practice, cloud computing is realized by computing clusters and data centers that offer these services. The cloud is defined [5] as an umbrella term covering the hardware as well as the software provided by such data centers. This thesis is, however, mainly centered on the hardware and the interpretation of the cloud as a platform. The research at the Embedded Systems Laboratory at Åbo Akademi University is concentrated on the cloud infrastructure, in particular the server hardware, rather than the applications that run in the cloud.

To provide the capacity needed to meet the demands of the services in the cloud, and moreover, to provide redundancy to ensure reliability, extremely large-scale data centers, so called server farms, have been built around the world [5, 10]. Google was, for example, already in 2007 estimated [33] to have a cluster containing half a million servers located at around a dozen different locations.
When computing systems grow to this extent, the energy consumption becomes an important factor. The electricity used by the servers is transformed into heat, which is dissipated from the circuits. The large amount of dissipated heat creates a need for cooling and other supporting infrastructure. The total energy consumption of a data center does therefore not only consist of the energy consumed by servers to perform operations, but also of the energy consumed by supporting infrastructure.

Energy consumption of this magnitude is not only to be considered because of environmental aspects, but it also results in other considerable problems. These problems include:

- Electricity costs.
- Cooling and heat transfer.
- Space requirements.

The electricity consumed by servers creates a cost that can be even higher than the cost of the server hardware itself [8]. Consequently, the energy consumption is a major part of the total cost of ownership of data centers. The energy consumption and cost of typical servers for Internet-scale services are discussed by Hamilton [14], who also state that the majority of the infrastructure cost of a data center is related to power. Figure 2.5 shows how the monthly costs are divided in a hypothetical example of a typical data center presented and discussed by Hamilton. The total amount of cost related to power distribution and cooling stands for approximately 82% of the cost of all infrastructure. This is compared to the cost of the actual facilities, which is estimated to correspond for 12-15% of the total infrastructure cost.

Further problems caused by high energy consumption include the limited ability to build and extend data centers. The extensive cooling demands limit the density of servers. Servers cannot be constructed to be very compact due to the risk of overheating. The space requirements reduce the possibilities to construct larger data centers [15]. The cooling and power demands of vast server farms need to be considered already in the planning stage when
building such a farm. Today, major factors considered when deciding the locations of new server farms include available power sources (e.g. existing power plants) and natural cooling resources (e.g. cold climate and water) [33].

To reduce the energy consumption of a cloud, the servers should be managed according to the demand. As the demand of the service shrinks, the energy consumption should shrink. To achieve minimal energy consumption, the system must be fully energy proportional. Energy proportionality implies that the energy efficiency is constant for all levels of utilization [7]. As a result, the energy consumption would, for example, be zero when the whole cloud is idling. Complete energy proportionality cannot be achieved in practice due to leakage currents and other overheads. Energy proportionality is, nevertheless, a goal for power management of all kinds of computing systems, including clouds.

When considering the load fluctuations created by the varying demand of the services, energy proportionality becomes an essential factor for the energy consumption of clouds. The workload and demand of resources for a cloud service can vary in a drastic manner [21]. Peaks of up to 19 times the average demand have been observed [31]. Hardware with the ability to meet the demand of such peaks will have a very low usage rate during periods of less demand. Thus, a large part of cloud hardware is often idling. As stated by
Barroso and Hölzle [7], a modern server spends the majority of its running time at a low utilization level even though a low utilization level typically offers the lowest energy efficiency. The need of power management with the ability to fully exploit not only voltage and frequency scaling, but also low-power states, on a system level is thus very important when trying to reduce the energy consumption of cloud infrastructures. Such power management will be discussed in Chapter 3.

2.5 Clouds based on low-power nodes

A new approach to building computer clusters offering cloud services is addressing the problems created by the persistently increasing energy consumption. Processors for embedded systems such as mobile phones are designed for low power dissipation and high energy efficiency. The energy consumption has been emphasized due to hardware constraints and other characteristics (e.g. battery life) of the systems that the processors are designed for. So called mobile processors also have a small physical size and generally have no need for any active cooling. Building a cluster based on such low-power processors aims at having higher energy efficiency as well as lowering the cooling requirements and thus also reducing the individual physical size of server cards.

The typical contemporary server design is based on processors built for maximum performance rather than minimum energy consumption. The new approach suggests to, instead of designing the system around the typical high performance cores, use enough low-power processor cores to offer corresponding total performance. The idea behind this approach is to optimize the energy efficiency of the cluster by using mobile processors specifically built to minimize power dissipation. Another potential profit of using low-power cores is that modern mobile processors are cheaper than modern, more complex, server grade processors. As Figure 2.5 shows, the cost of the initial hardware investment is a major part of a data center’s total cost of ownership. The low price of mobile processors helps retaining the price of the server hardware.
Reliability is a central feature in traditional server design; servers are designed to be able to run for several years without failing. The continuous development of new technology enables more power efficient hardware. According to Hamilton [2], the development often results in servers being replaced by newer, more efficient, hardware within three to five years. The extra effort and cost invested in developing and manufacturing high quality reliable hardware does thus usually not actually provide anything. This results in money being wasted on quality that in the end is not utilized.

To achieve maximal energy efficiency, the servers should not be designed to last longer than needed. The estimated failure time should be equal to the estimated time of replacement. Mobile processors are not developed based on server design criteria. A server design implementing mobile processors cannot offer as low failure ratio and long life time as a traditional server design. As embedded components fail easier there is a need for redundant hardware to provide equal reliability. The overhead from the unnecessary quality can, however, be minimized using this design choice.

Another key-issue for a cluster architecture based on slower, but more energy efficient, processor cores is system level power management. The performance of one low-power node is lower than a server grade node. A cluster of mobile processors must thus consist of a significantly larger amount of nodes to offer a performance level that corresponds to the performance of a cluster of typical server grade processors. The increased amount of cores creates an opportunity to closer match the performance to the load variations, compared to a cluster consisting of traditional server-grade processors. The power management needs to be considered carefully as the architecture of the system changes and the amount of cores increases.
CHAPTER

THREE

POWER MANAGEMENT IN THE CLOUD

The extensive costs and problems caused by the energy consumption of cloud infrastructure have made power management an important research topic. Questions that need to be studied are how power management should be applied in cloud servers and what different factors affect the power management. The goal of this chapter is to form an understanding of what these questions implicate and how they can be answered.

This chapter begins with a brief presentation of system level power management. The idea and purpose of this approach to power management in a many-core system is explained in Section 3.1. A short description of what the concepts granularity, task migration and load balancing mean, in the context of system level power management, follows. Section 3.2 discusses different costs that affect power management. An introduction of how these costs can be modeled is also included in Section 3.2. A framework developed to simulate system level power management in a many-core system is presented in Section 3.3. The chapter is completed with a short conclusion regarding the important aspects of power management in a cloud.
3.1 System level power management

3.1.1 Introduction

System level power management is based on the idea to apply power management techniques according to the requirements of the computing system as a whole. The purpose of performing power management on a system level is to scale the power dissipation and performance of the nodes so that they match the needs of the whole system. For a cloud, system level power management is thus more suitable and more efficient than implementing power management individually in each core or each processor.

3.1.2 Power scaling

When power management in a many-core system is done on a system level, the power state of each node is managed according to a policy that defines the requirements of the whole system. These individual power states determine the power dissipation and the operating level, i.e. the capacity to execute tasks, of each core in the system. In a system utilizing the power management methods introduced in Chapter 2, the power states of a node could for example include: one state for full voltage and frequency, another state for low voltage and frequency, and one low-power sleep state. The performance and power dissipation of the node is determined by the power state the node is operating at. For example, full voltage and frequency provides best performance but has the highest power dissipation, while switching nodes in a sleep state reduces both the overall performance and the power dissipation.

The system level power management controls the power states of all nodes and switches between power states dynamically. The power management policy decides in which power state each node is at any given point of time based on the load of the system. The policy thereby controls the performance and energy consumption of the system as a whole. If the policy aggressively turns nodes into sleep states or low voltage and frequency states, the system will dissipate less power, but will typically not be able to respond to an increased
demand as fast as a system where the nodes more easily are left running at power states with higher voltage and frequency.

