Energy Efficient Programming

Merlin Steuer

Seminar Effiziente Programmierung
Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

24.11.2016

Outline

Motivation

Motivation

- Power Consumption IS a deal nowadays
- DKRZ: $\sim$ 1,2 MW
- $\Rightarrow \sim$ **10,5 GWh/yr**[1]!
- $\sim$ 2800 households (3 persons)[2]

[1] [1]
[2] [2]

More power...

- Top500 No. 2: National Super Computer Center in Guangzhou
- $\sim$ 18 MW [3]
- $\sim$ 160 GWh/yr
- $\sim$ 41000 households [4]
- 10x Reinfeld (Holst.)! [5]

---

[3] [1]
[4] [2]
[5] [3], [4]

Hardware layer

Overview

- DVFS - Dynamic voltage and frequency scaling
- ACPI - Advanced Configuration and Power Interface

DVFS

- *Dynamic Frequency and Voltage Scaling*

[6] [5]

DVFS

- *Dynamic Frequency and Voltage Scaling*
- $P_{static} = m \cdot V$ [1]
  - $m = const.$
  - V: Core voltage

[6] [5]

DVFS

- *Dynamic Frequency and Voltage Scaling*
- $P_{static} = m \cdot V$ [1]
  - $m = const.$
  - V: Core voltage
- $P = \frac{1}{2} C \cdot V^2 \cdot f + P_{static}$ [6]
  - C: capacitance of switched circuit
  - V: Voltage of switched circuit
  - f: Frequency

[6] [5]

DVFS

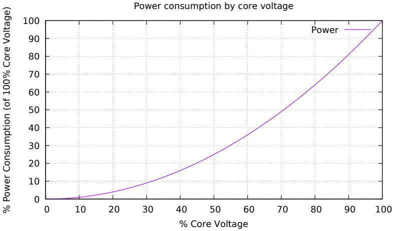

Figure: Power vs. Voltage

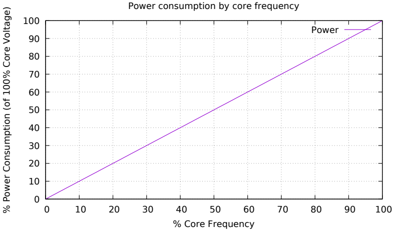

Figure: Power vs. Frequency

DVFS

- Possible f is a function of V
- Lower $f \Rightarrow$ Lower execution speed
- Good for memory-intensive applications
- Not useful for CPU-intensive applications
- **Careful!** Sometimes $f \searrow \Rightarrow$ Memory Bandwith $\searrow$ [7]

[7] [6]

ACPI

- Advanced Configuration and Power Interface
- Industrial standard
- Allows OS-directed control over voltage and frequency of a system
- P(erformance)-States and C(ore)-States

P-States

- P_0: Normal operation, highest performance and power consumption
- $P_i, i > 0$: Reduced power consumption, reduced performance
- Example: AMD Opteron 6128 @ 2.0 GHz [8]

i	V_i (V)	f_i (GHz)
0	1.23	2.00
1	1.17	1.50
2	1.12	1.20
3	1.09	1.00
4	1.06	0.80

[8] [7]

C-States

- Core states
- Cannot be set by code, only by hardware
- $\Rightarrow$ Conditions can be set, so the hardware sets a specific C-State
- C_0: Normal operation
- $C_i, i > 0$: Core halted, transition time to C_0 rises with higher i
- Can be set on core or socket level, depending on CPU

C-States

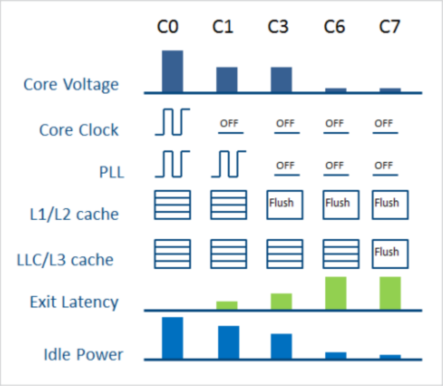

Figure: Different C-States (example)[9]

