Core Energy Efficiency

Seminar “Energy-Efficient Programming”
Dr. Manuel Dolz, Michael Kuhn, Dr. Julian Kunkel, Konstantinos Chasapis, Prof. Dr. Thomas Ludwig

Marcus Soll

Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

2014-11-19
Motivation

- **Goal:** Computers with one ExaFLOPs
  - \(10^{18}\) float operations per second
- Important for more accurate simulations and massive data analysis
  - Biotechnology
  - Nanotechnology
  - Materials science
- Biggest problem: Energy consumption
  - Power consumption needs to be around 20 MW maximum
The goal of “high performance computing” is to achieve computers with one ExaFLOP capacity. This is necessary for advanced simulations and analysis of massive data amounts, for example in the fields of biotechnology, nanotechnology or materials science. The biggest challenge is to reduce the energy to a reasonable amount (max. 20 MW).
Figure: Energy needed for one ExaFLOP based on Green 500. Source: [LPK+13]
Small figure of the theoretical energy consumption needed for 1 ExaFLOP. Although the energy consumption was decreased a lot in the past few years the 20 MW goal is still far away.
Formula for power consumption: $P = C \cdot f \cdot V^2$
  - But each frequency need a specific minimal voltage
  - Reducing voltage also reduces frequency
  - Requirement of advanced power management

This talk will discuss basic principles concerning energy efficiency
- Basic principles of other methods
- Focus: CPU, Memory
The power consumption calculates from the capacitance, the frequency and the square of the voltage. The problem is that the frequency depends on a minimal voltage, so reducing the voltage also reduces the frequency (and therefore the speed of the component). To use this reduction efficient, we need advanced power reduction methods. Therefore this talk presents the most basic methods for reducing energy consumption. This are the basic principles of other methods presented in other talks.
(a) Idle power consumption, all components are utilized 0%.

(b) Load power consumption, all components are utilized 100%.

**Figure:** Distribution of energy consumption. Source: [Min09]
This figure illustrates the power consumption. The highest consumption on a computer is by the CPU and the memory. Therefore we focus on the CPU and the memory in this talk.
Introduction

CPU
  General
  ACPI
  Implementations

Memory
  General
  Movement of data
  Energy reduction

Examples
  ACPI
  Memory

Conclusion
CPU
General information

- The CPU (processor) is the main component of a computer
- It fetches instructions and executes them
- Contains a limited amount of “registers” and gets all other data from the memory
The CPU is the most important part of a computer. Its purpose is to fetch some instructions (usually from the memory) and executes them. For this execution the processor has a limited amount of instructions it can execute (like add, subtract, multiply, read from memory or write to memory). To execute commands quickly a (small) set of data is saved into “registers” which can be reached immediately, everything else has to be saved into memory.
History

- 1965: Moores Law: Computer performance double every 18 month
- Around 2000: Slower growth on single chip - shift to multi core
- Today: Physical limits of multi core systems - shift to many core
In 1965 an observation associated with Gordon Moore was made on single core processors: The performance of CPUs will double every 18 month. Around the year 2000 the growth of performance on single core CPUs was shrinking - therefore the manufacturer decided to build multi core chips, containing multiply cores on one chip to still match this observation. As for today, the growth of performance of multi core processors is shrinking - so we are in another shift to many core systems, containing multiply chips on one platine.
ACPI

- Specification defines an interface for power management
- First released December 1996
- Each device can be controlled through power states
- OS is in control of power management
- Bytecode language (AML)
The ACPI specification defines an interface, through that the operating system can access the power status of computer components. The components can be controlled by assigning different “power states”, each state defining different power consumption and latency. Contrary to prior solutions (like APM) the operating system is in control of the power status. This is important as the operating system can do more accurate decisions than the BIOS. ACPI is defined over a bytecode language which has to be interpreted (AML = ACPI Machine Language).
Figure: Basic ACPI structure. Source: [LSM99]
This picture gives a good overview about the basic ACPI structure. You can see the division into three parts: Operating system, ACPI interface, Hardware.
Figure: ACPI power states. Source: [CCC+13]
This image represents the basic ACPI interface specification. You can see the different subsystems as well as their hierarchy. This slide is here to give a small overview before going into detail.
G-States / S-States

