PA2 Report – Christopher Westerman

Algorithm Description

The Sieve of Eratosthenes algorithm aims to find all prime numbers between 2 and some value N. When first approaching sieve, the brute force algorithm sequentially marks off multiples of a prime number then moves to the next unmarked value until it reaches N. Immediately, the algorithm benefits from removing even elements in the array and changing the sequential marking to moving by a stride. The algorithm can further be improved by taking advantage of cache sizes and marking off values in chunks.

Parallelization Approach:

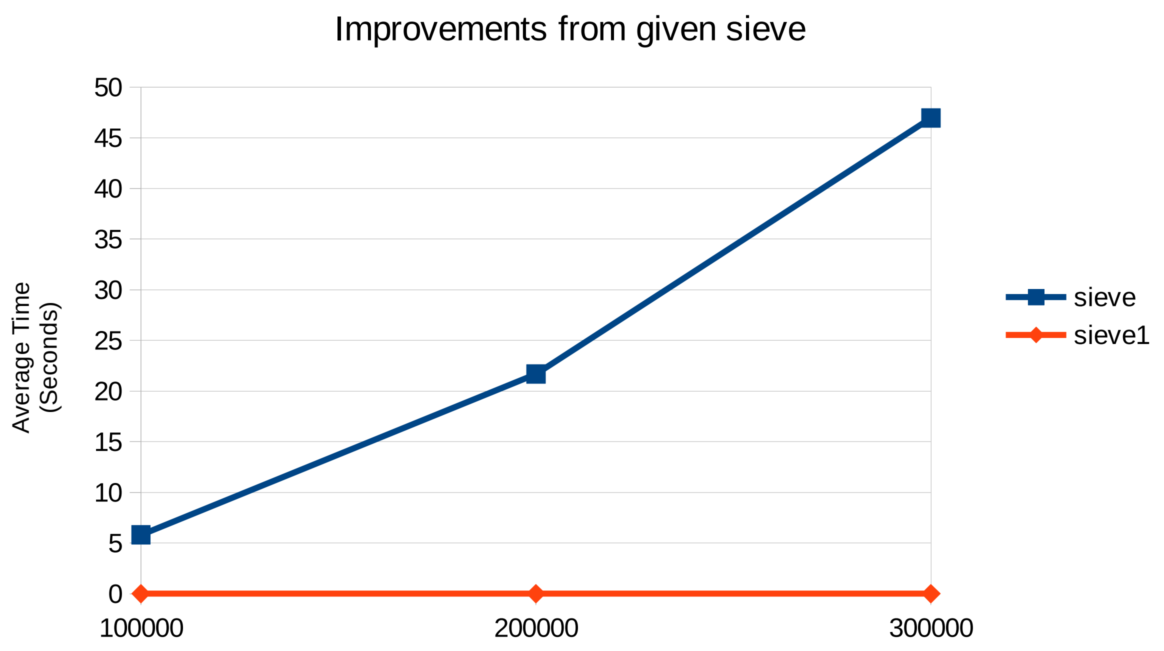
It becomes difficult structurally to parallelize sieve1 due to the dependent nature of the outer loop so the only alternative is to run the inner loop in parallel. Sieve3, by using the independent blocking paradigm, we can run each block in parallel in addition to the previous.

System Specs:

Name:	frankfort@cs.colostate.edu
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	20480K

