Analysis of Compiler Optimizations in Power Consumption

Hari Amoor - Programming Languages & Compilers II

amoor.hari@rutgers.edu

February 21, 2020

Abstract

Energy conservation is an important goal for compilers in various settings, including embedded or otherwise power-constrained systems.

In this write-up, we (1) analyze how power and energy consumption is impacted by different optimization levels, and (2) showcase a C program intended to consume as much power as possible for a fixed time period.

1 Power Consumption of GCC-optimized Programs

We present *Fib*, a C program (given as specified on TX cluster) that creates multiple processes with the Unix *fork* system call. Each process generates random values for some $n \in \mathbb{N}$ and computes the n^{th} Fibonacci number with respect to some initial state defined in *main.h* with the C *double* type (the classical Fibonacci sequence has initial state $f(1) = f(2) = 1$).

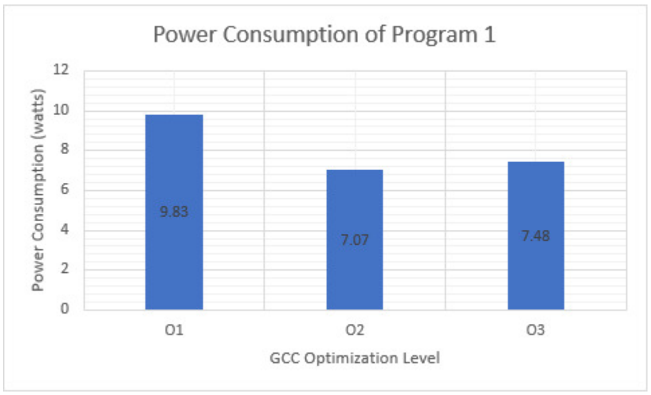

Figure 1: Comparison of *Fib* under different levels of compiler optimization

Each process also writes the computed value to a specified file. The computation of the "Fibonacci" numbers is done in each process in $\mathcal{O}(n)$ time and space complexity; in particular, the implementation uses tail-recursion and floating-point arithmetic.

Consider the power consumption of *Fib* compiled with GCC under optimization options *-O1*, *-O2*, and *-O3*. Figure 1 indicates that *Fib* consumes at most over 9.8 watts of power under *-O1*, whereas it consumes under 7.1 watts when compiled with *-O2*.

The generated assembly code reveals that the program runs recursively as intended when compiled with *-O1*, whereas it runs iteratively when compiled with *-O2*. This suggests that one reason for the decreased power consumption is the tail-recursion optimization conducted by the compiler under *-O2*. Such an explanation stands to reason on the basis that provisioning stack memory for $\mathcal{O}(n)$ recursive calls, as the program does under *-O1*, is very power-intensive relative to the equivalent iterative algorithm.

The increased power consumption with *-O3* relative to *-O2* is less clear. However, GCC enables optimizations like `-ftree-loop-vectorize` involving loop-vectorization that *-O2* does not. Therefore, conjecturally, it is probable that optimizations involving vectorization and unsafe floating-point arithmetic enabled in *-O3* result in greater speed at the tradeoff of slightly greater power consumption.

2 Maximum Possible Power Consumption

We showcase *Mandelbrot*, a program designed to consume the maximum possible amount of power. *Mandelbrot* computes an image file containing a Mandelbrot fractal, then removes the file; constants describing how many times *Mandelbrot* does this, and how many processes it does this on, are parametrized with macros in `main.h`.

Mandelbrot causes at most a 7.76 watt consumption of energy. It utilizes complex floating-point arithmetic, dynamic memory usage, file I/O, and multiprocessing, each of which result in heavy power-utilization.

3 Conclusion

At a high level, optimizing compilers such as GCC optimize for performance not only w.r.t. speed, but also other metrics such as memory and power or energy consumption.

Low-complexity optimizations such as constant propagation and variable elimination can result in strictly-better performance, i.e. improvements in some metrics without any regressions in others, but usage of higher-complexity optimizations such as the ones called for by the *-O3* option can result in tradeoffs between different optimization targets. Case in point, there are many optimizations called for in *-O3* that result in greater speed at the cost of increased power consumption.