Energy Efficient Programming

Merlin Steuer
2steuer@informatik.uni-hamburg.de

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

01.03.2017

Abstract

This article is a written report for a presentation given within the seminar *Effiziente Programmierung*. The target of the presentation was to give the listeners an overview over the fundamentals and techniques of energy efficient programming.

Beginning with the basics, the potentials to save energy are described for the hardware layer, the operating system and the application layer, with a focus on the latter, as this is the field the majority of listeners is most experienced in and will work on in the future.

Contents

1 Introduction

With rising energy prices and the awareness that high energy consumption has an effect on the environment, being energy efficient becomes more of a deal these days. Being efficient in using the available ressources is now a big requirement everywhere, especially in software and hardware projects.

Looking at High Performance Computing (HPC), the DKRZ[1] has an annual energy consumption of approx. 10.5 GWh[1] at a power as high as 1.2 MW. This is the electrical energy 2800 households of 3 persons use per year[2]. These numbers seem to be quite big, but get smaller when looking at the Top500 No. 2, the National Super Computer Center in Guangzhou, China[1] which has a power consumption of 18MW, resulting in 160GWh of electrical energy being used every year, which is as much as 41000 households[2] or 10 times as much as a small German city like Reinfeld consume[3].

But not only in HPC, but also in every mobile device energy efficiency is a big problem. To give an example: Batteries of mobile phones should last for at least a day with rising display sizes, smaller space for the battery inside the housing and bigger processors.

Within the next sections, the fundamental basics of writing energy efficient programs will be described by going bottom-up from the hardware layer to the application layer, outlining possibilities and potentials to save energy within each layer.

2 Energy saving at the hardware layer

Having a wide variety of processors (and generally hardware) in the field makes finding a generic approach of saving energy directly within the hardware complicated. At the same time, saving energy directly within the hardware layer has the most potential, since it may be possible to save energy without user-code interaction. Two terms will be discussed here: *Dynamic Voltage and Frequency Scaling* (DVFS) and *Advanced Configuration and Power Interface* (ACPI). The combination of both create a partly-automatic approach to saving energy in the hardware layer.

2.1 Dynamic Frequency and Voltage Scaling (DVFS)

DVFS is a very basic, but still very powerful approach to saving energy in a clocked system[2] as it makes use of their physical properties. The electrical power used by a processor is a combination of two seperate terms:

- $P_{static} = m \cdot V$[4] with m being a constant of the unit $\frac{W}{V}$ and V being the core voltage the processor is clocked at.

  This term represents the power used statically by a processor. It is independent from the clock frequency and is a result of leakage currents within the semi-conductors inside the circuit.

- $P = \frac{1}{2}C \cdot V^2 \cdot f + P_{static}$ [4] with

[1] Deutsches Klimarechenzentrum

[2] Every processing unit in computers are clocked systems, meaning the voltage within the circuit changes cyclically.

- C being the capacitance of the clocked circuit[3]
- V being the voltage the circuit is clocked at
- f being the frequency the circuit is clocked at

This term represents the overall power consumption of a clocked system. To save power, we need to reduce one or more of the parameters C, V and f. The constant m is usually nothing which can be influenced easily, since it highly depends on the layout of the circuit. Changing C can be done in small dimensions in modern processing units by disabling peripherals within the processor. This will be discussed in section 3.

What can be done quite easily and has a big effect on the power consumption of a system is changing the voltage and frequency of the clocked system. Let us first look at the voltage as the power is dependent quadratically on it. Reducing the voltage by 50% means saving 75% of the energy. Also reducing the frequency by 50% halves energy consumption.

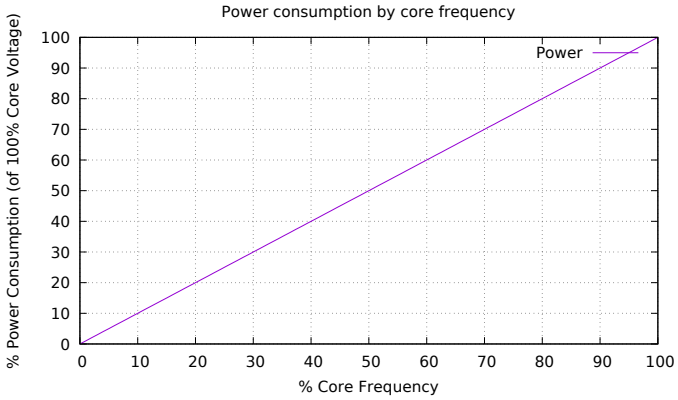

Figure 1: Power vs. Frequency

Figure 1 and 2 give a graphical representation of the dependencies on frequency and voltage within the above formula.