▶ The “global states” ("sleeping states") define the overall system state
  ▶ G0 (Working)
  ▶ G1/S1-S4 (Sleeping)
  ▶ G2/S5 (Soft off)
  ▶ G3 (Mechanical off)
▶ Only in G0 user application are executed
▶ G0 offers further customisation
▶ G2 and G3 require restart of OS
The g-states (global states) control the overall system state. They are divided into four different states. The state G0 represents the normal working mode. The state G1 represents the sleeping mode. The system is still running, but no user threads (application) are executed. G1 is divided into several “sleeping states”. The state G2 is called “Soft off” (or S4). The operating system has to reboot from this state. Almost no power is consumed. The in the state G3 no power is consumed (excluding battery for real-time clock). It is usually entered via a mechanical switch.
C-States

- The “processor power states” (c-states) can be used to control the CPU while the system is in G0-state
- The states differ in latency and power consumption
  - C0
  - C1
  - C2 · · · Cn
- In C0 the processor executes instructions
- In C1 the processor does not execute instructions. Switching to C0 has almost no latency
- All other states are optional and can be defined by the manufacturer
The C-states (control states) can be used while the system is in the G0 state to regulate the power consumption of the CPU. The states differ in power consumption and the time it takes to switch back to C0. In the C0 state the processor executes instructions. In the C1 state the processor does not execute instructions. However it is specified that from this state the processor has to switch to C0 with almost no latency. The C2 and C3 state are specified but optional. All other states can be defined by the manufacturer of the CPU and are not specified.
Deep Power Down Technology

Available on Mobile Penryn Family Processors

- New Power Management State
- Significantly reduces processor power consumed in idle mode
- Further Extends Battery Life

**Figure:** C-states of the “Intel Penryn Family” architecture. Source: [Lin07]
This graphic shows the different c-states in an “Intel Penryn Family” processor. “Deep Power Down” technology state is also called C6.
P-States

- “Performance states” (p-states) enable further control over CPU (and devices) when in active state (C0/D0)
- Up to 16 states (P0 ··· P15)
- Controls the power and frequency of the processor
- Implementation is optional
The p-states offer a way to regulate the CPU (and also other devices, see D-States below) even further while they are in an active state. The implementation of p-states is completely optional and a manufacturer may implement up to 16 states (called P0 to P15).
Figure: P-states of an “Intel Pentium M”. Source: [Cor04]
This graph shows the different p-states of an Intel Pentium M processor together with the power consumed in each state.
Throttling provides an alternative interface to performance control

A throttling-value may be specified

This value determines how much performance (in percent) the CPU should run on

Throttling is ineffective compared to p-states
Throttling is an alternative interface to controlling the CPU performance. Only one (p-state, throttling) can be used at a given time. You can specify the percent of performance a processor should perform. Throttling is done by inserting special no-operation instructions to the CPU execution queue. Because throttling is more expensive than p-states, we should prefer to use p-states instead of throttling.
D-States

- Used to control devices like CD-reader, printer, modems, drives...
- Four states
  - D0 (full-on)
  - D1
  - D2
  - D3 (off)
- Latency and power saving highly dependent on device
The D-states are states based around controlling different other devices. This devices include cd-reader, printer, modems, drives and more. Four states are defined - their meaning (and their latency and power saving) highly depends on the device. For example, a printer might have a high latency (seconds) and high power saving where a drive can not afford those high latency times.
Figure: ACPI power states. Source: [CCC+13]
This image represents the basic ACPI interface specification. You can see the different subsystems as well as their hierarchy. This slide is inserted here to give a summary about the states.
Implementation - Linux

- Core ACPI system implementation called “ACPICA”
  - Does not implement policies
- “ACPI drivers” implement policies
  - C-states are controlled by “idle loop”
  - P-states are controlled by different “governors”
  - Throttling is used on thermal emergencies
The ACPI implementation in Linux is based around a ACPI core (ACPICA) which manages the ACPI. The policies are implemented by different drivers: c-states are controlled via the kernel idle loop, p-states are controlled by different governors like “ondemand” “power saving” “userspace” “performance”, throttling is only used in emergency situations as it is ineffective compared to p-states
Implementation - Windows