3.1.3 Power management granularity

The granularity of power management represents how many different power and performance levels are used in a system. A system with fine granularity is able to closer match the actual power and performance to the wanted power and performance than a system with a rough granularity. Finer granularity results in better power scaling capabilities and ability to achieve better energy proportionality. A system with fine granularity is thus able to achieve higher energy efficiency than the same system with rougher granularity.

Consider a system consisting of eight nodes where each node has three power states. The total amount of power state combinations for this system is $3^8 = 6561$. In a homogeneous system where all nodes are identical, the nodes are, however, not always considered as unique. The nodes not being unique means that the importance is not in the order of the nodes, i.e. in which node is in which state, but rather in the combination of power states. For example, if the system has seven nodes in a sleep state, it does not matter which one of the nodes is not sleeping. The theoretical amount of combinations, $C$, of power states, not considering the order, can be calculated as [13]:

$$C = \binom{\text{NumberOfStates} + \text{NumberOfNodes} - 1}{\text{NumberOfNodes}}$$ (3.1)

Inserting the numbers of the system in our example, we get:

$$C = \binom{3 + 8 - 1}{8} = 45$$ (3.2)

The power management of the system in our example thus has from 6561 to 45 combinations of power states available to chose between. The combinations are used to match the systems performance with the load and to thereby form an as energy proportional system as possible. How many of these combinations are used in actuality depends on how the power management
is implemented. For example, a very poor, but simple, policy could only use two states: one working state and one when the whole system is idling. The number of used power state combinations defines the granularity of the implemented power management.

Equation (3.2) shows that the number of combinations is determined by the number of states and the number of nodes. A system with more nodes gives a finer granularity if power management methods are applied on a per-node basis. More nodes result in more possibilities when the power state of each node is set individually. Consider, for example, that one high performance node provides the same performance as four low-power nodes. With this assumption, a system of 8 high performance nodes would correspond to a system of 32 low-power nodes. The 32 low-power nodes would, assuming that the two systems have the same power states, provide more combinations of states and thus provide an opportunity for system level power management with a finer granularity than the 8 high performance nodes.

The high variation in load produced by cloud services profits the usage of a cluster design based on a large amount of nodes. A cluster consisting of many low-power nodes allows for different power scaling capabilities and better energy proportionality as it offers a finer granularity. The importance of system level power management thus becomes one of the key-issues for the approach to develop cloud infrastructure based on low-power nodes.

3.1.4 Task migration

There are two design choices for the memory architecture in a multi-core system. One is to use a shared memory to run programs for all processor cores. The other design is to run programs from the own local memory of each core. Compared to the shared memory design, the design using the local memory allows the cores to access memory in parallel. The support for parallel memory accesses makes a core less dependent on memory accesses of the other cores. Parallel memory accesses increases the scalability of the platform. The architectural approach to use the local memory is typically implemented in embedded multi-core platforms.
Running programs from local memory introduces a need to move running task between processor cores. To move executing tasks is called task migration. The methodology is to stop executing a task on one core and move it to another. The migration needs to be done so that the execution on the new core can continue exactly from where the execution was stopped before the task was moved. More detailed descriptions of the task migration process and alternatives to implement task migration in multi-core real-time systems is provided in Holmbacka’s Master’s Thesis [17]. Task migration is needed for load balancing and is thus an important part of both energy efficiency and performance optimization.

### 3.1.5 Load balancing

Load balancing is a central part of system level power management. How well power management methods can be applied depends on how the load is distributed among the cores in a multi-core system. The goal is to distribute the load so that the system consumes as little energy as possible, while still satisfying any other constraints (e.g. performance constraints). Load balancing can be done by choosing on which core each task should be executed before actually assigning the task to a core. The load can, however, also be balanced after the task has been assigned to a core, even after the execution of the task has started, by migrating the task to a new core.

Figure 3.1 shows an example of load balancing using task migration. Core1 is loaded with more tasks than it has capacity to handle. The other core, Core2, is not used to its full capacity. The utilization of the processor cores is maximized as the load is balanced by migrating tasks from Core1 to Core2.

Dynamically distributing the load in a manner that maximizes the utilization can improve both the overall performance and the energy efficiency. The performance is increased when the tasks can be processed in parallel. Low utilization levels result in low energy efficiency due to the disproportionate energy consumption of cores. Maximizing the utilization thereby increases the energy efficiency.
Load balancing and task migration can also be used in power management as an aid to use sleep states. Migrating tasks away from cores with low load and thereby making them free creates an opportunity to put cores to sleep. Figure 3.2 shows an example of how task migration can enable the usage of sleep states. All tasks of Core2 are moved to Core1. Since there are no tasks to process for Core2, it can be put in a sleep state.

There are different approaches for implementing load balancing. Load balancing can, for example, be centralized so that one node is responsible for distributing the load. Another way to implement load balancing is to distribute the responsibility. Load balancing with distributed responsibility implies that every node has to be able to decide what kind of load they should have. This approach introduces questions about how the nodes should communicate to find a node that is able to execute the task. Scalability is an important issue that must be evaluated when deciding on how to implement the load balancing in a many-core system.
3.2 Power management costs

There are several elements that affect the efficiency of power management. These elements should be taken into consideration by the software that makes the power management decisions. As described in Section 3.1 and displayed by equation (3.2), there is a large variety of possible combinations of power states for a processor cluster.

The purpose of the power management software is to choose the optimal combination of power states, i.e. the states that minimizes the energy consumption of the system. The decisions to change the power states of the system are based on evaluations of each available state. These evaluations consist of estimations of the costs of each state. The estimated costs are compared and the power management chooses the state that has the lowest total cost.

The total cost of a state consists of the cost of the transition from the current state to the new state and the cost of running the system in the new state. The cost of a state is a trade-off between energy consumption and performance. The operating cost of a state is thus compared to the performance level of that state. The two following costs are included in the total cost of a state transition:

- Cost originating from the transition latencies.
- Cost of any task migration done due to the state change.

This section continues with a comprehensive analysis of the latencies of power state transitions. A more brief discussion about the costs of task migration follows when both the reasons and the impacts of transition latencies have been explained. To complete this section, we introduce the idea to create a formal model that represents costs and give an overview of an example of how cost modeling for power management in many-core operating systems can be implemented.
3.2.1 Transition latencies

The switch from one power state to another is not instantaneous. The time required to switch a power state depends on both hardware and software. A part of the latency comes from the time required by the hardware to make the actual change of the power state, e.g. to power off a component or to stabilize new voltages and frequencies. The rest of the latency is caused by the all the operations, executed by the software part of system, that are needed to be able to perform the change. For example, transitions to and from deep sleep states that power down memories typically require time consuming data transfers so that no data is lost when the memories are not powered.

The total latency of a state transition is the time span from when the decision to make the change is made, to when the system is stable in the new state. If the target of the transition is a working state, the time required by the transition ends when the system is able to continue processing operations that are not related to the state transition. The latency varies depending on the states that the system is switching between. Generally, a switch between two adjacent performance states is faster than a switch between a sleep state and an active state.

Latencies caused by power state transitions contribute to the total cost that needs to be estimated when evaluating power management decisions. From an energy efficiency perspective, the latencies waste energy. The system dissipates power during the state change, i.e. the transition consumes energy. A study [6] shows that the power dissipation of a power state transition is not constant, and that the power dissipation in general is larger than the idle power dissipation since the switch includes operations that consume energy.

The switch to and from a sleep state is displayed in Figure 3.3. The figure illustrates both the existence of the transition latency ($t_{ss}$ to $t_s$ and $t_{sr}$ to $t_r$), as well as the energy consumption during the transitions ($w_s$ and $w_r$). The figure also shows that the power during the transitions ($t_{ss}$ to $t_s$ and $t_{sr}$ to $t_r$) is higher than the idle power $p_i$. From the figure, one can note that the work and latency of switching the power state clearly adds an extra energy cost.
Additionally, from a performance perspective, the state transition latencies affect the response time of the system. For a system consisting of a single processor core, it is clear that the time needed by the system to respond to a suddenly increased demand is delayed by the latency of any required state switch. For a single-core system, an increased demand requires the core to offer higher performance, which can result in a state transition. The time to respond to the increased demand will in this case be extended by the time needed for the system to switch to a state that offers the needed performance. The same applies for many-core systems, e.g. a cloud. The aggressiveness of the system level power management, i.e. how easily the power management switches to a new state, is thus to be decided not only based on the wanted energy efficiency, but also based on the wanted response time.