2.2 Advanced Configuration and Power Interface (ACPI)

ACPI is an industry standard used in every processor for home computers and servers. It makes use of the previously described DVFS to reduce power consumption of the processor. To accomplish this, it introduces so-called P- and C-States, which represent different states of the processor.

2.2.1 P-States

[3]To speak in simple words, the capacitance of a clocked circuit is dependent on the number of electrons moved around in every power cycle. This is by far not the physical definition, but should make the point.

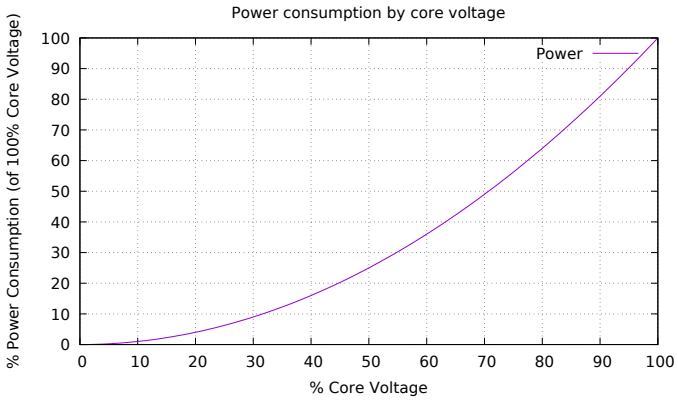

Figure 2: Power vs. Voltage

The P-States (Performance-States) represent the state of the processor according to voltage and frequency. A processor in P_0 is in normal operation with voltage and frequency being at 100%. As the index i in P_i rises, voltage and/or frequency are lowered and by this the power consumption is reduced.

The different P-States may be set by the operating system, allowing it to gradually set the power consumption when the processor is not (fully) needed.

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

Figure 3: Example: P-States of the AMD Opterion 6128 @ 2 GHz[5]

2.2.2 C-States

The C- or Core-States have even more potential to save energy than the P-States, but can not be set by the OS. C_0 again represents the state of the processor, in which it is in normal operation. The difference to the P-States is, that in C_i as soon as i gets bigger than 0, the processor is halted and no more code is executed. As i rises, more and more functions of the processor are switched off, resulting in a higher transition time from C_i to C_0, meaning it takes the processor longer to resume normal operation. Since the operating system cannot directly send the processor into a specific C-State it has to establish conditions, in which the chipset decides to enter a specific C-State.

Figure 4 gives a schematic overview over the C-States. Both P- and C-States may be set on socket- or core-level, depending on the processor used.

3 Energy saving at the OS layer

The operating system is a piece of software with the main task of managing and serving resources for the user code above the OS. It is intuitively clear that the OS can have quite some impact on the overall power consumption of a system, as it is able to set the processor's voltage and frequency (using ACPI) and manage all peripherals found within a typical computer.

Processors nowadays come with a lot of integrated peripherals, mainly used for communication between cores, between processor and peripherals on the

motherboard or between processor and other connected peripherals. These communication modules can, if unused, usually be switched off. Depending on the design of the processor, switching off modules may reduce the capacitance of the clocked circuits, which, as of section 2.1, has linear impact on the power consumption of a processor.

