Energy-Aware Programming Techniques

Dominik Lohmann

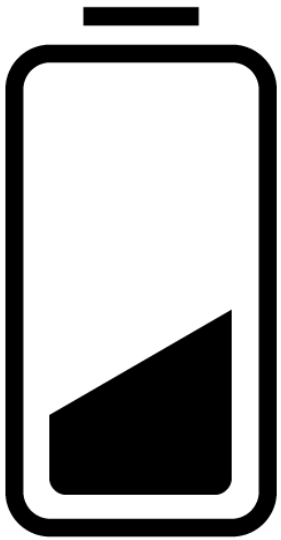
Universität Hamburg
Fakultät für Informatik, Mathematik und Naturwissenschaften
Fachbereich Informatik

64-174 Seminar Energy-Efficient Programming

2014-12-03

Outline

Motivation



Q: How many of you had to charge their phone today?

Q: How many of you had to charge their phone today?

1. Short introduction of myself and of the topic.
2. Reduce the problem to one everyone is having: Their phone battery.
3. The provoking question can be taken up to the HPC level, where energy-efficiency is becoming a real problem.

Energy-Awareness

What is energy-aware programming?

- Focus on efficiency: $\frac{Performance\ \ achieved}{Maximum\ \ performance\ \ achievable}$
- Optimization criterion should be decided based on TCO
- Applications need to be aware of their environment, such as power states

What is energy-aware programming?

- Focus on efficiency: $\frac{Performance\ achieved}{Maximum\ performance\ achievable}$
- Optimization criterion should be decided based on TCO
- Applications need to be aware of their environment, such as power states

1. Short definition of energy-awareness and energy-aware programming
2. Efficiency in programming is the performance achieved in relation to the performance achievable in an optimal setup.
3. Energy-aware optimization is not only based on efficiency, but also by the TCO.
4. In previous presentations, we learned about things like power states, which applications need to be aware of. Another example for enrionment awareness is a power saving mode on laptops, tablets etc.
5. The important bit: We need to not only think about what we code, but how we do it.

Performance

What does it mean to improve performance?

- The software is going to run on a specific, real machine
- There is some theoretical limit on how quickly it can work

Improving energy-efficiency by improving performance is called computational efficiency

"Every circuit not used on a processor is wasting power"
– Chandler Carruth

What does it mean to improve performance?

- The software is going to run on a specific, real machine
- There is some theoretical limit on how quickly it can work

Improving energy-efficiency by improving performance is called computational efficiency

"Every circuit not used on a processor is wasting power"
– Chandler Carruth

1. Introduce the concept behind computational efficiency.
2. Remind people of introduction presentation where we were told that energy-efficiency can be optimized by making the program run faster, because more idling is good!
3. Info about the quote: Chandler Carruth is the head engineer of LLVM at google, where energy-efficiency is a main topic in terms of compiler optimization

Outline

Algorithms

- Complexity theory allows comparing algorithm speed
- Use algorithms that allow the CPU to idle

Improving algorithmic efficiency means solving the underlying problem in another way

- Complexity theory allows comparing algorithm speed
- Use algorithms that allow the CPU to idle

Improving algorithmic efficiency means solving the underlying problem in another way

1. Fallback to our Algorithms & Data Structures lecture, we all once had to visit a lecture like it.
2. Complexity theory can be used to describe the speed of an algorithm.
3. In a previous talk, we learned why idling is good for the CPU
4. Explain how algorithmic improvement is NOT something that can be found, if at all - algorithmic improvement may be huge, but you should not rely on it

Example: Sub-String Searching

- Initially, consider a trivial $\mathcal{O}(n * m)$ algorithm
- Boyer-Moore algorithm is $\mathcal{O}(n + m)$ and can do the same thing (using the end of the needle)

Algorithmic changes can make a huge difference, but are not something that can necessarily be found by everyone

- Initially, consider a trivial $\mathcal{O}(n * m)$ algorithm
- Boyer-Moore algorithm is $\mathcal{O}(n + m)$ and can do the same thing (using the end of the needle)

Algorithmic changes can make a huge difference, but are not something that can necessarily be found by everyone

1. Example for algorithmic improvement
2. Explain both variants, show improvement through Boyer-Moore (including short explanation of how it works.)
3. Would everyone be able to fnd that optimization? Probably not.