### 3.2.2 Task migration costs

Similarly to power state transitions, task migration is not instantaneous. As described by Holmbacka [17], task migration involves multiple stages and core-to-core communication. To be able to continue executing a migrated task from where the execution was halted, the execution of the task cannot be halted at any arbitrary point.
The required communication, as well as having to halt and resume the execution of the task, result in extra work and a delay in the execution time of the task. Furthermore, the task migration initiative, i.e. the decision to migrate a task and to start the migration process, adds to the overhead. The overhead does not only depend on the hardware, e.g. how fast the communication channels are, but also on how the task migration is implemented.

Task migration can be a part of a power management decision in a multi-core system. As an example of how a power state transition can include task migration, consider a core executing a task. The task is migrated if the core is set to a sleep state and if there is another core with the ability to execute more tasks. The execution can continue on the new core while the other core is set to sleep. To form an as accurate cost estimation as possible for the evaluation of the power state transition, the power management must, however, consider the cost that the required task migration contributes. The energy cost of the task migration is the total amount of energy spent due to all extra work that the task migration introduces.

### 3.2.3 Cost modeling

The cost for each available state can be modeled to accurately evaluate power states and transitions between power states. Cost modeling supports the decision-making in power management. Cost models enable intelligent decision-making regarding which power states to use. By basing the decisions on a formal model, from which the costs of all possible transitions can be calculated, the power management can minimize the energy consumption. In this kind of advanced power management, the costs for every available power state is estimated using the model. The power management can thus, by estimating all possible costs, select the state with the lowest cost and thereby minimize the energy consumption.

Simone [25] presents in his Master’s Thesis a formal cost model for power management in a multi-core multiprocessor system (i.e. a system containing several multi-core processors). The presented model is based on various assumptions (e.g. regarding the power states, task migration and state
transitions). The model and how it is formed does, nevertheless, demonstrate important aspects of how power management can be based on a formal cost model.

To keep the model simple, putting processor cores to sleep and waking them up are the only power management features utilized in the model. Simone’s model does, however, also differentiate the state when all cores of a processor are in a sleep state. A processor that has all its cores in a sleep state is defined to be in a state called deep sleep. When the whole processor is set to deep sleep, the overhead, which is caused by parts of the processor that do not belong to any specific core, can be eliminated. Hence, the deep sleep state provides more energy savings than just having all cores of a processor in the normal sleep state.

Simone’s model simply adds the energy costs of running a state with the energy cost of any required transition to that state. A value that represents the amount of work that can be done in the state is subtracted from the sum. The cost of running a state is modeled as the energy consumed by the processors in that state. This way, the energy consumption of the whole system is calculated. The transition cost is modeled as the energy costs of task migration and the energy costs due to the actual state transition.

The model presented by Simone cannot be considered as complete as it does not fully represent the state costs in reality. The assumptions results in the model not being completely accurate. Furthermore, the model only uses three power states (running, sleep and deep sleep). To use only three states is rarely enough for effective power management in real, modern, solutions. The model does, however, produce rough cost estimates. More importantly, the model emphasizes the use of core allocation and the affect of transition latencies in system level power management.
3.3 A system level power management simulation framework

A simulation framework has been developed at Åbo Akademi University. The purpose of the framework is to demonstrate the benefits of using sleep states and performing power management on a system level. The framework has been created to simulate dynamic core allocation in a many-core system. The core allocation is done partly based on an incoming amount of requests and partly based on the ability of the system to handle incoming requests. By allocating cores, the framework aims at minimizing the energy consumption and improving the energy proportionality for the system as a whole.

3.3.1 Approach

The current version of the framework, which has been built in Simulink, is based on the design of a PID controller. The framework controls the power state of the cores according to incoming requests and the monitored quality of service (QoS). The QoS is calculated as the difference between the incoming requests and the capacity of the system. The QoS, which is measured in percent, acts as the feedback mechanism for the PID controller. The capacity of the system is determined by the states of the cores.

Currently, the framework supports one sleep state and DVFS. These methods are applied for each processor core in the system. The framework is set to first adjust the number of sleeping cores according to the PID calculations. The sleep states are set first since they have a larger impact on the power consumption and performance. After it has been decided how many cores will be in the sleep state and how many cores will be active, DVFS is applied on one core. The last core is set to run at a voltage and frequency that brings the performance of the system as close to the PID calculated value as possible. The amount of cores in the working state defines the performance of the system on a large scale, while applying DVFS on the last core adds the ability to do more fine-grained performance adjustments.
The framework includes parameters that must be defined by the user. The power dissipation of the cores needs to be defined so that the total energy consumption can be estimated. The power dissipation must be defined for all power states available in the framework. Future development of the framework includes adding more parameters that affect power management in a many-core system. Such parameters are for example the costs of both transition latencies and any needed task migrations. Values for parameters used in the simulation framework have been measured. These measurements are presented in Chapter 4. In Chapter 5 we present more measurements that have been done to support future development of the simulation framework.

### 3.3.2 Results

The energy consumption is estimated and the gain of using system level power management is evaluated for the generated requests. Figure 3.4 shows the output of a simulation. The top curve (A) represents the incoming requests of the simulation. The requests were randomly generated for the simulation in this figure. Curve B shows how the value of the QoS changes over time. Even if the framework tries to maximize the QoS, rapid increases in incoming requests result in the QoS dropping below 100 %. Curve C represents the amount of cores that are in the working state, i.e. cores that are not sleeping. One can easily note that the power management regulates the amount of operational cores according to the incoming requests. The bottom curve (D) displays the total power dissipation over time, i.e. the energy consumption of the system. The power dissipation is directly calculated from the amount of cores in the sleep state, the amount of cores in the active state and the performance level of the DVFS applied core.

The energy proportionality and power scaling of the system is reviewed by comparing the energy consumption to the incoming requests. A system with no DPM is used as reference to evaluate the results of the simulated power management. The cores of a system with no DPM are statically allocated, i.e. the cores are always in an active state. Furthermore, the system without DPM lacks DVFS implementation. Our simulated system level power
management can reduce the consumption by 50 % compared to the system without DPM, while maintaining the QoS at an average of approximately 95 %. The simulation in Figure 3.4 has an estimated energy consumption of 249 J. The same load is calculated to consume 560 J on the corresponding system without DPM. The QoS level, for the simulated system in Figure 3.4, is on average 94.1 %.

The results of our simulations demonstrate that it is possible to reduce the energy consumption of a many-core system by applying sleep states and DVFS. The performance, QoS and energy consumption of the system depends on how the PID parameters are configured. The PID parameters can be set according to a wanted QoS or energy consumption level. The user can determine the energy efficiency of the system on the cost of reduced QoS. The power management policy can thus be adjusted depending on if the user prioritizes low energy consumption over high performance or vice versa.
3.4 Conclusion

System level power management with load balancing is essential when trying to achieve maximum energy efficiency. The benefits of power management on a system level do, however, not come without problems. A vital part of power management is making correct power management decisions. Making the right decisions implies that the system runs using the optimal power states. Cost estimation is a solution to finding the optimal states. Intelligent and efficient power management require accurate estimations. Several, not only hardware, but also software specific, factors affect the costs.

Understanding subjects such as transition latencies is required to make the correct power management decisions. The affect of transition latencies must be considered in the implementation of effective power management. Simulating system level power management gives an indication of the results of using power management methods (e.g. sleep states) on a system level. By developing the simulation framework presented in Section 3.3, we will be able to more accurately represent a real many-core system. More accurate simulations promote the development of an efficient power management policy for cloud infrastructure. In the following chapters we study power scaling and transition latencies on a low-power platform to support future research and more advanced simulations.
This chapter contains a study of power dissipation and power scaling characteristics of a low-cost low-power platform containing a typical embedded microprocessor. The background and goals of this study are presented in Section 4.1. Section 4.1 also includes a description of the test platform. In Section 4.2, the implementation of our test is explained and the results are presented. Finally, the chapter is concluded in Section 4.3 by analyzing the results.