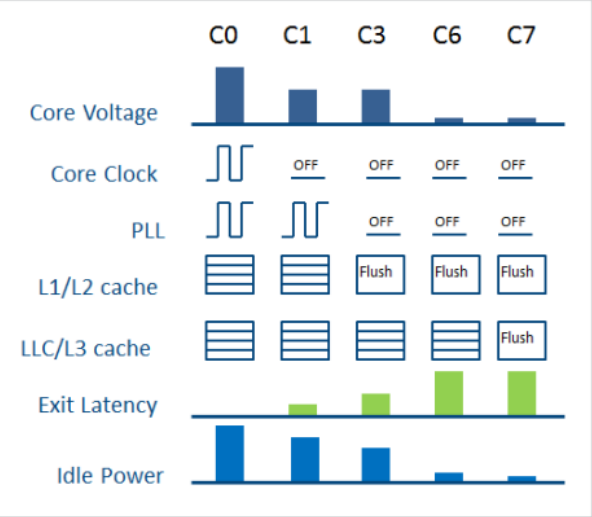

Figure 4: Schematic overview outlining the different C-States[6]

In the following sections the main power consumers within a typical computer system are listed with potentially power-saving actions which may be taken by the operating system.

CPU

As described above, the OS can lower the power consumption of a CPU by lowering the voltage and/or the frequency the CPU is working at. If the operating system has an efficient scheduler there is potential to enter higher P- or even C-States more often when the processor is idle.

Graphics card

For graphics cards, very similar rules apply as for general processing units. When idle, the processing units on the card may be driven at lower speed and lower voltage, resulting in similar power savings. In some computer systems there even are two graphics cards: A high-performance card which is only activated when it's needed and a smaller one, doing the less complex tasks. With this method, a lot of energy may be saved, because the smaller graphics card can have a very specific design for energy efficiency, while the high-performance card can be designed with only the performance in mind.

Network interfaces

Network communication between two (or more) end points takes place by sending clocked data over a long cable. As one might guess, this can also be interpreted as a clocked system like in section 2.1 with the same formulas applying in this scenario.

Having this in mind, reducing the speed of a network interface to e.g. 100Mbit/s instead of 1Gbit/s has a significant impact on the power consumption of the network interface. Of course, this only applies if the faster connection is not needed.

HDD

Hard disks with conventional magnetic discs inside need about 10W each when running[7]. By better scheduling and organizing disc accesses and using optimized caching of data, accesses to the disc can be reduced by the operating system. When the hard drive is not accessed, the motor can potentially be turned off, resulting in a significantly smaller energy consumption. It is important to note, that a hard drive with it's motor shut down takes a relatively long time (10^2 to 10^3 milliseconds) to start up. This should be considered, as the first access to the data is delayed and cannot take place immediately.

Main memory

It is important to have the main memory (RAM) in mind when talking about energy efficiency. RAM accesses need a specific amount of energy, which means that more RAM accesses need more energy. Additionally, accesses to the main memory need quite some time, which means a lot of RAM accesses slow down programs significantly. By mindfully using the caching functionality of a processor, programs can be sped up by multiple orders of magnitude, also resulting in a higher efficiency and lower power consumption.

4 Energy saving at the application layer

Having the basics from sections 2 and 3 in mind, the programmer has probably the biggest impact on the energy efficiency of the code he writes. He has to know what happens under the hood to make good decisions in terms of energy efficiency.

Within the following sections, some general and basic techniques are described which help developers to write energy efficient code. These techniques mostly apply to both conventional and mobile computing, as they are not platform-specific. Operating system- or architecture-specific rules are not discussed within this article.

4.1 Time efficiency

The most intuitive approach to making a program more energy efficient is making it more time efficient. The energy a program needs is proportional to the time it needs to execute, i.e. the CPU cycles needed. Many problems which occur while programming have faster alternative algorithms than the trivial approach. A good example here is sorting. Sorting an array of n elements using the most basic sorting algorithm called Bubblesort takes $\mathcal{O}(n^2)$ steps[4] in average, which is significantly worse than for example Quicksort, having an average time complexity of $\mathcal{O}(n \cdot log(n))$[8]. Using real numbers, sorting an array of 1.000.000 completely unordered elements takes 10^{12} steps using Bubblesort, and only $6 \cdot 10^6$ steps using Quicksort, which is 6 orders of magnitude faster.

Another example is searching for sub-strings of length m in a string of length n. The trivial approach takes $\mathcal{O}(n \cdot m)$ steps[4], while the Boyer-Moore algorithms accomplishes the same task in $\mathcal{O}(n + m)$ steps[8].

[4]Knowledge about the trivial approaches is presumed here

4.2 Data efficiency

Data efficient programs cause less memory operations, which increases both energy efficiency in terms of memory and time efficiency. Data efficiency includes writing programs which cause a lot of cache hits[5], also reducing memory accesses.