- First implementation in Windows 2000 (1996)
- All driver have to register to the ACPI driver
- The ACPI driver calls registered methods on ACPI changes
- The user can influence the power management by “policies”
- Applications can disable certain parts of the power management
The implementation in Windows is based around a ACPI driver. All device drivers have to register call-back methods to this driver. The behaviour of the ACPI driver can be controlled by the user (policies) or certain parts (like screen, sleeping) by applications.
Memory
General

- Second major component in modern PCs
- Cache results of operations
- Goal: Fast, large and cheap
  - Can not be done with current technology
  - Combination of multiple type of memory
The memory is the second major component of a modern PC. In the memory the results of operation should be cached for later use. Therefore some attributes would be nice to have: Memory should be fast to access, keep lots of data and should be cheap to buy. Unfortunately with todays technology we can not achieve all of this points at once, therefor we need to combine different types of memory.
Memory types

- Different memory types build into a hierarchy:
  - CPU-register
  - Cache (L1-cache, L2-cache...)
  - RAM
  - Persistent cache (Hard disk drives, magnetic tape...)

- Different costs and access time
In modern operating system the memory is usually divided into different types (registers, cache, drives...). This different memory types build up a hierarchy where the fastest and most expensive memory is on the top.
Non-uniform memory access

- Provides a single address space off all memory for all CPUs
- All memory can be accessed via unified instructions
- Access to local memory is faster than remote memory
NUMA is an interface to the system memory where all processors share the same address space. This leads to a model where each memory can be accessed via the same instructions. However, the most important point is that local memory is accessed much faster than remote memory. We will keep this point in our mind when we look at the cost of moving data.
Movement of data

- Experimental analysis of data movement costs
  - Average energy cost of moving data is 25%
  - Peak energy cost around 40%
Some experiments show an average energy consumption of 25% for moving data (with peaks up to 40%)
### Movement of data

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy Cost (nJ)</th>
<th>Δ Energy (nJ)</th>
<th>Eq. Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>0.48</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ADD</td>
<td>0.64</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>L1→REG</td>
<td>1.11</td>
<td>1.11</td>
<td>1.8 ADD</td>
</tr>
<tr>
<td>L2→L1</td>
<td>2.21</td>
<td>1.10</td>
<td>3.5 ADD</td>
</tr>
<tr>
<td>L3→L2</td>
<td>9.80</td>
<td>7.59</td>
<td>15.4 ADD</td>
</tr>
<tr>
<td>MEM→L3</td>
<td>63.64</td>
<td>53.84</td>
<td>99.7 ADD</td>
</tr>
<tr>
<td>stall</td>
<td>1.43</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>prefetching</td>
<td>65.08</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Figure:** Energy spend accessing memory (AMD Interlagos 6227). Source: [PWnt]
This table shows experimental results on how much energy access to the different memory component take. There is also a comparison to an “ADD” instruction. E.g. one access to the DRAM equals 99 ADD operations.
<table>
<thead>
<tr>
<th>Energy reduction</th>
</tr>
</thead>
</table>

### Energy reduction - Reduce data movement

- Reduce amount of data movement
- Algorithmic changes
  - Keep data redundant on multiple cores
  - Calculation of data instead storing

---

Marcus Soll
Core Energy Efficiency
One way of reducing the energy consumption is to reduce the data movement itself. This requires changes to today’s algorithms as well as caution in designing new algorithms. One example is to calculate parts redundant instead of moving the data between different cores.
Energy reduction

**Energy reduction- DVFS**

- Dynamically scale down frequency and voltage of DRAM
  - Experimental data suggest average 2.43% power reduction (max. 5.15%) [DFG+11]
  - Experimental data suggest minimal slowdown of average 0.17% (max. 1.69%) [DFG+11]
  - Problem: Data transfers take longer ⇒ more energy consumption
  - Problem: No current implementation
- Better results when scaling CPU and DRAM together
An other way of reducing power consumption is to scale down DRAM frequency and voltage (As the frequency depends on a minimal voltage level). Although giving good results, there are some problems with this approach: There are currently no implementation of this in the DRAM (you have to reboot to change frequency), the data transfer takes longer (this might even increase the power consumption). To address this, you can scale memory and CPU together.
## Examples