4.1 Introduction

The goal of the study in this chapter is to measure and present values for the power dissipation of a commonly available low-power platform containing an ARM Cortex-A8 processor. The three main purposes of the measurements are:

- To evaluate the power and performance scaling capabilities of an ARM Cortex-A8 platform.
• To provide values that can be used in the simulation framework presented in Chapter 3.

• To support future research on the architecture of cloud infrastructure.

Other purposes include providing values that can be used for calculations to evaluate the energy efficiency of ARM platforms.

Power scaling is studied on the test system by measuring the power dissipation for different performance levels. Comparing the power dissipation of the system running at different performance levels shows how the power scales relative to clock frequency and voltage levels. The relationship between power and performance can be analyzed based on the measured values. We thus obtain a better understanding of how power states should be used to achieve high energy efficiency. The measurements will also demonstrate the energy proportionality of the system. For the study in this chapter, we assume that the performance of the system can be represented by the ARM clock frequency. Further research, on how the actual performance of the system depends on the ARM clock frequency, is required to remove this assumption.

4.1.1 The BeagleBoard test system

The system used for our tests in this chapter consists of a BeagleBoard revision C3 single board computer. The BeagleBoard is an ARM processor-based low-cost platform designed to run on very low power. Figure 4.1a shows the block diagram of the BeagleBoard. The board uses an OMAP3530 applications processor from Texas Instruments. The core of the OMAP3530 is an ARM Cortex-A8 MPU subsystem, which is paired with a TMS320C64x+ digital signal processor (DSP) and a POWERVR SGX graphics accelerator [27]. A detailed block diagram of the whole OMAP3530 is shown in Figure 4.1b.

Ångström Linux, kernel version 2.6.32, is used as the operating system for the BeagleBoard platform in our tests. A standard 2GB Micro SD memory card is used as the primary storage device containing the operating system and the main file system. The BeagleBoard is connected to a PC using the serial port
available on the platform. The PC runs Minicom to communicate with the BeagleBoard through the serial port. The BeagleBoard can thus be remotely controlled from the PC.

Our tests use the operating performance points (OPP) of the OMAP3530. The OPPs are used for scaling the clock frequency and voltage of the ARM MPU subsystem. In the ACPI specifications, the OPPs are represented by the $P_x$ states (see Chapter 2). Table 4.1 shows the available OPPs and the specifications of each OPP. Each OPP defines the supply voltage of the ARM MPU subsystem and the DSP, as well as the individual clock frequencies of both processors.

Table 4.1: Specifications for the operating performance points of the OMAP3530 applications processor [29]

<table>
<thead>
<tr>
<th>OPP</th>
<th>ARM clock</th>
<th>Voltage</th>
<th>DSP clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPP6</td>
<td>720 MHz</td>
<td>1.35 V</td>
<td>520 MHz</td>
</tr>
<tr>
<td>OPP5</td>
<td>600 MHz</td>
<td>1.35 V</td>
<td>430 MHz</td>
</tr>
<tr>
<td>OPP4</td>
<td>550 MHz</td>
<td>1.27 V</td>
<td>400 MHz</td>
</tr>
<tr>
<td>OPP3</td>
<td>500 MHz</td>
<td>1.20 V</td>
<td>360 MHz</td>
</tr>
<tr>
<td>OPP2</td>
<td>250 MHz</td>
<td>1.06 V</td>
<td>180 MHz</td>
</tr>
<tr>
<td>OPP1</td>
<td>125 MHz</td>
<td>0.985 V</td>
<td>90 MHz</td>
</tr>
</tbody>
</table>
4.1.2 Power measurement on the BeagleBoard

The BeagleBoard platform includes a jumper, named J2, to measure the power consumption of the board. J2 is connected in parallel with a resistor, R6, providing the ability to measure the current consumption from the main voltage rail [12]. The current, \( I(t) \), through R6 is given by Ohm’s law:

\[
I(t) = \frac{V_{J2}(t)}{R_6}
\]

(4.1)

where \( V_{J2}(t) \) is the voltage drop over J2, i.e. across R6, for a given time, \( t \), and \( R_6 \) is the resistance of R6. The instantaneous power dissipation of the board is given by

\[
P(t) = I(t)V_{DC}
\]

(4.2)

where \( V_{DC} \) is the voltage of the power supply. Combining Ohm’s law (4.1) with the equation of the instantaneous power (4.2) gives us the instantaneous power dissipation of the board as a function of the voltage drop over J2:

\[
P(t) = \frac{V_{J2}(t)V_{DC}}{R_6}
\]

(4.3)

The BeagleBoard specifications [12] give us: \( R_6 = 0.1 \, \Omega \) and \( V_{DC} = 5 \, V \). The power dissipation can thereby be extracted from equation (4.3) by inserting the measured values of the voltage over J2.

4.2 Power dissipation measurements

The automatic DVFS is turned off using the Linux ACPI. The OPPs of the OMAP3530 are also set through the Linux ACPI. Our measurements indicate that the display subsystem (DSS) of the OMAP3530 dissipates a significant amount of power (approximately 0.8 W). We are, however, not interested in the DSS when studying the power dissipation of the BeagleBoard. The DSS was therefore disabled for all our tests. The voltage drop over the J2 jumper was measured with a multimeter. The voltage was measured for all available OPPs in Table 4.1. Furthermore, the voltage obtained for the system in the ACPI
specified S3 state was measured. After that, the board’s power dissipation for each power state is calculated using equation (4.3) and the measured voltages.

The measurements were done for low and high CPU utilization. The obtained power values for these two utilization levels represent the idle power dissipation and the power dissipation of the system during full load. To create the load, the processor was stressed to a constant utilization level of 100% using a simple program that recursively counts Fibonacci numbers. Table 4.2 shows the obtained power dissipation values for each OPP both during load and during idle. The power dissipation in the S3 state is calculated to approximately 0.2 W.

## 4.3 Analysis

Figure 4.2 displays a graph of the BeagleBoard’s power dissipation at full CPU utilization versus the clock frequency of the ARM MPU subsystem. Figure 4.3 displays the corresponding graph of the power dissipation when the system was idling. The dashed lines show the power as a linear function of the clock frequency. This linear reference line is based on the minimal power to clock frequency ratio. The minimal ratio is obtained at OPP4 during load and at OPP6 when idling. Assuming the actual performance is completely proportional to the clock frequency, the most efficient solution, not taking into account processing time, would thus be to run the board at OPP4. Using a higher OPP does, however, result in faster processing, while the difference in the power to clock frequency ratio is negligible.
Figures 4.2 and 4.3 reveal that the power consumption of the BeagleBoard is not linear. The measurements indicate that the divergence to a linear consumption increases when moving towards lower clock frequency and voltage levels, starting from OPP4. The power dissipation is higher relative to the clock frequency for lower OPPs. The measured idle power dissipation follows the same pattern as the dissipation during full load. The high OPPs, specifically OPP4 to OPP6, offer the highest energy efficiency, assuming that the performance is proportional to the clock frequency of the ARM processor core. There are two possible factors that can result in the lower OPPs having a higher ratio of power dissipation to clock frequency:

- Power dissipation in the parts of the board that are not affected by the OPP change.
- Static power dissipation.
The parts of the board that are not affected by the OPP change forms an overhead. This overhead remains approximately constant when the OPP changes. When the total dissipation increases as the system is set to run at a higher OPP, the constant overhead will stand for a smaller share of the total dissipation. Running the system at a high OPP, with the overhead remaining constant, thus has a lower power to clock frequency ratio. The overhead from parts other than the OMAP3530 are also present in the power dissipation of the system while in the S3 sleep state.