Having the memory layout in mind when writing programs is very important for data efficiency. Taking some time to choose the correct data structure for the given scenario may improve efficiency in multiple orders of magnitude. Looking at linked lists (from SE I) in comparison to arrays as data containers clarifies this point: A read access to a linked lists takes $\mathcal{O}(\frac{n}{2})$ time, while reading an element from an array takes place in $\mathcal{O}(1)$, i.e. constant time[9]. Additionally, in a linked list, when reading by iterating over all link containers, each container is usually located in a different memory location, which results in a cache miss for almost every step within the iteration. Algorithms with a lot of read-accesses (like Bubblesort) get very inefficient when using linked lists. A programmer should use linked lists only when a lot of data is appended to the list (which takes place in constant time, independent from the size of the list), but has no need for a lot of read-accesses.

Data efficiency is always a contrast to time efficiency. Algorithms are usually optimized for either data efficiency or time efficiency, resulting in the need for a good and well-considered design by the programmer to choose the best trade-off between both points.

4.3 Logging

Logging is a great tool during development to track and find errors. In production environments, logging can still be extremely useful to find errors that were not found until the time of occurrence. Logging usually is done by writing to the hard drive. When a lot of information is written, the hard drive is constantly running and using energy which could be saved by only logging important information. Additionally, logging log messages and writing them in a batch helps to reduce the load on the hard drive.

4.4 Letting the CPU rest

In terms of energy efficiency, a sleeping CPU is a good CPU. This means we should use techniques which help the OS to set the processor to sleep. One common technique is described below by using the example of two different implementations of network communication.

The implementations of network communication in figure 5 and 6 schematically show two different implementations of reading from a network socket. Although both code snippets accomplish the same thing, the listing in figure 5 prevents the CPU from going to sleep for long times. It polls the network socket for incoming data, if there is nothing to read, it yields the execution of the program for a specific amount of time and then tries again. The delay of one second within this example is relatively long, usually delays in the magnitude of milliseconds are chosen to repeat the query. This type of reading from a network might have performance benefits, but has significant downsides in energy efficiency, as the CPU has to wake up and execute user code for every read operation.

[5] Accessing regions of the memory which are already within the cache of the processor

```
1  while(true)
2  {
3      // Read data
4      result = recv(serverSocket, buffer, bufferLen, 0);
5
6      // Handle data
7      if(result != 0)
8      {
9          HandleData(buffer);
10     }
11
12     // Sleep and repeat
13     Sleep(1000);
14 }
```

Figure 5: A bad implementation of network communication[6]

```
1  WSANETWORKEVENTS NetworkEvents;
2  WSAEVENT wsaSocketEvent;
3  wsaSocketEvent = WSACreateEvent();
4  WSAEventSelect(serverSocket, wsaSocketEvent, FD_READ|FD_CLOSE);
5  while(true)
6  {
7      // Wait until data will be available in the socket
8      WaitForSingleObject(wsaSocketEvent, INFINITE);
9      // Read data
10     result = recv(serverSocket, buffer, bufferLen, 0);
11
12     // Handle data
13     if(result != 0)
14     {
15         HandleData(buffer);
16     }
17 }
```

Figure 6: A good implementation of network communication[6]

The code listing in figure 6 shows a different approach. After configuring the socket, the code tells the operating system to yield code execution until data has been received. The operating system then waits for new data and can decide to send the CPU to power saving states during this period. When data is available, the operating system resumes code execution and the read operation is more likely to be successful. This approach is called *event driven* and is usually the better choice when it comes to energy efficiency. One downside of this approach can be the operating system needing some time to handle incoming data and to wake up waiting programs, so for time-critical applications polling might still be the better solution.

As discussed before, network communication consumes energy on a lot of different layers. Since communication becomes more and more important nowadays, especially for mobile applications further considerations should be done before implementing a communication protocol. Each byte sent over a network (mobile or wired) needs energy, so it is important to reduce communication to the minimum needed when talking about energy efficiency. It is a good idea to

keep protocols slim and reduce communication overhead to the least possible amount[10]. Another thing to keep in mind is - especially considering the above code samples - that is should be preferred to send big messages less frequent than sending small messages very often, as this would wake up the CPU even in event driven implementations very often.