[9] [8]

OS layer

Overview

- The OS manages all peripherals within a computer system
- $\Rightarrow$ **Can** have a large impact on power consumption
- Can switch different peripherals **within** the CPU to reduce C in $P = \frac{1}{2} C \cdot V^2 \cdot f$

Peripherals

Peripheral	Power consumption (Watt) [10]
CPU	10^2 - 10^3
Graphics Card	10^2 - 10^3
HDD	$\sim$ 10 each
Optical Drives	$\sim$ 10
Network Interfaces	$\sim$ 10
RAM	$\sim$ 3 - 5 per module
Mainboard	25 - 40

[10] [9]

Energy efficient OS

- Graphics card
  - Use lower voltage / frequency
  - If present: use second low-power graphics card
- HDD
  - Reduce load to HDD by caching
  - Use differend Power-(Spindown-)States if disc is idle
- Network interfaces
  - Use lower speeds $\Rightarrow$ switch to 100MBit connection if 1GBit connection is not needed

Energy efficient OS

- CPU
  - Improved Scheduling
  - Enter higher P-States or even C-States when CPU is idle
- Memory: Efficient layout
  - RAM accesses need energy
  - Good memory layout allows better caching $\Rightarrow$ less read-accesses
- Mainboard peripherals can be disabled if unused to save power

Application layer

Overview

- We need to know what happens under the hood
- Have energy efficiency in mind when writing programs
- Generally: Time-efficient programs are often energy efficient

Techniques

- Increasing time efficiency
- Increasing data efficiency
- Letting the CPU rest
- Choosing a language
- Letting the compiler help us
- Libraries

Time efficiency

- Many problems may be solved fast than the trivial approach
- Example: Bubblesort vs. Quicksort
- $\mathcal{O}(n^2)$ vs. $\mathcal{O}(n \log n)$ (Average case)
- Sort a list with 10^6 entries
  - 10^{12} vs. $6 \cdot 10^6$ steps

Algorithms

- Another example: Sub-String searching
  - Trivial aproach: $\mathcal{O}(n \cdot m)$
  - Boyer-Moore: $\mathcal{O}(n + m)$
- Algorithms allowing the CPU to Idle are better
- Recursive algorithms are nice, but usually energy-inefficient (Cache misses)

Data efficiency

- Data efficiency causes less memory operations $\Rightarrow$ more energy efficient
- More cache hits $\Rightarrow$ less execution time
- Time efficiency vs. Data efficiency
- Choose your data structures wisely!

Example: Lists[11]

- Linked List vs. Array
- Linked lists (SE I) seem great, but:
  - Read in $\mathcal{O}(\frac{n}{2})$ vs. $\mathcal{O}(1)$
  - Each container-object is located at different memory locations
  - $\Rightarrow$ Cache miss for nearly **every** step
- Use linked lists only if you append a lot and have no need to often traverse the whole list
- Imagine a bubble sort in a linked list with 10^6 entries

[11] [10]

Logging[12]

- Logging is great during developement
- Logging to the hard disc is causing the drive to spin up very often
- Reduce logging to the minimum needed, maintaining the information
- Cache messages

[12] [11]

Letting the CPU rest

- A sleeping CPU is a good CPU
- Example: Network communications

Example[13]: Network communication: Don't

```
while(true)
{
    // Read data
    result = recv(serverSocket, buffer, bufferLen, 0);

    // Handle data
    if(result != 0)
    {
        HandleData(buffer);
    }

    // Sleep and repeat
    Sleep(1000);
}
```

13 [8]

