Investigating Prefetch Potential on the Xeon Phi with Autotuning

Saami Rahman
Texas State University
San Marcos, TX
saami.rahman@txstate.edu

Ziliang Zong
Texas State University
San Marcos, TX
ziliang@txstate.edu

Apan Qasem
Texas State University
San Marcos, TX
apan@txstate.edu

ABSTRACT

This paper presents experimental results of tuning two software prefetching parameters in the Intel C Compiler for the relatively new many-core architecture Xeon Phi. Prefetching has been found to be effective in improving performance on this architecture, and in this work, we investigate the potential of tuning compile-time parameters to optimize performance and energy. We have found that on select programs, tuning can result in upto 1.47 speedup and 1.39 greenup.

1. INTRODUCTION

Prefetching is a well-known technique that is used to hide memory latency. Modern compilers analyze the program and insert prefetch instructions in the compiled binary. The Intel C Compiler (ICC) allows the programmer to specify two parameters that can help the compiler insert more accurate and timely prefetch instructions. The two parameters are `-opt-prefetch` and `-opt-prefetch-distance`. When unspecified, ICC uses default heuristics.

In this work, we present the results of autotuning the two mentioned parameters. The motivation for this stems from the understanding that most programmers choose to leave the default parameters unchanged, but significant improvements can be achieved by tuning these parameters. Tuning these parameters by hand can be challenging and time consuming as it requires knowledge of memory access patterns as well as significant time investment. We have developed a simple autotuning framework for the Xeon Phi architecture that automatically tunes these two parameters for any given program. We have chosen Xeon Phi as it is a relatively new architecture, and prefetching has been shown to be critical to performance for it [1, 2].

We have found few instances of work involving prefetching on the Xeon Phi, and none of them have attempted to tune the ICC compiler, but have rather focused on different techniques of prefetching and an analysis of its performance [3, 5]. We believe that tuning compiler parameters is a more fitting approach that will enable more programmers to benefit from performance and energy improvements, as it is more likely to fit within their development model.

Supercomputing 2015, Austin, Texas

2. EXPERIMENT DESIGN

2.1 Benchmark Description

We used 4 memory-intensive programs in our testing. The programs are written in C and parallelized using OpenMP. We used a matrix multiplication program that multiplies two square matrices of size 5K; a program that computes a 4001×4001 fractal from the Mandelbrot set with a pixel depth of 256; an implementation of rank sort that sorts 1.2M integers; and a topological implementation [4] of SSSP that uses the roadmap of New York City with 264,346 nodes and 733,846 edges as input.

2.2 Compiler Tuneables

1. `-opt-prefetch`: This option in ICC can take 5 values - 0 through 4. The compiler uses a default value of 3 when a program is compiled using flags -O2 or higher. A value of 0 means no prefetch instructions are added. Non-zero values indicate the amount of prefetching done.

2. `-opt-prefetch-distance`: This option takes two arguments, `n1` and `n2`, where `n2` is optional. The values of `n1` and `n2` represent the prefetch distance in terms of loop iterations. `n1` represents the prefetch distance for prefetches from memory to L2, where `n2` represents the same for prefetches from L2 to L1. `n1` must be greater than or equal to `n2`. If values for `n1` and `n2` are not specified, the compiler uses internal heuristics to compute prefetch distances.

The autotuning framework tunes the two parameters independently. When tuning the `-opt-prefetch` parameter, it builds the program using `-opt-prefetch` = [0 .. 4]. For the `-opt-prefetch-distance` parameter, the programs are built using all possible combinations of 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 for `n1` and `n2`, where `n1`>=`n2`.

The reported greenup and speedup values are computed using the baseline configuration, which is built using -O2 (ICC 13.1.3), and the values for `-opt-prefetch` and `-opt-prefetch-distance` determined by the compiler. Greenup and speedup are both metrics of improvements over the base configuration in terms of energy and performance respectively. Performance is measured by instrumenting the source code, and energy is measured using the Xeon Phi sensor.