The static power dissipation is not affected by the OPP change as much as the dynamic dissipation and will therefore create an overhead. Similarly as the previously mentioned overhead, the overhead of static dissipation accounts for a larger part of the total dissipation at a low OPP than at a high OPP. The static dissipation is partially eliminated, as described in Chapter 2, when the system is in S3 sleep.
For obtaining an energy proportional system using DVFS, it is not only important that the DVFS results in linear power dissipation, but also that the dissipation during idle is low. We compare the power dissipation of the BeagleBoard during idle with the power dissipation during full load. Figure 4.4 displays the difference as percentage of the full load power dissipation. The graph shows that the system at idle only dissipates around 10 % less power than at full load. The percentage is close to constant for all OPPs.

![Figure 4.4: BeagleBoard power difference for idle and full load.](image)

### 4.4 Conclusions of measurements

In this chapter we have presented measurements of the power dissipation of a low-power platform. The obtained values indicate that the power to clock frequency relationship is not linear for the selected platform. Furthermore, we only obtained a power dissipation reduction of approximately 10 % when the
system was idling compared to the system fully stressed. The BeagleBoard platform is thus not energy proportional and complete energy proportionality cannot be achieved solely by applying DVFS. Parts of the static overhead caused by the board can, however, be reduced for a multi-processor cluster where the overhead per processor is lower. Future research, not only on the power dissipation of a multi-processor platform, but also on the performance to clock frequency relationship is needed to make any definite conclusions about the actual energy efficiency.
CHAPTER
FIVE

BEAGLEBOARD TRANSITION LATENCY

In this chapter we present measurements of power state transition latencies on a BeagleBoard platform. Latencies of power state transitions were introduced and discussed in Chapter 3. The power management methods described in Chapter 2 are applied on the BeagleBoard platform and the transition times are measured.

First, the background and goals of our measurements are presented. Section 5.2 contains a study of the transition latencies when applying voltage and frequency scaling on our system. In Section 5.3 we study the transition latencies when using the ACPI S3 sleep state. We end the chapter by summarizing and analyzing our results in Section 5.4.

5.1 Introduction

In Chapter 3 we explained why power state transitions have latency. We described how these latencies produce a cost that must be considered by the power management. We also showed that a cost model includes these transition latencies. Measuring the latencies on a BeagleBoard enables
us to develop further understanding of the transition latencies of a low-power node in a cloud server cluster. The measurements also grant us the opportunity to add values for transition latencies in future versions of the simulation framework described in Chapter 3. The transitions studied in this chapter are the ones performed by the power management methods currently implemented in the simulation framework, i.e. DVFS and a sleep mode.

5.2 Voltage and frequency scaling

In this section we first present the purpose of our tests. We provide general theory about testing the latencies of voltage and frequency scaling on our platform. We continue with explaining how the tests are implemented. Finally, the results are presented and the obtained results are analyzed.

5.2.1 Theory

The power dissipated from a processor can be reduced by dynamically changing the supply voltage and clock frequency according to the workload as described in Chapter 2. Scaling the operating voltage and clock frequency does, however, involve work. After changing the voltage and frequency of a processor, the voltages and clocks must stabilize before the system is ready to continue executing tasks. When switching the voltage and clock frequency, the processor is therefore unavailable for other use during a period of time. This unavailability period reduces the availability of the processor and causes a delay in the execution time of processes handled by that processor.

Any unavailability period that might affect the system must be considered by the power management making the decision to do the power state transition. In this section we study the unavailability period caused by changing the voltage and clock frequency by measuring the delay on our test system. By measuring the delay, we realize the magnitude of the unavailability period. We are thus able to form an understanding of the overhead of using voltage and frequency scaling in system level power management.
Similarly to the power measurements in Chapter 4, the OPPs of the OMAP3530 are used in the tests described in this chapter for switching between different voltage and clock frequency levels. The unavailability period studied in this section is the delay caused by changing the OPP of the OMAP3530 chip.

5.2.2 Implementation

The platform, which we use to measure the transition latency of an OPP change, is a revision B5 BeagleBoard. The only significant difference to the system described in Section 4.1.1 is that the OMAP3530 of the revision B5 BeagleBoard does not offer full support of OPP6. The BeagleBoard platform includes an expansion connector to route signals to and from the OMAP3530. The pins of the expansion connector was configured and used as general purpose input output (GPIO) pins for the measurements described in this chapter. In our tests, these pins can have two states depending on the voltage level of the pin: either 1.8 V (logic ‘1’) or 0 V (logic ‘0’). The expansion pins used in the voltage and frequency scaling tests were accessed through Linux and are used as output pins.

Expansion pin number 23 was set to alternate between ‘1’ and ‘0’ using a simple program written in C. The source code of the program is found in Appendix A.1. By alternating the voltage on an expansion pin we obtain an alternating signal. The frequency of this signal is dependent on the clock frequency of the ARM processor core. Expansion pin number 24 was used as a trigger signal. The trigger signal gives us the point of time when the OPP transition is initiated. The switch of the OPP was performed using another simple C program that sets the signal on pin 24 from ‘1’ to ‘0’ right before changing the OPP. The source code of this program is found in Appendix A.2.

The expansion pins 23 and 24 were connected to a PicoScope 2205 USB oscilloscope from Pico Technology. A schematic of the test is shown in Figure 5.1. The signals were recorded on a PC using the PicoScope 6 software. The starting point of our measurements were triggered on a falling edge on pin 24, i.e. just before the OPP change was initiated.
When the trigger signal is set to '0' and the OPP is changed, the alternation on pin 23 stops. The system starts running in kernel mode to change the OPP. All user processes, including our program that creates the alternating signal, are frozen. As a result, the alternation on pin 23 stops and the signal remains constant. While running in kernel mode, the system changes the OPP and waits for the frequencies and voltages to stabilize. When the new OPP is reached, the system returns to user mode. The processes that were frozen are allowed to continue and the signal on pin 23 starts alternating again. If the clock frequency of the processor has changed, i.e. if the system is running at a new OPP, the signal on pin 23 will alternate at a new frequency.

The time interval, from the triggering point when the signal on pin 24 is drawn low, to the point when the signal on pin 23 starts alternating between '1' and '0', is the delay caused by the OPP change. This delay, when changing from OPP2 to OPP5 on our test platform, is visualized by the PicoScope 6 software and shown in Figure 5.2. The bottom signal is the trigger signal on pin 24. The topmost signal is the alternating signal on pin 23. The diamond on top of the bottom signal is the triggering point when the OPP change is initiated.

We changed the OPP and measured the unavailability period for that change. To get reliable results, we repeated the procedure one hundred times for each OPP change. The test was done for all possible transitions between OPP1, OPP2, OPP3 and OPP5. We decided to exclude OPP4 from our tests due to time constraints. The data provided by OPP3 and OPP5 are sufficient for our purpose to analyze transitions to and from high OPPs. The transitions are displayed in Figure 5.3. By testing all these transitions we are able to see how
the different transition latencies differ from each other depending on both the initial OPP and the target OPP.

Furthermore, the delay occurring when the system is ordered to change from one OPP to the same OPP was measured. The cases with the target state being the same as the initial state demonstrate the delay caused by software. These delays do not include any time required by the actual change of the frequencies or the voltage on the hardware. A low value indicates that any overhead, caused by the method we use to measure the transition latency, is low. The total amount of measured transitions thus becomes 16.

The amount of measurements was so high that we need to be able to effectively extract the time periods that we are interested in. To extract the latencies, we used a python parser. The parser reads through the comma separated data produced by the PicoScope 6 software. The point of time, relative to the triggering point, when the first rising or falling edge on pin 23 occurs is extracted by the parser for each OPP change. More detailed information and the source code of the parser can be found in Appendix A.3.
5.2.3 Results

The average values, of the one hundred iterations for each transition, are shown in Table 5.1. It is clear that there is a notable period of time during the OPP change when the processor is not available. When we change the OPP, there is a delay in the execution time for any other processes running on the ARM processor. For our system, the delay ranges from 0.725 to 2.320 milliseconds.