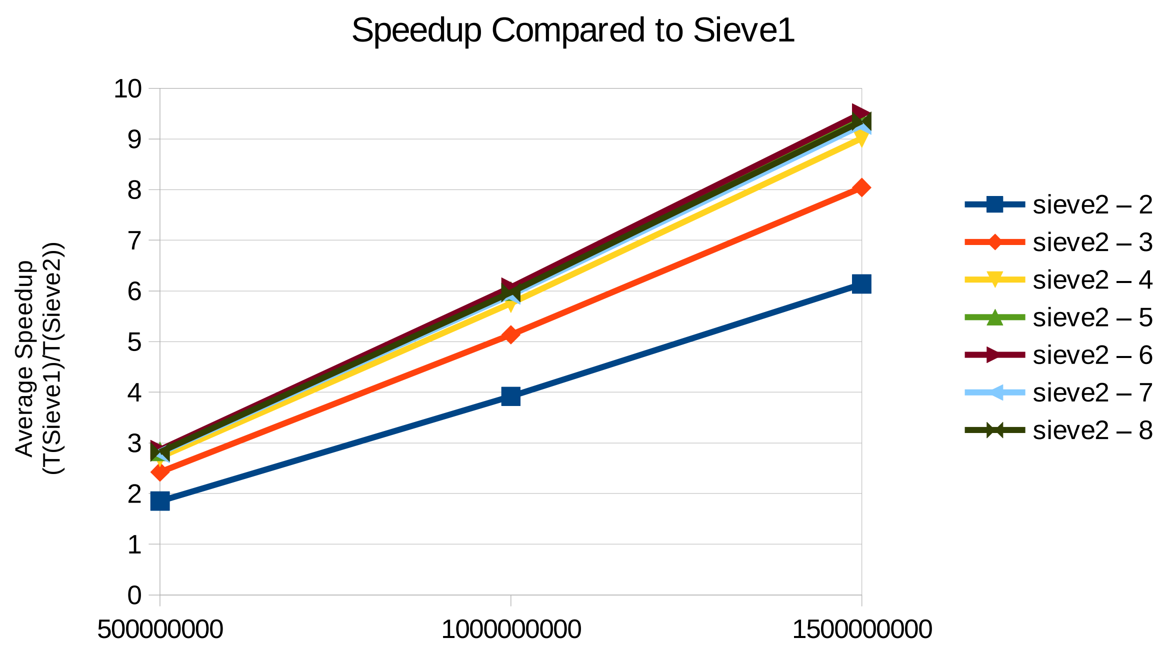
[Sieve1]

There was a great improvement from sieve.c to sieve1.c to the point where the plot shows sieve1 consistantly below 1 sec even for big values of N.



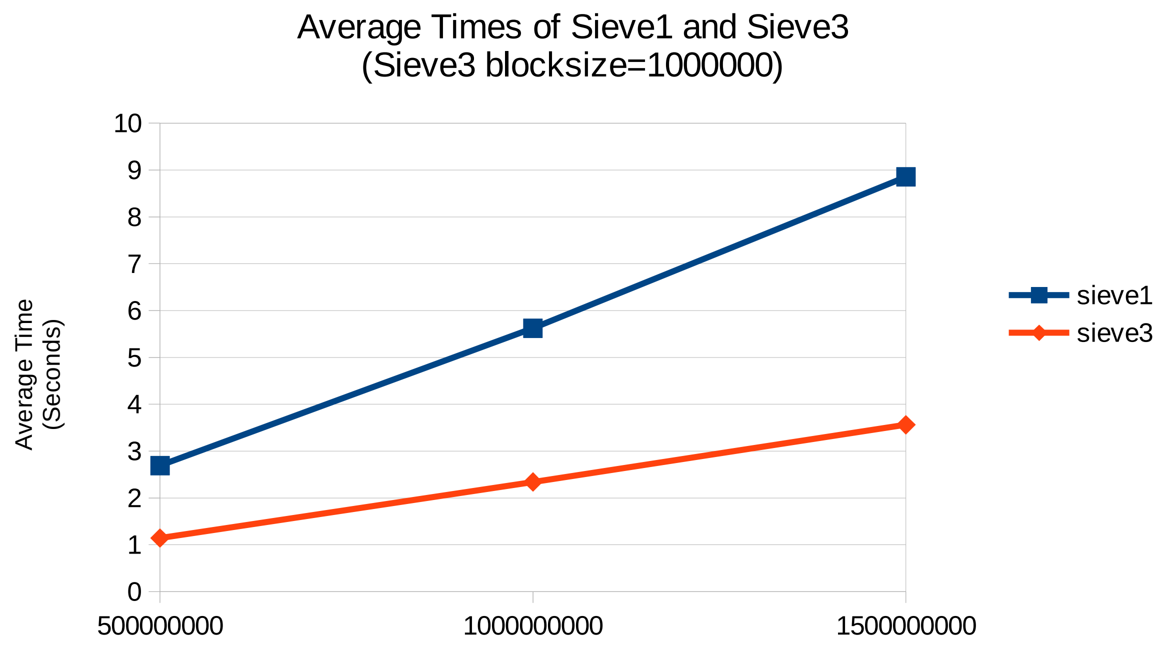
[Sieve 2]

