

Writing Energy-Efficient Software

Abstract

In his article “Harnessing Green IT: Principles and Practices,” San Murugesan (Murugesan, 2008) defines the field of green computing as “the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems—such as monitors, printers, storage devices, and networking and communications systems—efficiently and effectively with minimal or no impact on the environment.” *Using* a computer means that it is powered on and either in an idle state or performing some compute task in which software instructions and data are coursing through the electronic pathways and therefore causing energy to be consumed. This article takes a look into how software influences energy consumption and some ways in which engineers can develop software that uses less.

Computational Efficiency—Hurry Up and Get Idle

Typically, a profile of energy used during a run of the MobileMark '07 benchmark has an idle floor somewhere in the neighborhood of 9 watts, that is, when the processor was not actively working on something, the system consumed 9 W in its idle state. When the system was actively processing a workload, the power use varies, but could be as high as 30 to 40 W. Since we know the computer is not always working and there is a big difference between active power and idle power, it makes sense to conclude that the sooner we complete the compute task or *workload*, the sooner the system will get back to idle and the more energy we'll save. Computational efficiency is achieved by improving an application's performance and achieving a very beneficial side effect of saving energy.

Computational Efficiency Saves Energy

The idea of computational efficiency is to reduce the amount of time

that the system is in a high power state. In other words, if we can improve the performance of the application so that it processes its workload more quickly, the system should return to its idle state sooner and therefore consume less energy overall. We call this concept "*race to idle*"—get the work done as quickly as possible so that the system can return to a low-power state.

Let's look at a hypothetical example. Suppose you have a sorting algorithm that has efficiency $O(n^2)$ and another sorting algorithm that has efficiency $O(n \log n)$, shown in Figure 1. By our reasoning, the $O(n \log n)$ algorithm will deliver better performance and in doing so, save energy.

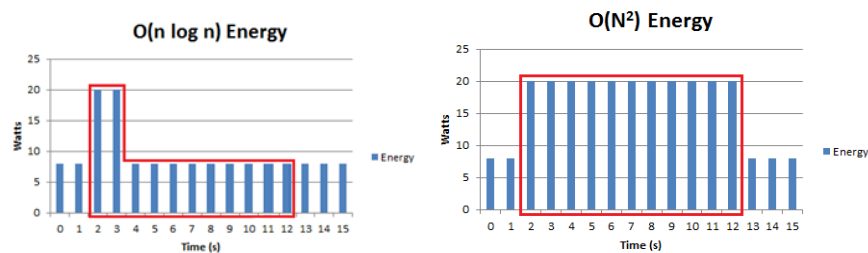


Figure 1 Relative Efficiency of Two Algorithms

Let's examine some data. Suppose our platform has an idle floor of 8 W and the platform, while executing the $O(n^2)$ algorithm on a set of data, consumes 20 W over a period of 10 s. The total energy consumed for the task is 200 joules ($20 \text{ J/s} \times 10 \text{ s} = 200 \text{ J}$). Our $O(n \log n)$ algorithm, by definition, is more efficient so let's assume it takes only 2 seconds to complete the same sorting task. If we assume it also consumes 20 W over the workload period, the total energy to complete the sort is 40 joules. ($20 \text{ J/s} \times 2 \text{ s} = 40 \text{ J}$). To be fair, we need to also add the energy consumed at idle up to a period of 10 seconds, using the same duration as the first test case. So the idle energy consumed is $8\text{s} \times 8 \text{ W}$ or 64 joules. Total energy for $O(n \log n)$ case is therefore $40 \text{ J} + 64 \text{ J} = 104 \text{ J}$. Using a more efficient algorithm has saved us 96 J ($200 - 104$).

Computational efficiency saves energy! That's the theory anyway. Of course the big hole in our logic above is the assumption that both algorithms push the system to consume an identical 20 W. But what if the more time-efficient algorithm has a complexity that drives the system to consume *more* power than the slower algorithm? Could the

total energy for the more time-efficient algorithm be worse? The answer is probably yes, but in the vast majority of cases, the *race to idle* yields favorable results.

Using Computational Efficiency to Reduce Power Consumption

With recent advances in power technology, power consumption has become as important as performance. Energy-efficient software plays a key role in exploring the latest hardware power savings offered by current generation architecture. Poorly-written code can prevent a system from taking advantage of new hardware features.

With mobile platforms, given the limited time that a battery can operate, power consumption becomes even more important. A mobile platform consists of various components such as a CPU, LCD, HDD, DVD, and chipsets, which individually contribute to the power drain of the notebook. Understanding the power contribution of each major component in the platform provides a better view on the total power usage and can provide guidance on optimizing power consumption.