<table>
<thead>
<tr>
<th>Initial state</th>
<th>OPP5</th>
<th>OPP3</th>
<th>OPP2</th>
<th>OPP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPP5</td>
<td>0.072 ms</td>
<td>0.736 ms</td>
<td>1.522 ms</td>
<td>2.320 ms</td>
</tr>
<tr>
<td>OPP3</td>
<td>0.729 ms</td>
<td>0.082 ms</td>
<td>1.383 ms</td>
<td>2.010 ms</td>
</tr>
<tr>
<td>OPP2</td>
<td>1.345 ms</td>
<td>1.249 ms</td>
<td>0.165 ms</td>
<td>1.184 ms</td>
</tr>
<tr>
<td>OPP1</td>
<td>1.842 ms</td>
<td>1.610 ms</td>
<td>1.054 ms</td>
<td>0.309 ms</td>
</tr>
</tbody>
</table>
Our measurements show that the time period from when the OPP change is initiated, to when the expansion pin starts alternating again, depends not only on the initial, but also on the target OPP. The measured unavailability periods demonstrate the existence of transition latencies for the power state transitions used when applying DVFS on the BeagleBoard.

5.2.4 Analysis

The latencies are visualized by the graph in Figure 5.4. The graph shows the average transition latency for each one of all the 16 OPP transitions. Our measurements indicate that the transition latency depends on how far apart the initial and target power states are from each other. By studying Figure 5.4, one can note that when the initial and target voltage and frequency levels are close to each other, the transition is not delayed as much as when they are further apart. The significant variance in the latency is caused by the hardware. It takes, for example, longer to clock up the frequency of the ARM from 125 MHz to 600 MHz, than from 500 MHz to 600 MHz.

One of the assumptions of the cost model [25] that was discussed in Chapter 3 is that transition latencies are symmetric. Studying Figure 5.4 one can note that our measured latencies are asymmetric; the pillars on the left side are taller than corresponding pillars on the right. The asymmetric latencies mean that the latency of an OPP change is not equal to the latency of an OPP change in the reverse order, e.g. the time consumed by a change from OPP1 to OPP5 is not equal to the time consumed by a change from OPP5 to OPP1. Our measured values indicate that switching from the higher OPP to the lower OPP always requires more time than vice versa. There are two possible reasons for asymmetric latencies:

- Unequal workloads before and after the state transition.
- Asymmetric hardware latencies.

If the work done before the actual hardware changes its state is unequal to the work done after the change, the latencies will be asymmetric. How fast
the workloads are processed depends on the performance of the system. As the latency always is longer for a switch from the higher OPP to the lower OPP than vice versa, the workload after the switch is larger than the workload before the switch. The workload after the state transition is dominant and the target OPP will thus have a bigger affect on the latency.

The other reason for the difference is variance in the part of the OPP transition that is done on the hardware. This would, for example, indicate that clocking up the processors and increasing the voltage is, on the actual hardware, achieved faster than clocking down the processors and reducing the voltage.

Furthermore, one can note that the measured values when the target OPP is the same as the initial OPP is determined by the OPP. A higher OPP results in a smaller delay. When the initial and target OPPs are equal, the transition latency of the hardware, i.e. to stabilize the voltages and frequencies, is removed. The measured values are instead caused by operations made by the software. The execution time of these operations naturally depends on the clock frequency.
Our measurements confirm the dependence by showing that when the ARM clock frequency is reduced with 50%, the delay is approximately doubled.

5.3 Sleep transitions

This section starts with some background theory about using sleep states for power management. The setup and implementation of our measurements will be explained, followed by a presentation of the obtained values. A short analysis of the results concludes the section.

5.3.1 Theory

Low-power states are an important part of power management for modern data centers. As discussed in Section 3.2, transitions between power states involve work and are not instantaneous. The transition from an idle state to a sleep state requires that the processor is freed of work. This should, however, be done so that the system is able to continue the execution when ordered to wake up.

When moving into a sleep state, the system starts running in kernel mode and all other processes are frozen, the cache memory is flushed and the states of all registers are saved. When waking up from a sleep state, everything needs to be fetched back to the processor before returning to user mode. This way the system can continue handling the processes that were frozen when the sleep transition was initiated, as if the execution never halted. Depending on the sleep state, the transitions to and from that state has different requirements.

Any work and delay caused by the transitions between power states, needs to be considered when creating a power management policy. In this chapter we will study the overhead caused by transitions to and from a commonly used sleep state. By measuring the time these transitions require, we will form an understanding of the cost and the possible delay that comes with such low-power states.
5.3.2 Implementation

In this chapter, our test system is set to the *Suspend to RAM* sleep state using the Linux ACPI. This is the sleep state named S3 in the ACPI specifications [16]. The transition times for the S3 state were measured on the same BeagleBoard revision B5 platform that was used to measure the DVFS transition times in Section 5.2. Interrupts are enabled on expansion pin 8 so that the transition to the working state can be initiated with an external signal. The signals used for the sleep state transition latencies were measured using the same Pico Technology oscilloscope and software that were used for the measurements in Section 5.2.

The sleep process is divided into two phases: transition from the active working state to the inactive sleep state and from the sleep state to the active state. A transition to an inactive state is referred to as a *sleep* transition and from an inactive state to an active state is called *wake-up*. The sleep and wake-up transitions for the domains of the OMAP3530 platform is illustrated in Figure 5.5.

![Figure 5.5: Domain sleep and wake-up transitions for OMAP3530 [29].](image-url)
Wake-up

To measure the wake-up latency we configured the system as illustrated in Figure 5.6. The expansion pin 23 of the BeagleBoard was again set to alternate between logic ‘1’ and logic ‘0’ the same way as described in Section 5.2. To initiate the wake up the system, the voltage on expansion pin 8 was set high. This will cause an interrupt that wakes up the system. The oscilloscope was connected to expansion pins 23 and 8. A transition from ‘0’ to ‘1’, i.e. the wake-up signal, on pin 8 was set to trigger the oscilloscope.

![Figure 5.6: Schematic of wake-up tests.](image)

Figure 5.7 displays how the PicoScope 6 software visualizes the latency for a transition from the S3 sleep state to OPP6 on our BeagleBoard platform. The bottom signal is the trigger signal on pin 8 that starts the wake-up process. The topmost signal is the signal on pin 23 that begins alternating when the transition has finished. The diamond on top of the bottom signal is the triggering point when the state change is initiated.

The system was first put to the S3 sleep state using Linux ACPI, and was then woken up. Using the oscilloscope, we measure time difference from when the signal on pin 8 was drawn low to when the signal on pin 23 started alternating. This is the time from the initiation of the wake-up, to when the system is ready and starts processing the program that controls the signal on pin 23. The results were then extracted from the data using the same parser as in Section 5.2.
Sleep

The latency of a sleep transition cannot be measured using the same method. The procedure used to switch from a working state to a sleep state includes freezing the processes. An alternating signal cannot be used as an indicator for when the system reaches the sleep state because the process that alternates the signal will be frozen before the sleep transition is finished. The time we would obtain using the alternating signal would be the time span starting at the initiation of the sleep transition and ending when the process that alternates the signal is frozen.

Other ways to measure the sleep time, e.g. basing it on when the power dissipation drops, are also problematic. Analyzing the method the Linux ACPI implementation uses to switch from working state to S3 shows that not only the processes are frozen, but also the file systems are synchronized, before the actual transition. A synchronization of the file systems will consume the majority of the time spent for the transition. The synchronization is highly
dependent on factors such as the type and amount of data that has been written since the last synchronization. Furthermore, the delay caused by the synchronization depends on the read and write speed of the storage media. The time needed for synchronization of file systems can thus not be included in the transition latency. To avoid this, the Linux ACPI must be configured to not perform the synchronization. The file systems must instead be synchronized manually before the transition.

5.3.3 Results

The latency was measured a hundred times for the wake-up transition to each OPP. The averages of the measured times are presented in Table 5.2. The fastest average is 644 milliseconds, which was obtained for the transitions from S3 to the working state using OPP5. The slowest transition is the switch from S3 to the working state using OPP1. This transition was on average completed in 709 milliseconds.