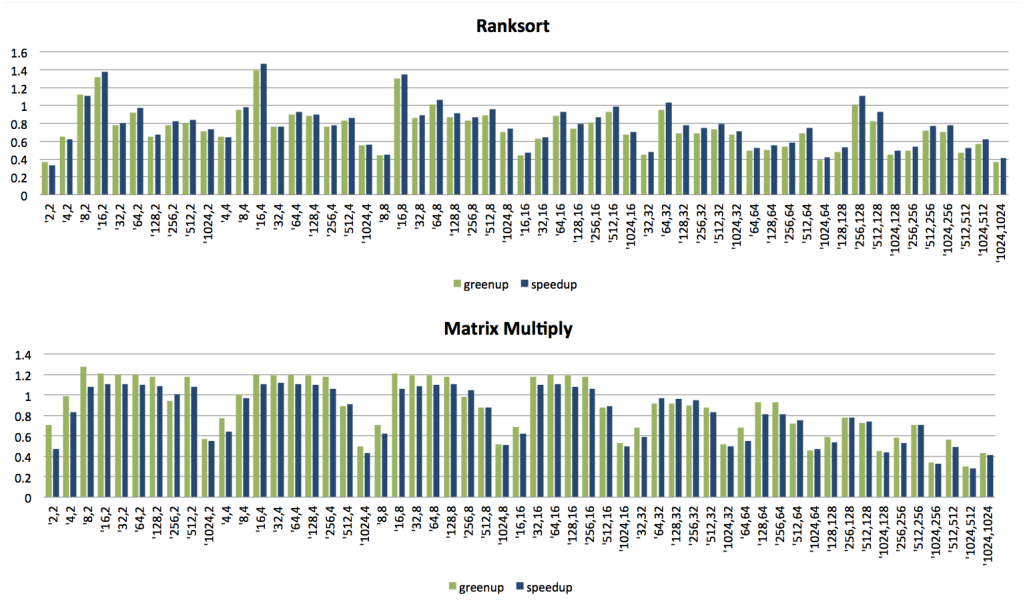

Figure 1: Effect of tuning -opt-prefetch-distance parameter on rank sort and matrix multiply

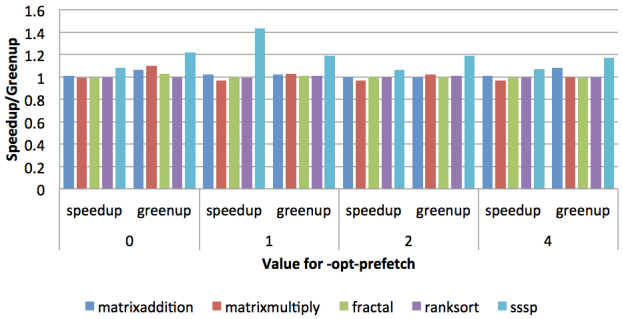

Figure 2: Effect of tuning -opt-prefetch parameter

3. RESULTS

From figure 2 it can be seen that only SSSP benefits significantly in terms of both performance and energy from tuning `-opt-prefetch`. However, when tuning `-opt-prefetch-distance`, the effects are more rapid for programs rank sort and matrix multiply. For rank sort, speedup of 1.47 and greenup of 1.39 are possible, and for matrix multiply speedup of 1.11 and greenup of 1.28 can be achieved.

There are several interesting observations to be made from figure 1. First, when `n1` is equal to `n2` or is much larger than `n2`, performance and energy suffer a large penalty. Second, smaller values of `n2` are better. As `n2` is the prefetch distance for prefetches from L2 to L1, and since the latency between L1 and L2 is relatively low, smaller values of it result in more timely prefetches. Third, performance and energy appear to be more sensitive to `n1`. These patterns collectively suggest that the space of feasible prefetch parameters can be reduced to a small set. Lastly, the best configuration for improving performance and energy are not always the same, as can be seen for matrix multiply, suggesting that they are independent optimization objectives. The graphs also show that improper tuning can dramatically hurt performance, in some cases up to 0.33, which underscores the effect of prefetching and that it should be tuned carefully.

4. CONCLUSIONS

We have used an autotuning framework to find the optimal parameters for improved performance and energy on Xeon Phi. Tuning can result in significant performance and energy improvements, and the optimal parameters for each of these objectives can be different. This indicates that they should be optimized individually. We have also found some repetitive patterns that suggest that the search space for the autotuning framework can be reduced.

5. REFERENCES

[1] R. Krishnaiyer. Compiler prefetching for the intel (r) xeon phi (tm) coprocessor, 2015. [Online; accessed 29-January-2015].

[2] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, and H. Saito. Compiler-based data prefetching and streaming non-temporal store generation for the intel (r) xeon phi (tm) coprocessor. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1575–1586, 2013.

[3] S. Mehta, Z. Fang, A. Zhai, and P.-C. Yew. Multi-stage coordinated prefetching for present-day processors. In *Proceedings of the 28th ACM international conference on Supercomputing*, pages 73–82, 2014.

[4] R. Nasre, M. Burtscher, and K. Pingali. Data-driven versus topology-driven irregular computations on gpus. In *Parallel & Distributed Processing, 2013 IEEE 27th International Symposium on*, pages 463–474, 2013.

[5] Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of intel's xeon phi processor. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 389–394, 2013.