Methods to Achieve Better Computational Efficiency

Now some methods for tuning software for both power and performance will be given, many of which are generic power optimization techniques that can be used on any computing platform. Others, such as vectorization using automatic vectorizing extensions (Intel® AVX), are specific to Intel's recently released microarchitecture codenamed Sandy Bridge. These features will help you optimize software for performance and achieve corresponding power benefits.

Algorithmic Efficiency

Algorithms and data structures are a long-standing area of research in computer science. Considerable effort has gone into research to find more efficient means to solve problems and to investigate and document the corresponding time and space tradeoffs. Theory (and experience) tells us that the choice of algorithms and data structures can make a vast difference in the performance of an application. For a particular problem, a stack may be better than a queue and a B-tree may be better than a binary tree or a hash function. The best algorithm or data structure to use depends on many factors and suggests that a study of the problem and a careful consideration of the architecture, design, algorithms, and data structures can lead to an

application that performs better and consumes less energy.

For example, in an audio processing application it was observed that some sine and cosine functions were repeatedly called on fixed values inside a very high count loop. This unnecessarily increased the computations inside the loop by quite a bit. Since the values on which transcendentals were called remained constant throughout the loop duration, they could have easily been computed prior to entering the loop. Using this approach yielded a 1.3x improvement (30-percent performance gain) on an Intel® Core i7™ system, which led to power savings as well.

Another application identified a high rate of last-level cache misses in a very large and active loop. Further analysis using various performance counters indicated that three loads in the loop were from constant locations and due to this, useful data was getting thrown from other loads out of the cache when the next iteration needed that data, resulting in a high rate of cache misses. This issue was solved by moving the fixed loads out of the loop, which delivered a 1.1x improvement for this application.

Compiler Optimizations

A relatively easy way to achieve better performance is to use an optimizing compiler; most compilers have settings to optimize the code they generate. The authors have done most of our optimization work with the Intel® Compiler (Intel, 2011, Tools), which provides numerous switches to optimize for specific Intel architectures for Windows and Linux platforms. In some video/audio applications, performance gains ranging from 1.1x–1.5x have been attained because the Intel compiler generates excellent code that is much more efficiently executed on Intel architecture and hence leads to performance benefits with corresponding power savings.

Reducing Active Cycles through Multithreading

A well threaded application that uses all available resources will usually finish faster than running single-threaded. A well balanced, threaded application is more likely to provide better power and performance benefits than a poorly threaded application. Choosing the right synchronization primitives also has significant impact on both power and performance.

The graph in Figure 2 shows the result of a study performed to

compare processor power consumption of single-threaded workloads with their multi-threaded equivalents. These applications show 12–60 percent energy savings based on how well they are threaded on a quad-core processor.

The cryptography workload has a 2x performance gain, the multithreaded benchmark application shows 3.9x performance gain, and the media application, with multithreading and other optimizations, shows a 10x performance gain over baseline.

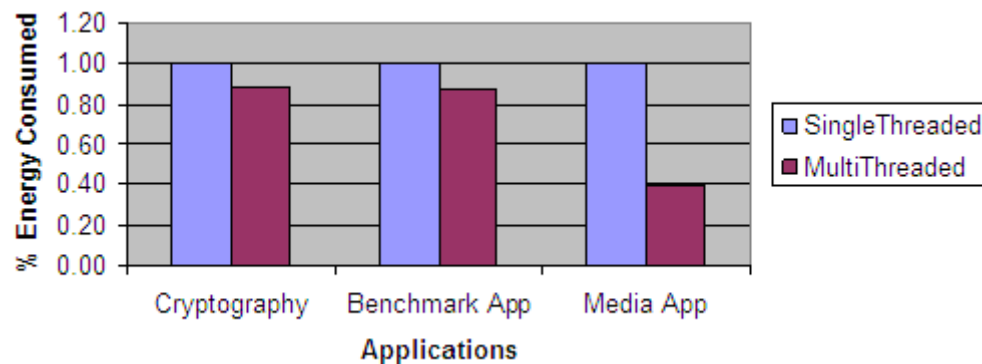


Figure 2 Power Impact of Multithreading

As done in the sorting example above, the power consumption measurements are normalized to the longest runtime. (That is, in case of workloads finishing faster in multithreaded version, the idle power is measured to match a single-threaded runtime). As the data indicates, the power saving depends on how fast the application can finish the task at hand. In case of the cryptography workload, only two threads were doing the real work, so not all four cores were being used. Hence less energy savings are seen between single-threaded and multithreaded version. The media application used all four cores more efficiently and so the energy savings were much greater.