<table>
<thead>
<tr>
<th>Introduction</th>
<th>CPU</th>
<th>Memory</th>
<th>Examples</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Marcus Soll

Core Energy Efficiency
Examples - ACPI in Linux

- You can control ACPI in Linux using `cpufrequtils`
  - `cpufreq-info` shows information about current power management settings
  - `cpufreq-set` allows changing current power management behaviour
  - `cpufreq-aperf` measures current power management stats
The tools combined in “cpufrequtils” allow control over ACPI functions. There are three different tools.
~ $ cpufreq-info

cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Bitte melden Sie Fehler an cpufreq@vger.kernel.org.

analysiere CPU 0:
  Treiber: acpi-cpufreq
  Folgende CPUs laufen mit der gleichen Hardware-Taktfrequenz: 0
  Die Taktfrequenz folgender CPUs werden per Software koordiniert: 0
  Maximale Dauer eines Taktfrequenzwechsels: 10.0 us.
  Hardwarebedingte Grenzen der Taktfrequenz: 933 MHz - 2.53 GHz
  mögliche Taktfrequenzen: 2.53 GHz, 2.40 GHz, 2.27 GHz, 2.13 GHz, 2.00 GHz, 1.87 GHz, 1.73 GHz, 1.60 GHz, 1.47 GHz, 1.33 GHz, 1.20 GHz, 1.07 GHz, 933 MHz
  mögliche Regler: conservative, performance
  momentane Taktik: die Frequenz soll innerhalb 933 MHz und 2.53 GHz liegen. Der Regler "conservative" kann frei entscheiden, welche Taktfrequenz innerhalb dieser Grenze verwendet wird.
  momentane Taktfrequenz ist 933 MHz.

analysiere CPU 1:
  Treiber: acpi-cpufreq
  Folgende CPUs laufen mit der gleichen Hardware-Taktfrequenz: 1
  Die Taktfrequenz folgender CPUs werden per Software koordiniert: 1
  Maximale Dauer eines Taktfrequenzwechsels: 10.0 us.
  Hardwarebedingte Grenzen der Taktfrequenz: 933 MHz - 2.53 GHz
  mögliche Taktfrequenzen: 2.53 GHz, 2.40 GHz, 2.27 GHz, 2.13 GHz, 2.00 GHz, 1.87 GHz, 1.73 GHz, 1.60 GHz, 1.47 GHz, 1.33 GHz, 1.20 GHz, 1.07 GHz, 933 MHz
  mögliche Regler: conservative, performance
  momentane Taktik: die Frequenz soll innerhalb 933 MHz und 2.53 GHz liegen. Der Regler "conservative" kann frei entscheiden, welche Taktfrequenz innerhalb dieser Grenze verwendet wird.
  momentane Taktfrequenz ist 2.53 GHz.