<table>
<thead>
<tr>
<th>Target OPP:</th>
<th>OPP5</th>
<th>OPP3</th>
<th>OPP2</th>
<th>OPP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time:</td>
<td>644 ms</td>
<td>649 ms</td>
<td>671 ms</td>
<td>709 ms</td>
</tr>
</tbody>
</table>

The same method, to use one alternating signal and one trigger signal, was also tested for the transition to S3. This resulted in a huge variance in the measurements. The obtained values range from under 50 to over 3000 milliseconds. The test confirms that the method does not measure the whole transition time, but rather the time spent before the processes are frozen. The variation of two orders of magnitude is a result of the difference in delays caused by the file system synchronization.
5.3.4 Analysis

Our measurements show that the transition latency is affected by the target OPP. A lower OPP results in longer transition latency. The system is restored during the wake-up process from the S3 sleep state. The state of the system before moving to the sleep state is restored so that the system can continue executing tasks. The system at a high OPP will perform this work faster than the system at a low OPP.

In the context of the simulated power management in Chapter 3, the important transition is the transition from sleep to the highest OPP. If the power management that uses sleep modes faces a demand that requires more active cores, the system will power management will transition most cores to the highest OPP as DVFS is only used for fine adjustments. Fully accurate cost estimations, nevertheless, require implementation of transition latencies for all OPPs.

5.4 Conclusions of measurements

We have demonstrated how not only the latency of transitions between OPPs, but also the wake-up transitions from the S3 sleep state, have been measured. We have shown that these transitions are not instantaneous by presenting measurements of the transition latency. Our measurements indicate that the transitions depend on both the initial power state and the target power state. It is clear that the less power demanding S3 state needs a longer time to reach an active state than a transition that only involves voltage and frequency scaling.

Our measured values of the transition latencies can be used in future power management research. Our values make it possible to include transition latencies in the simulation framework presented in Chapter 3. Furthermore, the energy consumption of the power state transition can be calculated by combining our measured latencies with values of the power dissipation during the state transitions. This energy cost can, as demonstrated by Simone [25], be a part of the cost estimation used for DPM.
6.1 Conclusions

We have studied energy consumption and power management in cloud infrastructure with the intent to support the development of more energy-efficient data centers. We have discussed the idea behind the approach to use low-power processors in cloud infrastructure. Methods to reduce the power dissipation of servers have also been described. We have given an introduction of how these methods can be applied on a system level for improved energy efficiency. Furthermore, we have shown an example of how the benefits of system level power management can be demonstrated using a simulation framework. How power management should be implemented for maximum energy efficiency in large computer clusters does, however, require further research.

Measurements have been done on a low-power single-board computer containing an ARM Cortex A-8 processor core. Our measurements show that the power dissipation of the platform does not decrease linearly as the clock frequency and voltage of the processor is reduced. The results of our
measurements thus indicate that low-power states are important to be able to achieve an energy proportional processor cluster. Furthermore, latencies of not only transitions between different OPPs, but also transitions from the S3 sleep state to active states, have been measured. The results of these measurements show that the use of sleep modes suffer from higher transition times than the voltage and frequency scaling. The usage of sleep states should thereby be controlled, for example by using accurate cost models.

Improving the energy efficiency of cloud infrastructure is an important research topic. System level power management and low-power processors show high energy-saving potential. Further research, which can be based on the work presented in this thesis, is, nevertheless, needed to be able to fully realize the true benefits of system level power management and low-power processors.

6.2 Future work

The importance of further research will increase as cloud services continue to gain popularity. This thesis creates opportunities for future work that will support the development of energy efficient cloud infrastructure. There is not only a need to review the advantages and disadvantages of using low-power nodes for cloud services, but also a need to continue the research on system level power management.

Future work includes further research on the power and performance scaling capabilities of low-power nodes in a cluster. The power dissipation should be studied for a many-core board. The development of cloud infrastructure using low-power nodes also requires research on the architecture of the cluster. Different designs must be evaluated to establish a suiting memory architecture and an optimal way to interconnect cores. The development of system level power management also requires an analyze of different approaches to implement load balancing.

Furthermore, our transition latency measurements are still to be completed. Without measured values for transitions to the S3 sleep state, the implemen-
tation of latencies for sleep state transitions must be based on the measured wake-up latencies alone. Accurate simulations should, however, include latencies for all power state transitions. Using our measured transition latencies to calculate the energy cost of transitions requires values of the power dissipation during the transitions. Further development of simulation framework should also include support for simulating task migration.
BIBLIOGRAPHY


Evaluering av strömhanteringsattribut för en strömsnål datorplattform

Introduktion

Den växande marknaden för stora distribuerade datorplatssystem, s.k. datormoln, har skapat en växande efterfrågan på servrar och datacenter. Dagens enorma datacenter konsumerar stora mängder energi. Energiförbrukningen har blivit en betydande ekonomisk faktor för såväl konstruktionen som driften av datacenter. Utvecklingen av energieffektiva lösningar har därmed blivit ett populärt forskningsområde.

Detta examensarbete är utfört som en del av ett forskningsprojekt som utreder möjligheterna att bygga energieffektiva processorkluster av strömsnåla mikroprocessorer. Syftet med detta examensarbete är att analysera energiförbrukningen och strömhantningsmöglerheterna för en strömsnål mikroprocessor. Ett av målen är att fastställa nödspecifika parametrar som påverkar prestanda och energiförbrukning i ett processorkluster. De etablerade parametrarna kan användas i framtida forskning för att utveckla en energihanteringspolicy som ökar datormolnens energieffektivitet. Relationen mellan effekt och prestanda
studeras på en strömsnål enkortsdator försedd med en typisk mobilprocessor. Även värden för de etablerade parametrarna mäts för enkortsdatorn.

Energieffektiva datormoln


Forskning [3, 14, 21, 26] visar att strömsnåla processorer designade för inbyggda system har potential att minska energiförbrukningen då de används i stället för traditionella serverprocessorer i datormoln. Strömsnåla s.k. mobilprocessorer har högre energieffektivitet, men erbjuder inte lika hög prestanda som dagens moderna serverprocessorer. En infrastruktur som baserar sig på mobilprocessorer måste därmed bestå av betydligt fler processorer än traditionell serverinfrastruktur för att uppnå motsvarande prestanda. Strömhantering är väsentligt för att minska datorsystemens energiförbrukning. En klusterarkitektur som består av många processorer med låg prestanda ökar behovet av att undersöka effektiva strömhanteringsmetoder ytterligare.
Strömhantering


Mätningar

Som en del av vår forskning har mätningar utförts på en strömsnål BeagleBoard-enkortsdator. Syftet med mätningarna är dels att göra det möjligt att evaluera effektförlusten för en datorplattform försedd med en ARM Cortex A-8-processor, dels att erhålla mätvärden som kan användas i vår simuleringsstomme och dels att på övriga sätt stöda framtida utveckling av energieffektiva datormoln. Våra mätningar delas upp i två grupper: mätning av effekt och mätning av övergångstider.

BeagleBoard-plattform har en kontakt som gör det möjligt att beräkna strömmen, och därmed effekten, som Plattform drar. Genom att mäta spänningsfallet över kontakten beräknades systemets effekt. Mikroprocessorn på vår BeagleBoard-plattform har olika prestandanivåer som definierar klockfrekvenser för ARM-processorn och systemets signalprocessor samt spänningen för båda enheterna. Systemets effekt beräknades för sex olika prestandanivåer
för att skapa en uppfattning för hur effekten påverkas av prestanda. Resultaten av våra effektmätningar visar att förhållandet mellan prestanda och effekt inte är konstant. En låg prestandanivå leder till en större relativ effektförlust jämfört med en hög prestandanivå. Låga prestandanivåer kan inte erbjuda samma energieffektivitet som höga prestandanivåer eftersom den statiska effektförlusten inte elimineras.


**Slutsats**

Vi har undersökt energiförbrukning och strömhantering med avsikt att gynna utvecklingen av ny energieffektiv infrastruktur för datormoln. Metoder att förminska effektförlusten för datorkomponenter har beskrivits. Hur dessa metoder skall appliceras på de omfattande processorkluster som datormoln består av kräver dock fortsatt forskning. Genom simuleringar kan nyttan med att utföra strömkontroll på systemnivå demonstreras.

Vi har även utfört mätningar på en strömsnål enkortsdator. Våra mätningar visar att plattformens effekt inte avtar linjärt då vi reducerar klockfrekvensen och spänningen för processorn. Resultaten av våra mätningar indikerar därmed att vilolägen är betydande i skapandet av effektiv strömkontroll för processorkluster. Vi har även utfört mätningar av tidskraven för övergångar mellan effektlägen. Resultaten av dessa mätningar visar att användningen av vilolägen lider av högre övergångstider än spännings- och frekvensreglering. För att användningen av vilolägen skall vara effektiv måste energihanteringen

69
därmed anpassas enligt verklighetstrogna kostnadsestimeringar.