Reducing Active Cycles through Vectorization

Code that performs the same operation on multiple independent data elements is a great candidate for *vectorization*. Modern processors from Intel and AMD provide advanced SIMD (single-instruction, multiple data) instructions (called SSE – SIMD Streaming Extensions) that can perform many simultaneous floating point operations. Intel's

latest Sandy Bridge processor has registers that are 256 bits wide and a set of SIMD instructions called Intel AVX (automatic vectorizing extensions) that can perform *eight* 32-bit floating point operations simultaneously. These vectorization techniques are typically applied to applications that loop on floating point operations over a set of data. There is a little bit more overhead to get the data ready for the AVX instruction but the efficiency of the instruction more than makes up for that.

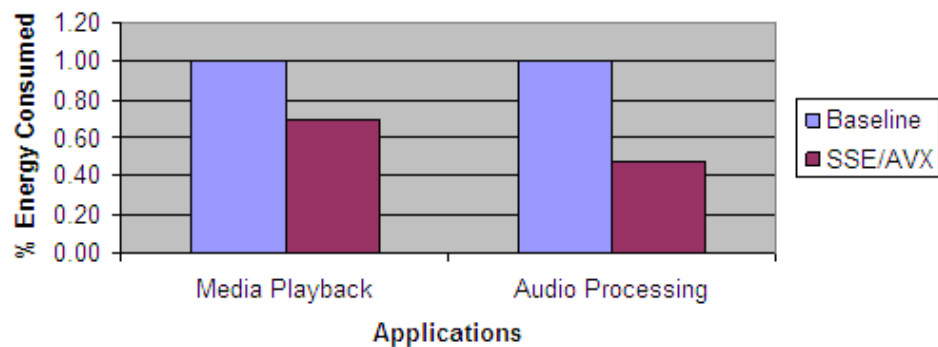


Figure 3 Power Impact of SIMD

The graph in Figure 3 shows the results of optimizing a media playback application with Intel SSE2/4 instructions, yielding a 2.15x performance improvement and 30-percent less energy consumption. An audio processing application optimized with Intel AVX achieved 5x better performance and 55-percent lower energy. It's easy to see that SIMD used well delivers significant performance and power benefits.

Summary

While power management at the operating system and platform levels has intensified, consideration for energy efficiency at the application level is still somewhat nascent. As the world moves toward green technologies and consumer demand for longer battery life in mobile devices increases, the demand for higher performance and new usage models will also continue to grow. Energy efficiency will be crucial for the computing industry in the future both to increase battery life for mobile platforms and to reduce energy expenses for desktop and server platforms. Software behavior can have a significant effect on

platform power consumption and battery life. Modern processors and platforms have many energy-saving features, particularly for performance and the ability to enter low-power states when idle. Software should work in harmony with these features. To get the most benefit, developers should take advantage of performance features by designing computationally efficient, multi-threaded algorithms, and being sure to use a highly-optimizing compiler such as the Intel® Compiler.

Even small improvements when amplified across millions of systems can make a dramatic difference. To simplify access to the status of many platform components, Intel, and others, have built software developer kits that abstract all of the API calls into easily understood function calls. One such tool is the Intel® Mobile Platform SDK.

This article is based on material found in the book *Energy Aware Computing* by Bob Steigerwald, Chris D. Lucero, Chakravarthy Akella and Abhishek R. Agrawal. For more information about minimizing energy use of computers, please refer to the book: Chapter 5 describes methods that help software achieve better energy-efficiency under a workload and Chapter 6 describes methods to improve software idle behaviour and idle efficiency.

Visit the Intel Press web site to learn more about this book:

www.intel.com/intelpress/sum_tmip.htm

About the Authors

Bob Steigerwald is an engineering manager at Intel Corporation. He has over 30 years of industry experience as a software engineer, associate professor of computer science, program manager, and engineering manager. He has spent the past four years leading an Intel team researching methods to improve software performance and energy efficiency.

Christopher D. Lucero has been working in product design for 22 years. His experience ranges from medical to aerospace engineering. For 15 years at Intel, Chris has been focused on thermal/mechanical enabling for embedded applications. He has 13 U.S. patents related to thermal applications for embedded markets.

Chakravarthy Akella has six years of experience as a thermal engineer at Intel working in the field of embedded design. He has three years of

research experience in the field of electronics cooling at the University of Arizona.

Abhishek Agrawal has over 10 years of research and industry experience and currently drives Intel's initiatives on power efficiency for client and Intel® Atom™ processor-based platforms. Abhishek is Intel's representative for the Climate Savers Computing Initiative.

Copyright © 2011 Intel Corporation. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Publisher, Intel Press, Intel Corporation, 2111 NE 25 Avenue, JF3-330, Hillsboro, OR 97124-5961. E-mail: intelpress@intel.com .