Data Efficiency: Hardware Characteristics

One cycle on a 1 GHz CPU	1	ns
L1 Cache reference	0.5	ns
Branch mispredict	5	ns
L2 Cache reference	7	ns
Mutex lock/unlock	25	ns
Main memory reference	100	ns
Send 1kB over 1Gbps network	10,000	ns
Read 4kB randomly from SSD	150,000	ns
Read 1MB sequentially from memory	250,000	ns
Read 1MB sequentially from SSD	1,000,000	ns

Data Efficiency: Hardware Characteristics

One cycle on a 1 GHz CPU	1	ns
L1 Cache reference	0.5	ns
Branch mispredict	5	ns
L2 Cache reference	7	ns
Mutex lock/unlock	25	ns
Main memory reference	100	ns
Send 1kB over 1Gbps network	10,000	ns
Read 4kB randomly from SSD	150,000	ns
Read 1MB sequentially from memory	250,000	ns
Read 1MB sequentially from SSD	1,000,000	ns

1. To show the importance of Cache-Hits, show that L2 Cache ref = 14x L1 Cache ref
2. Talk about the importance of data structures
3. Another throwback to the Algorithms & Data Structures lecture, here the importance of choosing the right data structure.

Example: Linked Lists

- Nodes are separately allocated
- Traversal operations chase pointers to totally new memory
- In most cases, every step is a cache miss
- Only use Linked Lists when you rarely traverse the list, but frequently update it

Linked Lists are rarely what you want to use

- Nodes are separately allocated
- Traversal operations chase pointers to totally new memory
- In most cases, every step is a cache miss
- Only use Linked Lists when you rarely traverse the list, but frequently update it

Linked Lists are rarely what you want to use

1. Example for the possibly worst data structure ever created (or: the one with the worst name)
2. In C++ (on my machine), updating std::vector w/ 100k elements is still faster than updating a linked list of the same size, traversal is more than 200x faster
3. What can we learn from that?
4. - We need to choose the name for our data structures wisely
5. - We need to know what data structure to use in what situation
6. - There is no universally good data structure

Loops

- Minimize the use of tight loops
- Convert polling loops to be event-driven
- Have the lowest polling frequency usable, if polling must be used
- Eliminate busy wait (spin-locks) when possible

Loops

- Minimize the use of tight loops
- Convert polling loops to be event-driven
- Have the lowest polling frequency usable, if polling must be used
- Eliminate busy wait (spin-locks) when possible

1. Explain spinlocks and condition variables (most importantly, their difference in terms of busy wait/idling)
2. Explain what a polling frequency is (has been explained in a previous talk before)
3. Explain CPU pipelines and the effect of branch mispredicts
4. Tight loops probably not as big of an impact as some years ago, because branch predictors are incredibly smart today

Multithreading

- Modern CPUs are able to run things in parallel, allowing faster computation with parallelelized algorithms
- Often requires a (partial) rewrite of legacy applications
- Balancing load across threads allows the CPU to be throttled while maintaining the performance
- Threading done right provides a massive performance boost while having almost no energy impact
- OpenMP, pthreads, TBB and PPL are examples of often used implementations

- Modern CPUs are able to run things in parallel, allowing faster computation with parallelelized algorithms
- Often requires a (partial) rewrite of legacy applications
- Balancing load across threads allows the CPU to be throttled while maintaining the performance
- Threading done right provides a massive performance boost while having almost no energy impact
- OpenMP, pthreads, TBB and PPL are examples of often used implementations

1. Explain what multihtreading is, and also explain some of the models behind OMP, TBB and PPL
2. OMP is good for legacy code, while TBB and PPL feature a rich parallel algorithm library
3. Compare sequential vs parallel energy-efficiency =¿ Multithreading becoming a must nowadays

Performance Libraries/Extensions

- Using (architecture specific) instruction sets such as SSE2 and Intel AVX can often result in increased performance
- Reducing the amount of CPU instructions per calculation directly relates to the applications energy-efficiency
- Certain applications can be optimized using hardware acceleration
  - Focuses mostly on graphics

- Using (architecture specific) instruction sets such as SSE2 and Intel AVX can often result in increased performance
- Reducing the amount of CPU instructions per calculation directly relates to the applications energy-efficiency
- Certain applications can be optimized using hardware acceleration
  - Focuses mostly on graphics

1. Explain SSE2 based on example from PAPO14 (We did a SwarmFlocking Project), where we were able to increase our performance by more than 45% while at the same time not using more instructions
2. This was done using vector mathematics for distance, velocity and force calculations
3. Ratio instructions used / performance becomes interesting
4. Also note things like hardware accelerated video decoding

Compiler Optimizations

- By default, compilers optimize for the average processor
- When possible, enable the use of architecture-specific instruction sets using **-mtune=X** and/or **-march=X** (in gcc)
- Enable general compiler optimization using **-Ox**
- Read your compilers man-page for more details

Proebsting's Law: Compiler advances double computing power every 18 years.