4.5 Choosing a programming language[9]

Before starting a project, a programmer should always put some time into choosing a good programming language for the task to accomplish. A trade-off has to be made between the convenience high-level languages like Java or C# and the abilities to have very granular control on execution and memory management in lower-level languages like C or C++. When it comes to energy efficiency, the low-level languages are almost always the better choice, as there is more potential to control, optimize and improve both memory layout and management as well as execution of single commands. The ability to embed assembler instructions into the code does not necessarily make it easier to maintain, but - if used correctly - improves execution times and energy efficiency. On lower level languages, the compiler might help to optimize the code by using architecture-specific instructions or executing instructions in other orders, which both can have huge impact on the instruction counts per task and by this on the efficiency.

One more important consideration concerns interpreted languages like Python or Ruby. These languages are not compiled into machine code once, but every time a program is started. This increases execution time and gives the interpreter less freedom to optimize the code, because the user wants the program to immediately start. Both points lead to a less energy efficient program, so compiled languages should always be preferred over interpreted ones when writing energy efficient code.

4.6 Libraries

One last thing to have in mind before starting to write a program should be to look for libraries. A lot of problems have been solved already, those solutions have usually been developed over years and include lots of improvements and bug fixes. For multi threading and parallel computing the libraries OpenMP and MPI take a lot of work away from the developer, giving him the chance to focus on the important things. Albeit, the developer should read the documentation of libraries carefully, as the developers of libraries might have had different targets in mind (e.g. performance over energy efficiency), which can lead to unexpected results.

5 Tools to analyse energy efficiency[11]

Quite a variety of tools is available for multiple platforms and architectures to analyse the energy efficiency of programs. These tools usually monitor the overall system action, allowing the data to be analysed in detail after recording it. In this article, two tools will be covered, one for Windows operating systems and one for Unix based operating systems.

This list is neither comprehensive nor complete, but points to the two most important tools on the two most important platforms widely used. There are similar tools available for all other platforms like mobile phones.

5.1 powertop (Unix)

Powertop is a powerful tool to detect energy inefficient applications in Unix/Linux environments. When started while a specific program is running, it records detailed statistics about CPU usage, CPU time, memory usage, caching information and statistics about P- and C-States as well as the different frequencies the CPU was clocked at during measurement. It helps developers to detect applications that cause the CPU to wake up very often and records detailed information about sleep/wake-up times. This tool should be within the basic toolbox of every developer concerned about energy efficiency in Linux/Unix environments.

5.2 perfmon (Windows)

Perfmon is successor of System Monitor from older Windows Versions and has similar abilities like powertop mentioned above. It can display a lot of information about the system in a handy chart or in numerical form. It also has the ability to display all the information in real-time, enabling for faster debugging of applications during development on Windows machines.

6 Conclusion

Writing energy efficient code is not a fully intuitive task. The developer needs to know a lot about the processor, the operating system and the hardware used. By knowing what happens *under the hood*, better decisions can be made in terms of energy efficiency. It is also not easy to write generic programs which are efficient on every platform out there, as requirements for mobile computing and high performance computing for example differ significantly. Nevertheless, some simple rules described in this article help developers in making good decisions and writing more efficient code. It is also always a good idea to look up documentation for the platform used, as there are often considerations already made about energy efficiency. It's important to keep in mind that the trivial approach to a problem is hardly ever the best, so putting some energy into solving problems better potentially improves energy efficiency significantly.

References

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

[2] 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

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

[4] 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

[5] 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

[6] Cprogramming. Green Code Developement. Accessed: 15.11.2016. [Online]. Available: http://www.cprogramming.com/appup6.html

[7] buildcomputers. Power Consumption of PC Components in Watts. Accessed: 15.11.2016. [Online]. Available: http://www.buildcomputers.net/power-consumption-of-pc-components.html

[8] B. e. a. Vöcking, *Taschenbuch der Algorithmen.* Springer: Heidelberg, Berlin, 2008.

[9] 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

[10] 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

[11] F. Goekkus, "Energy Efficient Programming. An overview of problems, solutions and methodologies," p. 52, 2013.