Energy Auto-tuning using Polyhedral Approach

Wei Wang, John Cavazos
University of Delaware
101 Smith Hall
Newark, DE 19702
weiwang,cavazos@udel.edu

Allan Porterfield
University of North Carolina - RENCI
100 Europa Dr
Chapel Hill, NC 27517
akp@renci.org

ABSTRACT

How is tuning for lower energy related to tuning for faster execution? Understanding that relationship can guide both performance and energy tuning for exascale. In this report, a strong correlation is presented between the two that allows tuning for execution to be used as a proxy for energy tuning. We also show that polyhedral compilers can effectively tune a realistic application for both time and energy.

1. INTRODUCTION

Tuning an application for power efficiency requires us to understand how energy consumption is related to the application execution time. Knowledge of the relationship between performance and energy can guide the tuning effort.

Using the Resource Centric Reflection daemon tool (RCRtool) developed at RENCI, energy consumption can be measured at a fine granularity for any OpenMP program. Fine-grain measurements enable attribution of energy consumption to particular application regions. This allows accurate study of the correlation between execution time and energy consumption of an application.

In addition to the study of energy and time correlation, we examine a small application with the polyhedral framework to determine the frameworks' effectiveness at reducing overall energy consumption.

2. ENERGY MEASUREMENT-AND-TUNING TOOLS

RCRtool provides user-level fast access to hardware counters. It is able to collect energy consumption from Sandy Bridge processor as well as from Xeon Phi. On Sandy Bridge system, RCRtool keeps track of energy counters exposed by Intel Running Average Power Limit (RAPL) interface. On Xeon Phi, RCRtool obtains the energy information from a file that holds such information. The energy information obtained by RCRtool is available to any OpenMP applications through a simple API that delineates a code region for measurement with a start and end call.

The Polyhedral Compiler Collection (PoCC) was used to generate *polybench* program variants with different optimizations. PolyOpt (a Polyhedral Optimizer for the ROSE compiler) was used to automatically detect SCoPs in larger applications. PolyOpt has better support for side-effect free program features like math functions, allowing some function calls within a SCoP. PoCC and PolyOpt generate hundreds and even thousands of program variants for programs like Polybench and some larger applications.

3. BENCHMARKS FOR ENERGY MEASUREMENT AND TUNING

Using PoCC, *Polybench* program variants were generated using a different set of the optimizations from loop fusion, loop unrolling, loop tiling, loop vectorization, and loop parallelization.

LULESH OpenMP implementation contains 30 parallel regions, 6 of which take up more than 60% of the total application time. In this work, we focus on optimizing the 2nd (largest) parallel region of LULESH because the polyhedral compiler takes too long to transform the largest one.

In addition to the Polybench programs, a 2D monodomain cardiac wave propagation simulation (named *brdr2d*) was used as a test case. Different program variants were generated to explore data locality and parallelism.

4. EXPERIMENTAL RESULTS

The polyhedral framework was first used to examine the energy usage of (and the execution time) of the Polybench programs and LULESH on the Intel Sandy Bridge architecture. A more realistic application *brdr2d* on both the Intel Sandy Bridge and the Intel Xeon Phi architectures is then studied.

4.1 Execution Time and Energy Consumption Correlation

The first experiments verify the relationship between execution time and energy consumption.

Figure 1 shows the relationship between the execution time and the energy usage for the 5135 variants of the *covariance* benchmark, sorted by execution time. The left y-axis shows the energy consumption (in joules) and the right y-axis shows the execution time (in seconds).

The energy line (blue, mostly the bottom) generally follows the time line. The best optimized program variant for time (bottom right in the figure) consumed the least amount of energy. The energy line has many places where 2 runs that take the same amount of time consume significantly different energy. These appear as spikes in the graph. Examining the data, we noticed that the higher energy usage value always had the "maxfuse" flag set.

For 200 variants of LULESH, Figure 2 shows the energy used and execution time. The energy curve mirrors the execution time. No optimizations resulted in a significant increase in power, although the power required did rise slightly (from 160 Watts to 180 Watts - 12% increase).

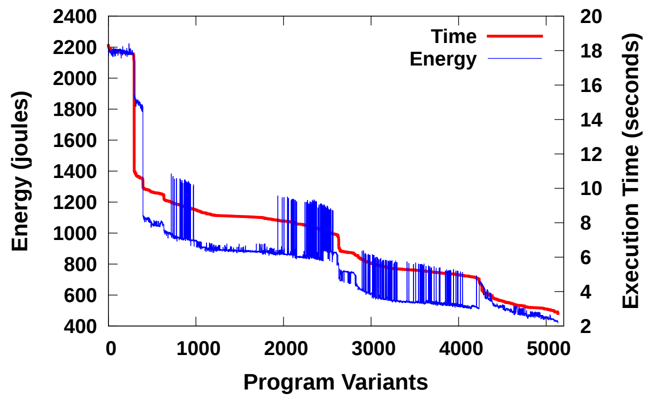

Figure 1: Graph showing the relationship between the execution time and the energy consumption of all *covariance* Polybench program variants on Sandy Bridge Processor (sorted by execution time). The spikes are caused by "maxfuse" loop transformation.

