K Nearest Neighbors

Abby Malkey

July 28, 2014

1 Introduction

STAPL (Standard Template Adaptive Parallel Library) is the parallel version of the C++ STL, created in Parasol Lab at Texas A&M University. Parallel computing involves distributing the work required to run a program over several processors to decrease the run time. Test cases are created to evaluate the performance of basic algorithms and demonstrate their usability. This particular test case calculates the k closest points for a vector of three dimensional points of floating-point numbers and a variable number k given a target point.

2 Design

The general design of the code is as follows:

- create and fill vector/view (parse command line arguments for vector size, k, and target point
- create map and iterator to hold and parse results
- compare every point in the vector with the target point, calculating the distance between them
- put the results in a map with distance as the key, taking only the first k elements
- output results

3 STAPL Components

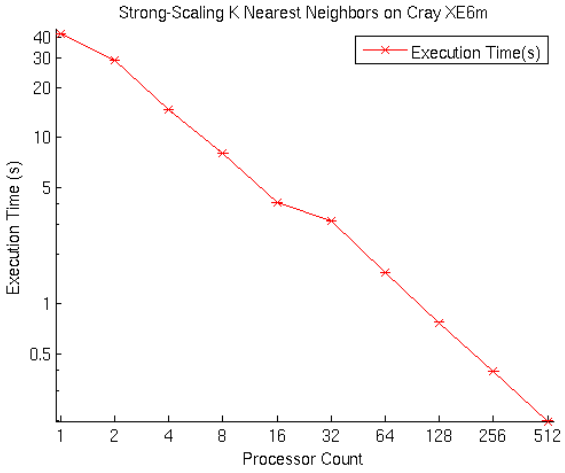
The following parts of the STAPL library were used:
vector/vector view: store elements of data set to be processed

- map_reduce: applies multiple work functions to the elements of multiple views and reduces to a single answer to be assigned to a variable

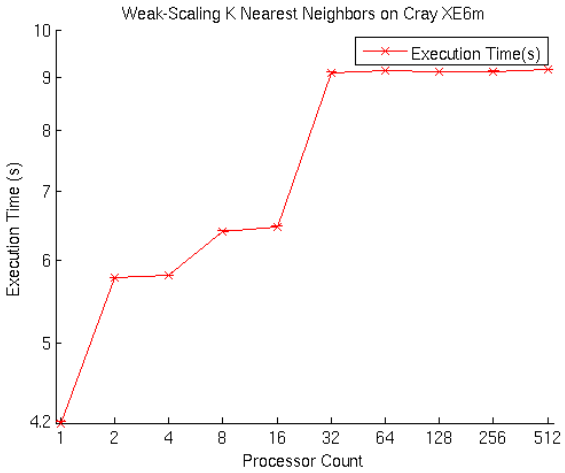
- do_once: performs action on only one location
- proxy specialization for point struct: allows an object to be referred to across nodes
- random_sequence: generates random doubles to fill point vector

4 Results

The experiments were run on the Rain system, a Cray XE6m supercomputer with 576 cores at Texas A&M. Strong-scaling and weak-scaling trials were performed. The difference between strong and weak scalabilty refers to the amount of work the processor must perform. Strong-scaling is when a constant amount of work is loaded into an increasing number of processors, so each processor has an increasingly smaller workload. Weak-scaling invloves increasing the workload and the processor count at the same rate. The graphs below show the mean execution times of 32 samples with respect to the core count and 95 percent confidence intervals.



(a) Strong-scaling



(b) Weak-scaling

5 Analysis

The anticipated outcomes were for the times to decrease dramatically for strong-scaling and hold steady for weak-scaling. For strong-scaling, the execution time drops quickly as the processor count is increased since the work is being split among the processors. The weak-scaling results were as predicted, varying only slightly, except for a few small jumps. When going from 4 to 8 cores, the jump is due to the memory bandwidth being completely utilized, as all 8 cores were placed by the batch system onto a single die that has limited bandwidth to the memory on the compute node. The jump between 16 and 32 cores can be attributed to the processes running on two nodes of the system instead of

one. The increase from one to two cores is a bit higher than expected, but it is because the program is now running on two cores and communication is occurring, whereas no communication occurs when running on only one core. The confidence intervals were small enough to suggest that the execution time of the algorithm on a given number of processors is stable.

6 Conclusion

In conclusion, relatively simple algorithms can easily be written in STAPL, which speeds up the execution time by distributing the work over a chosen number of processors.