- By default, compilers optimize for the average processor
- When possible, enable the use of architecture-specific instruction sets using **-mtune=X** and/or **-march=X** (in gcc)
- Enable general compiler optimization using **-Ox**
- Read your compilers man-page for more details

Proebsting's Law: Compiler advances double computing power every 18 years.

1. Explain the basic compile flags for optimization
2. Possibly show the man-page (depending on time left, probably not)
   mtune and march can enable SSE2 and similar optimizations
3. The compiler man-pages have lots of detail about this

Programming Language

Consider choosing a programming language, which

- is idle-friendly
- lets you program without any further abstraction layers
- has a minimal runtime
- supports multithreading
- is fast

Languages like Fortran, C and C++ are highly recommended

Consider choosing a programming language, which

- is idle-friendly
- lets you program without any further abstraction layers
- has a minimal runtime
- supports multithreading
- is fast

Languages like Fortran, C and C++ are highly recommended

1. Quick little summary of what we need for a programming language
2. Fortran if you like ugly code, otherwise C or preferably C++ (because of its algorithm library)

Outline

Total Cost of Ownership

- Defines the total operation costs of a computing environment like an HPC cluster
- For most applications, increasing the computational efficiency decreases the TCO
- However, some applications require solution-specific changes

- Defines the total operation costs of a computing environment like an HPC cluster
- For most applications, increasing the computational efficiency decreases the TCO
- However, some applications require solution-specific changes

1. Total Cost of Ownership yet again
2. What if increasing the Computatioal Efficiency also increases our TCO? (happens rarely, if ever)
3. Thus we need solution-specific changes

Energy-to-Solution

- Applications need to be aware of their environment
  - For HPC: Adapting CPU clock speed based on application
  - For Mobile: Respecting power states and energy saving modes to allow switching into low-power modes
- Computational efficiency is not always the optimal solution
- Multiple approaches exist to this
  - Throttling of CPU frequency and threads
  - Adaptive run-time based

Energy-to-Solution

- Applications need to be aware of their environment
  - For HPC: Adapting CPU clock speed based on application
  - For Mobile: Respecting power states and energy saving modes to allow switching into low-power modes
- Computational efficiency is not always the optimal solution
- Multiple approaches exist to this
  - Throttling of CPU frequency and threads
  - Adaptive run-time based

1. Follow-up on the last slide: Dealing with solution-specific setups
2. Environment-Awareness becomes important
3. Respecting the power states of some devices also becoming important, especially in low-battery situations

Adaptive Run-Time Systems

- Measure performance slowdown against CPU energy savings
- Evaluate β-effectiveness on (current) savings
- Adapt CPUs on an HPC cluster based on current β-effectiveness