Overall, we can see a pretty linear speedup from sieve1.c to the parallelized code in sieve2.c although it tapers off heavily as the number of cores increases. Since each task is limited to running on small sections of redundant code, this bottlenecks threading preformance.



[Sieve 3]

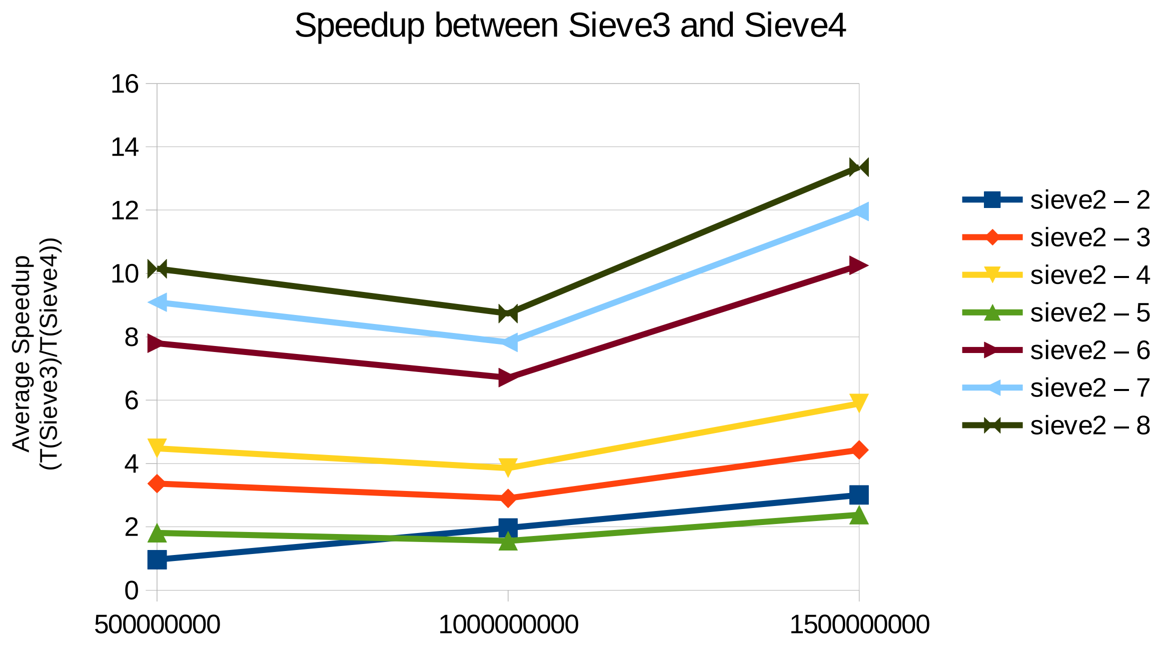
Blocking dramatically increased performance due to keeping values in cache for longer than the non-blocked sieve1.c.



[Sieve 4]

Even though the graph looks strange, we can easily see a linear speedup as the number of threads increases. Due to the blocking nature of the sieve3.c algorithm, each thread can easily work in parallel without being bottlenecked by other threads.

There is a strange dip when N = 1000000000 which I would assume is due to the square root not being an easy to work with value.


Speedup between Sieve3 and Sieve4
Average Speedup (T(Sieve3)/T(Sieve4))
sieve2 – 2
sieve2 – 3
sieve2 – 4
sieve2 – 5
sieve2 – 6
sieve2 – 7
sieve2 – 8