Auto-tuning Sorting Algorithms

Sam White, Yeab Wondimu, Kyle Knapp

October 4, 2013

Abstract

The purpose of this project is to develop a system that can auto-tune the sorting of a given set of data. Auto-tuning involves empirically searching through a set of parameters such that a given set of data can be sorted as fast as possible. The parameters involved in this empirical search include the type of sorting algorithm used, the type of hardware used, the degree of parallelism, and the partitioning of data. The sorting algorithms that will be used include quicksort, merge sort, and bitonic sort. In terms of hardware,, the GPU, the network, the number of cores, and the size of the L1 and L2 cache will all factor into the empirical search. Parallelism will be tuned via controlling the number of threads processing the data. Then, based on the hardware being used and the amount of parallelism, an appropriate partitioning size will be determined. Once auto-tuned, the system will produce a set of parameters that can be input to the system such that data of similar type can be sorted as fast as possible without the need for further auto-tuning.

Motivation

Sorting algorithms constitute one the most performance-critical and well-researched domains in computer science. Automatic performance tuning, or auto-tuning, is a more emergent topic. Research into both general auto-tuning and domain-specific auto-tuning has grown in the last decade, with projects like ATLAS, FFTW, and SPIRAL tuning numerical algorithms for different architectures and memory hierarchies. We hope to take these methods and apply them to sorting algorithms in order to gain optimal performance for different types of input data and different types of systems.

Auto-tuning involves two basic steps: generation of parameterized code or various algorithms, and searching through these algorithms in order to find the best possible runtime performance. We will take into consideration parameters such as the randomness of the data to be sorted, the cache and memory resources of the machine being used, and the level of parallelism used. Our goal is to create a library that will provide users with a single API to a set of sorting

algorithms that are performance tested and tuned based on the input data and the underlying machine architecture.

Background

Research into auto-tuning domain-specific problems has been less extensive the further one gets from numerical linear algebra problems. We were only able to find two papers on auto-tuning sorting algorithms. Both were authored by Xiamong Li, Mari Jesus Garzaran, and David Padua.

Their first paper, "A Dynamically Tuned Sorting Library" [1] , focuses on examining the inputs to sorting algorithms and using an empirical search algorithm to figure out what combinations of parameters generate optimal performance. Specifically, the group focused on quicksort, multi-way merge sort and radix sort. They found that a cache-conscious radix sort gave them the best data locality and runtime performance. The multi-way merge sort was also highly tunable and adaptive. Quicksort, though, was harder to optimize, because of its inherently random memory access pattern.

More specifically, Li et al. tried to tune performance based on both architectural factors and characteristics of the input data. Architectural parameters included cache size, cache line size, and numbers of registers available to the CPU. The authors also preliminarily scanned the data being sorted to determine its size and standard deviation. Different sorting algorithms perform better on data with less of a distribution, while others are more agnostic to the input datas randomness.

Their second paper, "Optimizing Sorting with Genetic Algorithms" [2] , concentrates more on the Artificial Intelligence concepts used to speed up the empirical search. They use a genetic algorithm to search through the many combinations of parameters that could be used in order to adaptively partition the data and select the best sorting routines for each partition.

For each sorting algorithm, the authors use a library generator that searches through algorithm-specific parameters to find the optimal parameters for that search algorithm. Some of the examples of parameters that the library generator searches are: the value of the pivot used in the first phase of quicksort and the size and number of children in the heap used to merge the input array.

Work

Li, Garzaran, and Padua did not implement any parallel sorting algorithms, so we hope to in some ways extend their work by considering the optimal combinations of different levels of parallelism. We will tune our system based on parameters such as the randomness of the data to be sorted, the cache and memory resources of the machine being used, and the level of parallelism used. We hope to experimentally build up a knowledge base of tuning parameters and their effects on performance, so that eventually we can have a set formula for

handling different input data and hardware specifications. Our goals section details the work we hope to finish for each milestone.

Goals

Milestone 1 First, we will write code for all of the various sorting algorithms that will be used during auto-tuning. Also, we will ensure that they are fully tested and are functional. Secondly, a suite of functions will be implemented such that the sorting algorithms can be performed via threads, processes, GPU, or the network. It is important that all of these components are modular such that any of the sorting algorithms can be easily combined with any of the implementation methods. Yeab will primarily work with the implementation of the sorting algorithms. Sam will focus on writing code involving the GPU and MPI. Kyle will focus on writing code for making the threads and processes.

Milestone 2 For the second milestone, we plan to have our core system in place. First, this means we will be able to choose one of our algorithms and one of our implementations from the suite of functions and be able to successfully sort the data. When choosing, these functions we have should be able to pick important factors related to the function such as size of partitions and number of threads. Secondly, we will have created some expert system along with some AI that will help our system close in on the appropriate choice of functions and variables. This part of the system will use a combination of past experimental data and dynamic profiling to come to a decision. Yeab will work on the first part of this system. Kyle and Sam will work together on this second part.

Milestone 3 For the third milestone, the system will be completely finished. This entails that both functions and the expert system work together in harmony. We will be in the middle of writing our final paper and collecting data for the paper. This part of the project will be done together, and work will be appropriately allocated based on what needs to be completed.

References

[1] Xiaoming Li, María Jesús Garzarán, and David Padua. A dynamically tuned sorting library. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '04, pages 111–, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Xiaoming Li, Maria Jesus Garzaran, and David Padua. Optimizing sorting with genetic algorithms. In *Proceedings of the international symposium on Code generation and optimization*, CGO '05, pages 99–110, Washington, DC, USA, 2005. IEEE Computer Society.