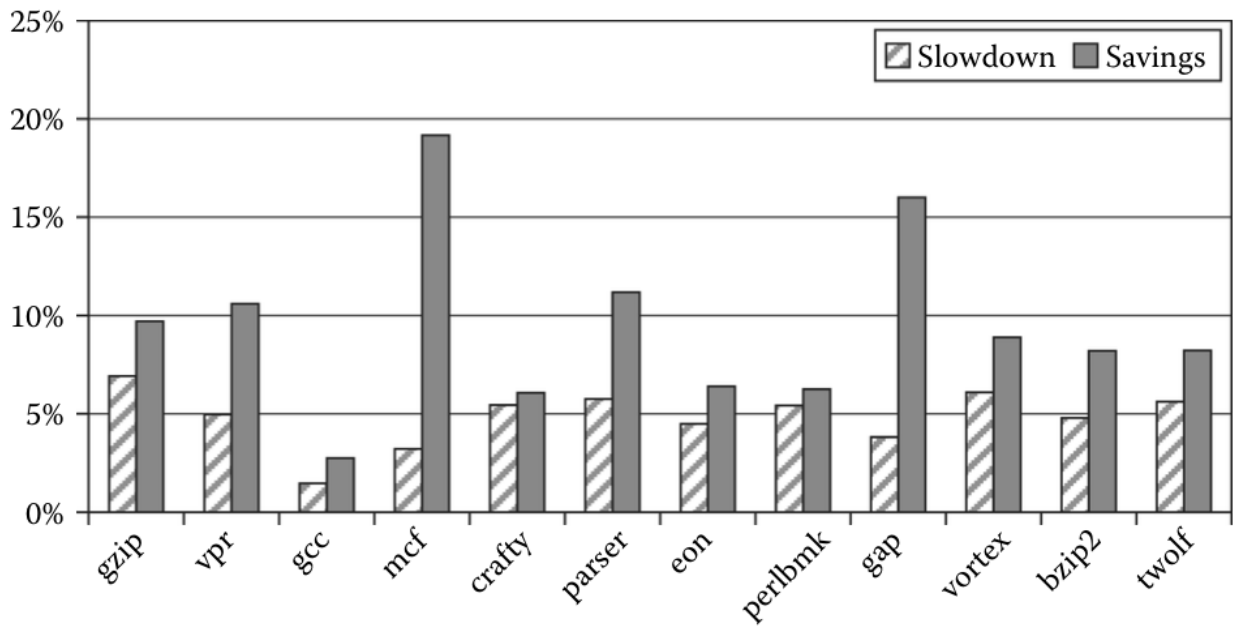


Figure: The actual performance slowdown and CPU energy savings of CPU2000 benchmarks using the presented run-time system

Adaptive Run-Time Systems

- Measure performance slowdown against CPU energy savings
- Evaluate β-effectiveness on (current) savings
- Adapt CPUs on an HPC cluster based on current β-effectiveness

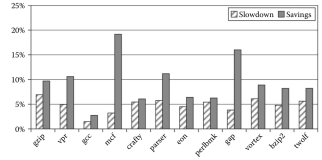


Figure: The actual performance slowdown and CPU energy savings of CPU2000 benchmarks using the presented run-time system

1. Expand on the adaptive run-time system from the last slide
2. Compare slow-down vs. savings
3. BUT: Especially in HPC, Time == Money
4. Probably not worth it, because of TCO also includes the intial cost of the HPC
5. Also, people often rent "time" on an HPC system, so then only performance is important
6. Talk about some of the numbers shown in the graph

Outline

Testing for Energy-Efficiency

- Profile system power during application runtime
  - Understand the impact of Idle and Running states
  - Examine timer interrupts
  - Examine disk and file access
- Measure using tools like **Extrae**
- Check for cache misses and hits using e.g. **perf**
- Focus on optimizing code that is executed a lot
  - This can be checked using e.g. **gprof**

Testing for Energy-Efficiency

- Profile system power during application runtime
  - Understand the impact of Idle and Running states
  - Examine timer interrupts
  - Examine disk and file access
- Measure using tools like **Extrae**
- Check for cache misses and hits using e.g. **perf**
- Focus on optimizing code that is executed a lot
  - This can be checked using e.g. **gprof**

1. Testing is important: Measure, Change, Measure. And Measure again.
2. Short summary of what Idle/Running states, timer interrupts and disk/file access mean for energy-efficiency
3. What should we change based on our results? (bundle file access together etc.)