analysiere CPU 2:
Example of letting cpufreq-info output. Shows basic information for all CPUs.
ACPI

```sh
$ cpufreq-info -fmc 0
933 MHz
$ cpufreq-info --governor
conservative performance
$ sudo cpufreq-set -g performance
Passwort:
$ cpufreq-info -fmc 0
2.53 GHz
$ sudo cpufreq-set -g conservative
$ cpufreq-info -fmc 0
933 MHz
```
Change the governor and watch the change in frequency
```
~ $ sudo cpufreq-aperf

<table>
<thead>
<tr>
<th>CPU</th>
<th>Average freq(KHz)</th>
<th>Time in C0</th>
<th>Time in Cx</th>
<th>C0 percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1063860</td>
<td>00 sec 048 ms</td>
<td>00 sec 951 ms</td>
<td>04</td>
</tr>
<tr>
<td>001</td>
<td>1089190</td>
<td>00 sec 061 ms</td>
<td>00 sec 938 ms</td>
<td>06</td>
</tr>
<tr>
<td>002</td>
<td>1317160</td>
<td>00 sec 021 ms</td>
<td>00 sec 978 ms</td>
<td>02</td>
</tr>
<tr>
<td>003</td>
<td>1266500</td>
<td>00 sec 002 ms</td>
<td>00 sec 997 ms</td>
<td>00</td>
</tr>
<tr>
<td>000</td>
<td>1089190</td>
<td></td>
<td>00 sec 016 ms</td>
<td>01</td>
</tr>
<tr>
<td>001</td>
<td>1114520</td>
<td></td>
<td>00 sec 008 ms</td>
<td>00</td>
</tr>
<tr>
<td>002</td>
<td>1418480</td>
<td></td>
<td>00 sec 023 ms</td>
<td>02</td>
</tr>
<tr>
<td>003</td>
<td>1393150</td>
<td></td>
<td>00 sec 002 ms</td>
<td>00</td>
</tr>
<tr>
<td>000</td>
<td>0987870</td>
<td>00 sec 022 ms</td>
<td>00 sec 977 ms</td>
<td>02</td>
</tr>
<tr>
<td>001</td>
<td>1215840</td>
<td>00 sec 007 ms</td>
<td>00 sec 992 ms</td>
<td>00</td>
</tr>
<tr>
<td>002</td>
<td>1114520</td>
<td>00 sec 011 ms</td>
<td>00 sec 988 ms</td>
<td>01</td>
</tr>
<tr>
<td>003</td>
<td>1215840</td>
<td>00 sec 028 ms</td>
<td>00 sec 971 ms</td>
<td>02</td>
</tr>
</tbody>
</table>
```
Core Energy Efficiency

Examples

ACPI

Shows information about acpi-stats
Examples - Memory management in Linux

- Algorithm “Dynamic Memory Switching”
- Developed by Prof. Rajat Moona, Sharad Chole, Sanchay Harneja
- Implemented for Linux 2.6.15
- Goal: Switch off unused memory
We will look at an implementation for energy reduction for memory. This algorithm is called "Dynamic Memory Switching" and was developed by Prof. Rajat Moona, Sharad Chole and Sanchay Harneja. It is implemented for Linux 2.6.15. The primary goal is to switch off unused memory.
Dynamic Memory Switching

- New kernel daemon
  - Migrates memory pages and frees parts of memory (banks)
  - Sets banks to low-power state

<table>
<thead>
<tr>
<th>Power State/Transition</th>
<th>Power</th>
<th>Time</th>
<th>Active Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>300mW</td>
<td>-</td>
<td>Refresh, clock, row, col decoder</td>
</tr>
<tr>
<td>Standby</td>
<td>180mW</td>
<td>-</td>
<td>Refresh, clock, row decoder</td>
</tr>
<tr>
<td>Nap</td>
<td>30mW</td>
<td>-</td>
<td>Refresh, clock</td>
</tr>
<tr>
<td>Powerdown</td>
<td>3mW</td>
<td>-</td>
<td>Refresh</td>
</tr>
<tr>
<td>Standby To Active</td>
<td>240mW</td>
<td>+6ns</td>
<td></td>
</tr>
<tr>
<td>Nap To Active</td>
<td>160mW</td>
<td>+60ns</td>
<td></td>
</tr>
<tr>
<td>Powerdown To Active</td>
<td>150mW</td>
<td>+6000ns</td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** Energy of different memory power states. Source: [MCH07]
This is done by copying used memory together and freeing memory banks (parts of the memory). This free, unused memory banks could then be switched to a low energy mode when the memory is not needed. As we can see in the figure, this can reduce quiet some energy, but increase the response time if more memory is needed.
Conclusion

- Core method of reducing energy consumption of CPU
  - ACPI
- Energy consumption of memory
  - Problems
  - Possible solutions
We have looked in this talk over the core methods of reducing energy consumption on CPUs - ACPI. We have also looked on the energy consumption of memory - the problems and the possible solutions.


[Cor09] Microsoft Corporation. 
*Power Availability Requests*, June 2009.

Memory power management via dynamic voltage/frequency scaling.


Software Controlled Power Management.  

[MCH07] Prof. Rajat Moona, Sharad Chole, and Sanchay Harneja.  

