Energy Tuning of Polybench Programs on Multi-Core Processors

William Killian, Wei Wang, EunJung Park, John Cavazos
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
Email: {killian,weiwang,ejpark,cavazos}@udel.edu

***Abstract*—TODO:**

I. Introduction

Reducing application energy consumption is important to improve user experience of embedded systems, e.g. smart phones, GPS navigators etc. Tuning applications for better energy efficiency is faced with challenges brought by diverse architectures, difficulty of obtaining fine-grain measurements of power, as well as enormous amount of tuning choices. Previous work which performed coarse-grain measurement have provided evidence for the existence of opportunities to auto-tune for energy in parallel applications[1]. This work uses the RCRtool [2], a fine-grain energy measurement tool, to easily attribute energy consumption to particular regions of application kernels and to serve energy tuning. For most applications, nested loops consume the significant portion of the total running time. When tuning the application for better execution time and energy usage, some combination of loop optimizations, including loop tiling, loop unrolling, and loop fusion, are usually performed on the program along with the auto-parallelization. Determining which set of optimizations produces the best results is hard. Polyhedral auto-tuning frameworks have shown promising results at simplifying that effort[3] for small computation kernels like the Polybench programs. This work investigates the effectiveness of such framework to tune for energy on two different multi-core architectures: Intel Sandy Bridge and Many Integrated Core (MIC).

This paper has two main contributions: **TODO: 1)Performance speedups and energy savings on Sandy Bridge and MIC for various Polybench kernels applying loop transformations. 2)Evaluation of architectural differences of energy behaviours corresponding to the same loop transformations.**

The rest of the paper is organized as follows. In Section II, we describe the tools used to measure energy consumptions. Section III describes the benchmarks used for measuring the energy consumptions. Experimental setup, results and analysis are presented in Section IV and Section V. Section VI explains and compares with related work. Section VII has our conclusions.

II. Energy Measurement-and-Tuning Tools

To understand energy consumption, execution time and various optimizations, a light-weight fine-grained measurement RCRtool is required. Finding the optimal combination of compiler optimizations requires a compilation framework, like the Polyhedral Compiler Collection, that easily produces a large number of program variants with specific optimization parameters.

A. RCRtool

The Intel Sandy Bridge architecture allows users to track energy usage through the exposed Running Average Power Limit (RAPL) hardware counter. The RCRtool records the current value of the counter at least 1000 times a second by writing into a shared-memory data structure. This "blackboard" structure provides a hierarchical view of the system where various current performance information is stored. The information is available to any OpenMP applications through a simple API that delineates a code region for measurement with a start and end call. When the program finishes execution, the elapsed time, the amount of energy used (in Joules), and the average computed power (in Watts) of the kernel regions and the whole application are output.

B. RCRtool on Xeon Phi System

RCRtool collects power information of Intel Phi natively. Users can track power usage in microWatts through a file (/sys/class/micras/power) updated every 50 millisecond. RCRtool monitors the power at user level and computes the energy consumption over time. The information is available to the applications through the same simple API as on Sandy Bridge.

C. Polyhedral Optimizations Tools

The Polyhedral Compiler Collection (PoCC) was used to generate program variants with different optimizations. The PoCC requires that programs contain static

control parts (SCoP) so that valid transformations can be applied. Polybench is a collection of programs that contain SCoPs and can be polyhedral optimized.

III. Benchmarks for Energy Measurement and Tuning

In this work, we evaluate six representative Polybench programs energy auto-tuning with loop optimizations: 2mm, gemm, gramschmidt, covariance, seidel-2d, and jacobi-2d.

Using PoCC, program variants were generated using a different set of the optimizations from the following four groups:

- Loop fusion: smartfuse, maxfuse, nofuse
- Loop unrolling factor: 1, 2, 4, 8
- Loop tiling: 1, 16, 32, 64. Note that the number of different flags depends on the level of nested loops
- Loop vectorization: on, off

Loop parallelization was always turned on.

IV. Experimental Setup

The tests ran on a 2-socket quad-core Intel Xeon E5-2603 processor with 10MB (20MB total) L3 cache. PoCC v1.2 was used to generate program variants from Polybench v3.2. ICC v14.0.0 was the backend compiler. Every executable was compiled with -O3 optimization flag. Experiments were also run on a Xeon Phi coprocessor. The Phi architecture accelerator card contained 60 cores clocked at 1.053GHz. Each core had 512KB of L2 cache. The generated program variants used ICC v14.0.0 compiler as their backend, producing OpenMP programs that ran natively on the Phi.

V. Experimental Results

TODO:

VI. Related Work

TODO: do we need this at all?

VII. Conclusion

TODO:

VIII. Acknowledgments

This work is supported by the DOE XPress (DE-SC0008704) and the DoD ATPAR (PNNL-214990).

References

[1] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Auto-tuning for energy usage in scientific applications," in *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 178–187.

[2] A. Porterfield, R. Fowler, S. Bhalachandra, and W. Wang, "OpenMP and MPI Application Energy Measurement Variation," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, ser. E2SC '13, 2013, pp. 7:1–7:8.

[3] E. Park, J. Cavazos, and M. A. Alvarez, "Using graph-based program characterization for predictive modeling," in *CGO*, 2012, pp. 196–206.