Example: Extrae and pmlib

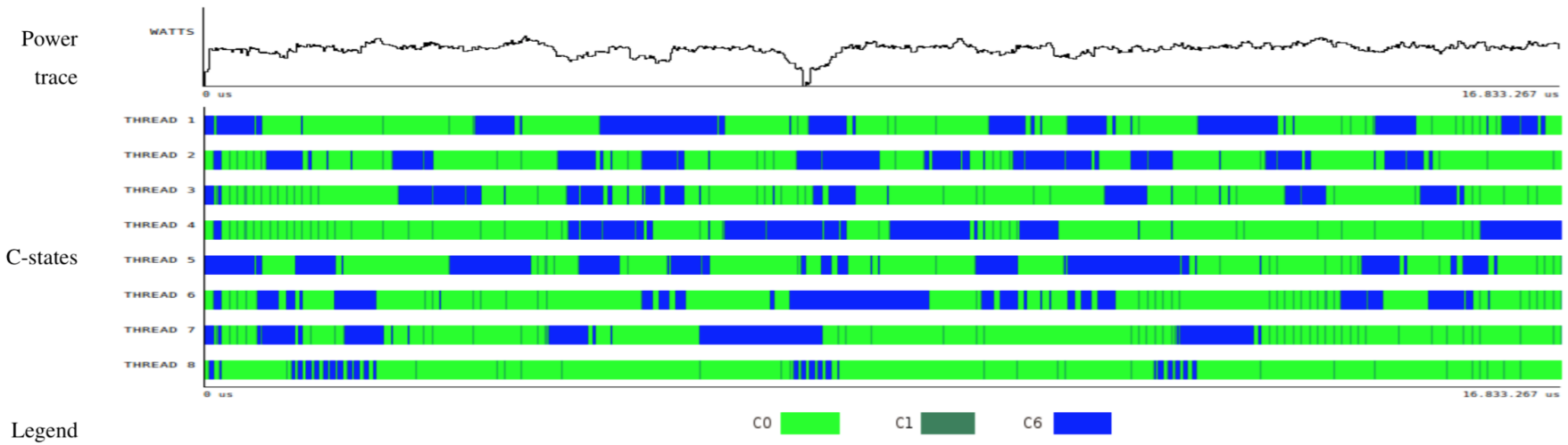


Figure: Power consumption and C-states

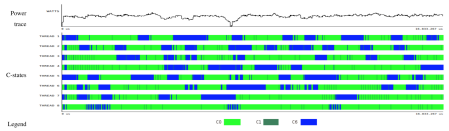


Figure: Power consumption and C-states

1. Show graph (which I believe also was in the introduction presentation)
2. Explain whats visible in the graph and how we can utilize it

Example: perf

```
perf stat -B -e cache-references,cache-misses
             -e cycles,instructions,branches sleep 5

Performance counter stats for 'sleep 5':

  10573 cache-references
   1949 cache-misses         # 18.34 % of all cache refs
1077328 cycles               # 0.000 GHz
 715248 instructions         # 0.66 isns per cycle
 151188 branches

5.002714139 seconds time elapsed
```

Example: perf

```
perf stat -B -e cache-references,cache-misses
             -e cycles,instructions,branches sleep 5

Performance counter stats for 'sleep 5':

  10573 cache-references
   1949 cache-misses        # 18.34 % of all cache refs
1077328 cycles              # 0.000 GHz
 715248 instructions        # 0.66 isns per cycle
 151188 branches

5.002714139 seconds time elapsed
```

1. Perf output for sleeping 5 seconds
2. Especially show the cache-misses/cache-refs rate and its importance
3. Yet again: Example PAPO14-SwarmFlocking - we had a cache-miss rate of about 70%, optimized some data structures and got more performance for just a little bit hassle with the code
4. perf is easy-to-use and there really is no excuse not to use it

Recommendations

Practical recommendations regarding some things said in the previous slides:

- Algorithms: Do not reinvent the wheel
  - You are less likely to get it right by yourself
  - Many programming languages already come with an abstract algorithms library
- Testing: Never trust your instincts, measure instead

Recommendations

Practical recommendations regarding some things said in the previous slides:

- Algorithms: Do not reinvent the wheel
  - You are less likely to get it right by yourself
  - Many programming languages already come with an abstract algorithms library
- Testing: Never trust your instincts, measure instead

1. Some recommendations regarding previous slides that did not make it onto them, but I feel are necessary to mention
2. Why I favor C++ over C: The algorithms library
3. "Measure. Measure again."

Conclusion

By adopting an energy-aware approach to programming, huge energy-savings can be achieved while often also optimizing the performance at the same time.

Programmers need to be aware that even simple to implement things such as the cache-efficient use of data structures or compiler optimizations can often have a huge impact on their applications energy consumption.

By adopting an energy-aware approach to programming, huge energy-savings can be achieved while often also optimizing the performance at the same time.

Programmers need to be aware that even simple to implement things such as the cache-efficient use of data structures or compiler optimizations can often have a huge impact on their applications energy consumption.

1. Quick little conclusion for our everyday-programming

Literature I


Arndt Bode.
Energy to solution: a new mission for parallel computing.


Chandler Carruth.
Efficiency with Algorithms, Performance with Data Structures.


Wu-Chun Feng.
The Green Computing Book: Tackling Energy Efficiency at Large Scale.


Petter Larsson.
Energy-Efficient Software Guidelines.


T. A. Proebsting.
T.A. Proebsting's Law: Compiler Advances Double Computing Power Every 18 Years.

Literature II


Rüdiger Kapitza Timo Hönig, Christopher Eibel and Wolfgang Schröder-Preikschat.
Energy-Aware Programming Utilizing the SEEP Framework and Symbolic Execution.