Example: Network communication: Do

```
WSANETWORKEVENTS NetworkEvents;
WSAEVENT wsaSocketEvent;
wsaSocketEvent = WSACreateEvent();
WSAEventSelect(serverSocket, wsaSocketEvent,
    ↪ FD_READ|FD_CLOSE);
while(true)
{
    // Wait until data will be available in the socket
    WaitForSingleObject(wsaSocketEvent, INFINITE);
    // Read data
    result = recv(serverSocket, buffer, bufferLen, 0);

    // Handle data
    if(result != 0)
    {
        HandleData(buffer);
    }
}
```

When talking about communication[14]

- Network communication needs energy
- Keep your protocols slim
- reduce overhead to a minimum
- Send big messages less frequent than small ones very often
  $\Rightarrow$ less CPU-wake-ups

[14] [11]

Choosing a language[15]

- Lower level languages like C, C++ are better for energy efficiency
- More control over memory and program flow
- High-Level languages (C#, Java) are convenient, but coder has less control
- Prefer compiled languages over interpreted ones

[15] [10]

Letting the compiler help us[16]

- If possible, let the compiler use architecture-specific instruction sets
- Hardware acceleration gives a huge boost in efficiency
- Let the compiler optimize the code (**-Ox**)
- Less instructions per task $\Rightarrow$ better efficiency

[16] [10]

Libraries

- A Lot of tasks have been solved already
- Libraries save time and money
- Multithreading/Parallel computing may give a good performance improvement:
  - OpenMP
  - MPI
- Search for Libraries before inventing the wheel over and over again

Conclusion

Conclusion

- To make a program energy-efficient, it's good to know what happens in hardware
- Know what the OS does to make a system energy efficient
- Simple rules make programs more efficient
- Think about your problem - The trivial approach is the best in rare cases only

Thank you.
Questions?

Literature

Literature I

[1] Top500.org. Top500 list november 2016. Accessed: 15.11.2016. [Online]. Available: https://www.top500.org/list/2016/11/

[2] Die-Stromsparinitiative.de. Stromverbrauch im haushalt: Durchschnitt und spartipps. Accessed: 15.11.2016. [Online]. Available: http://www.die-stromsparinitiative.de/stromkosten/stromverbrauch-pro-haushalt/

[3] B. für politische Bildung. Bevölkerung und haushalte. Accessed: 15.11.2016. [Online]. Available: http://www.bpb.de/nachschlagen/zahlen-und-fakten/soziale-situation-in-deutschland/61584/bevoelkerung-und-haushalte

Literature II

[4] Reinfeld.de. Starting page. Accessed: 15.11.2016. [Online]. Available: http://www.reinfeld.de/

[5] M. Travers, "CPU Power Consumption Experiments and Results Analysis of Intel i7-4820K," p. 5, 2015. [Online]. Available: http://async.org.uk/tech-reports/NCL-EEE-MICRO-TR-2015-197.pdf

[6] D. M. Robert Schone, Daniel Hackenberg, "Memory performance at reduced cpu clock speeds: An analysis of current x86 64 processors," p. 3, 2014. [Online]. Available: https://pdfs.semanticscholar.org/8668/5044b78aed871688f4c7e8d95b4b62538570.pdf

Literature III

[7] M. Dolz, "Paving the way towards energy-aware high-performance-computing," p. 18, 2014. [Online]. Available: https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2014_2015/eep-1415-eehpc.pdf

[8] Cprogramming. Green code developement. Accessed: 15.11.2016. [Online]. Available: http://www.cprogramming.com/appup6.html

[9] buildcomputers. Power consumption of pc components in watts. Accessed: 15.11.2016. [Online]. Available: http://www.buildcomputers.net/power-consumption-of-pc-components.html

Literature IV

[10] D. Lohmann, "Energy-aware programming techniques," 2014. [Online]. Available: https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2014_2015/eep-1415-lohmann-programming-techniques.pdf

[11] C. Weekly. 8 ways to make your software applications more energy efficient. Accessed: 15.11.2016. [Online]. Available: http://www.computerweekly.com/blog/Green-Tech/8-ways-to-make-your-software-applications-more-energy-efficient