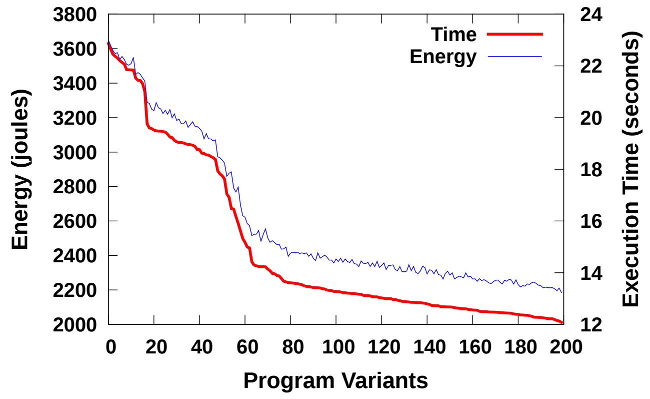

Figure 2: Graph showing the correlation between the execution time and the energy consumption of LULESH program on Sandy Bridge Processor (sorted by execution time).

4.2 Polyhedral Optimization Results on the Realistic Application

Optimizing *brdr2d* on the Sandy Bridge Processor and on Phi coprocessor show the advantage using the polyhedral framework to optimize for both execution time and energy.

Figure 3 compares the best variant for each input size with the base-line OpenMP version. The optimal tiling was different as the input grew. For the 256 case 1×128 resulted in the fastest execution, For the larger cases, the variant with tile size 1×256 was fastest. As the problem size grew the optimized variants' relative performance and energy consumption improved (256 - 2.5% to 2048 - 21%).

Figure 4 shows the *relative* speedups and the normalized energy savings offered by the polyhedral transformations and auto-parallelization. The best PoCC variant of *brdr2d*, is over 20% faster than the baseline Phi version for small sizes. For the largest size, 2048, the best Polyopt/PoCC variant is still slightly better than the baseline and has an absolute speed up of over 150×. The optimal tiling size changes as the input grows. In each case, $1 \times size$ is preferred

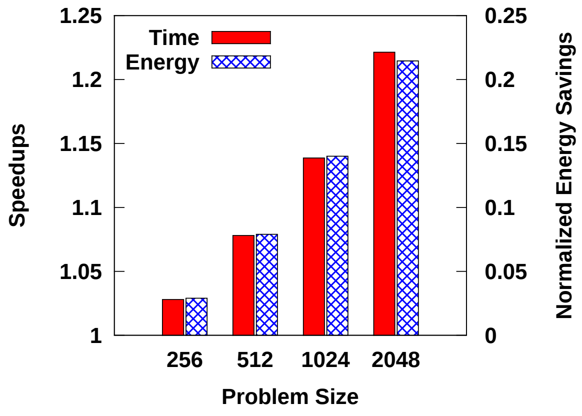

Figure 3: Graph showing the performance improvement and energy savings of the optimal program variant over the baseline OpenMP implementation for different problem size on Sandy Bridge Processor.

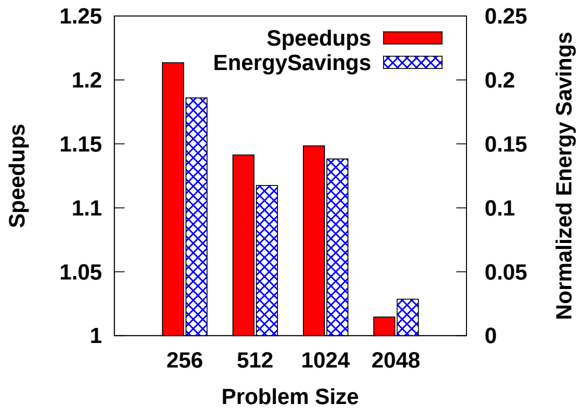

Figure 4: Graph showing the performance improvement and energy savings of the optimal Polyopt/PoCC generated program variant over the baseline OpenMP implementation for different problem size on MIC architecture.

for maximum vectorization. The polyhedral optimizations also improves energy. The energy savings approximately match the relative speedups, ranging from 20% down to 3% as size increases (and baseline vectorization improves).

5. CONCLUSION

Using an auto-tuning framework on a variety of small benchmarks and small applications has shown the high degree of correlation between execution time and energy consumption. Individual optimization can however have significant impact on the Power required by an application. In the Polybench program, *covariance*, using the "maxfuse" option resulted in a 20+% percent power increase. With the correct tile size "maxfuse" also resulted in the 50+% time decrease.

Polyhedral optimization techniques can provide significant increases in performance. For small real applications, like *brdr2d*, polyhedral transformations allow the discovery of effective tiling sizes for the applications.