För att komplettera arbetet som detta examensarbete behandlar, krävs bl.a. mätning av tider för övergångar från aktiva effektlägen till ett viloläge. Fortsatta, mer avancerade och verklighetstrogna, simuleringar bör göras för att skapa en policy för strömhantering på systemnivå. Framtida forskning innefattar även evaluering av olika arkitekturalternativ för hur t.ex. mobilprocessorer och minne kan sammankopplas i ett datormoln.
CODE USED FOR TRANSITION LATENCY MEASUREMENTS

A.1 Code used to create an alternating signal

The following C-code creates a signal on a GPIO pin that continuously alternates between logic ‘1’ and logic ‘0’.

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

FILE *fp;

int main(int argc, char** argv)
{
    // Variables for values that will be written to files
    char set_value[4];
    // Switches value of pin183 between 1 and 0
    #include <string.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
```
char one = '1';
char zero = '0';
int charSize = sizeof(char); //Save charSize

//Set up pin by writing the port to "export"
if ((fp = fopen("/sys/class/gpio/export", "wb+")) == NULL)
{
    printf("Cannot open export file.\n");
    exit(1);
}
rewind(fp);
//Write 183 to the file
strcpy(set_value,"183");
fwrite(&set_value,charSize,3,fp);
fclose(fp);

//SET DIRECTION
if ((fp = fopen("/sys/class/gpio/gpio183/direction", "wb+")) == NULL)
{
    printf("Cannot open direction file.\n");
    exit(1);
}
rewind(fp);
//Write "out" to the file
strcpy(set_value,"out");
fwrite(&set_value,charSize,3,fp);
fclose(fp);
printf("Pin ready...\n");

//SET INITIAL VALUE
if ((fp = fopen("/sys/class/gpio/gpio183/value", "wb+")) == NULL)
{
    printf("Cannot open value file.\n");
    exit(1);
}
rewind(fp);
fwrite(&one,charSize,1,fp);
rewind(fp);
printf("Looping...\n");
A.2 Code used to change clock frequency

The following C-code is used to change the clock frequency of the processor. The code makes the change by using the Linux ACPI. The initial frequency is specified by the first argument (125 MHz if no valid argument) while the target frequency is specified by the second argument (600 MHz if no valid argument).

The code first sets the initial frequency, and then waits five seconds. A GPIO pin is then configured, immediately after which the new clock frequency is set. The code waits ten seconds after each run. If not terminated, the program continues to loop and change the frequency so that multiple tests can be done sequentially.

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Loop until Ctrl-C
while(1)
{
    // Write 1
    fwrite(&one, charSize, 1, fp);
    rewind(fp);

    // Write 0
    fwrite(&zero, charSize, 1, fp);
    rewind(fp);
}

fclose(fp);
return 0;
```
```c
{
    //Variables for values to set freq and pin
    char set_value[10];
    char one = '1';
    char zero = '0';
    int charSize = sizeof(char); //Save size of char, used often

    //INITIALIZE: set up pin and get ready to write

    //Set up pin by writing the port to "export"
    if ((fp1 = fopen("/sys/class/gpio/export", "wb+")) == NULL)
    {
        printf("Cannot open export file.\n");
        exit(1);
    }
    rewind(fp1);
    //Write our pin number to the file
    strcpy(set_value,"168");
    fwrite(&set_value,charSize,3,fp1);
    fclose(fp1);

    //SET DIRECTION
    if ((fp1 = fopen("/sys/class/gpio/gpio168/direction", "wb+")) == NULL)
    {
        printf("Cannot open direction file.\n");
        exit(1);
    }
    rewind(fp1);
    //Write "out" to the file
    strcpy(set_value,"out");
    fwrite(&set_value,charSize,3,fp1);
    fclose(fp1);

    printf("Pin ready...\n");
```

//SET UP FREQUENCY MANAGEMENT
//Set scaling governor
if ((fp2 = fopen("/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor", "wb+"))
    == NULL)
{
    printf("Cannot open scaling_governor file.\n");
    exit(1);
}
rewind(fp2);
//Write userspace to the file to select that governor
strcpy(set_value,"userspace");
fwrite(&set_value,charSize,9,fp2);close(fp2);
printf("Scaling governor set to 'userspace'...\n");

while(1){

//SET INITIAL PIN VALUE
if ((fp1 = fopen("/sys/class/gpio/gpio168/value", "wb+")) == NULL)
{
    printf("Cannot open value file.\n");
    exit(1);
}
rewind(fp1);
//Write 1 as initial value
fwrite(&one,charSize,1,fp1);
rewind(fp1);
printf("Pin value set to starting value (1)...\n");

//SET INITIAL CLOCK FREQUENCY
if ((fp2 = fopen("/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed", "wb+"))
    == NULL)
{
    printf("Cannot open scaling_setspeed file.\n");
    exit(1);
}
rewind(fp2);
//Check input parameters for frequency
if(argc > 1 && (strcmp(argv[1], "720000") == 0 || strcmp(argv[1], "600000") == 0 ||

75
    strcmp(argv[1], "500000") == 0 || strcmp(argv[1], "250000") == 0) )
    strcpy(set_value, argv[1]);
else strcpy(set_value, "125000"); //If no initial frequency specified, set to 125MHz

    fwrite(&set_value, charSize, 6, fp2);
    rewind(fp2);
    printf("Initial clock frequency set (%s)... \n", set_value);

    //Get ready to write new frequency
    if(argc > 2 && (strcmp(argv[2], "720000") == 0 || strcmp(argv[2], "500000") == 0 ||
                    strcmp(argv[2], "250000") == 0 || strcmp(argv[2], "125000") == 0) )
        strcpy(set_value, argv[2]);
else strcpy(set_value, "600000"); //If no target frequency specified, set to 600MHz

    //Write pin value and freq
    fwrite(&zero, charSize, 1, fp1);
    fwrite(&set_value, charSize, 6, fp2);

    //WAIT 5s BEFORE CHANGE FREQ
    printf("Setting new clock frequency (%s) in 5s... \n", set_value);
    sleep(5);

    fflush(fp1); //Set pin
    fflush(fp2); //Set freq

    //Sleep to minimize cpu usage
    sleep(2);

    fclose(fp1);
    fclose(fp2);

    printf("Running again in 10s... Ctrl+c to quit.\n");
    sleep(10);
}

return 0;
A.3 Data extracting Python parser

PicoScope 6 does not support measurements from one rising or falling edge to another. To get these measurement data, the signals are saved as comma separated values, which then is parsed through using the Python-code listed at the end of this section. The comma separated data is arranged as follows: the first column consists of the time stamp and the second two of the values of signal A and B, respectively.

The parser, consisting of code written in Python, parses through comma separated values produced by PicoScope 6. The difference between the values of the second column in two successive rows differ from each other is checked. If the difference is greater than threshold value (delta) the code will stop parsing and print the value in the first column for that row. This will be done for all comma separated value (CSV) files in the current working directory.

By using this code on comma separated values produced by PicoScope 6, the time stamp of the first occurrence in a sudden change, which exceeds the threshold value, on the signal data in column two, will be extracted. As the code parses through all CVS files in the directory, fast extraction of the time stamps of interest is also possible when multiple CSV files are produced by PicoScope 6.

```python
# Parses through all CSV files in working dir and extracts
# the value of the first column in the row where the difference
# of to successive values in the 2nd column exceeds delta.

import os
delta=50

path=os.getcwd()
dirList=os.listdir(path)
for fname in dirList:
    if not fname.endswith("csv"):
        continue

    f = open(fname)
```
lines = f.readlines()
f.close()
prev = float(lines[3].split(","))[1]
for line in lines[4:]:
    current = line.split(",")
    difference = abs(float(current[1]) - prev)
    if difference > delta:
        if float(current[0]) < 0:
            break
        else:
            print(current[0])
    break