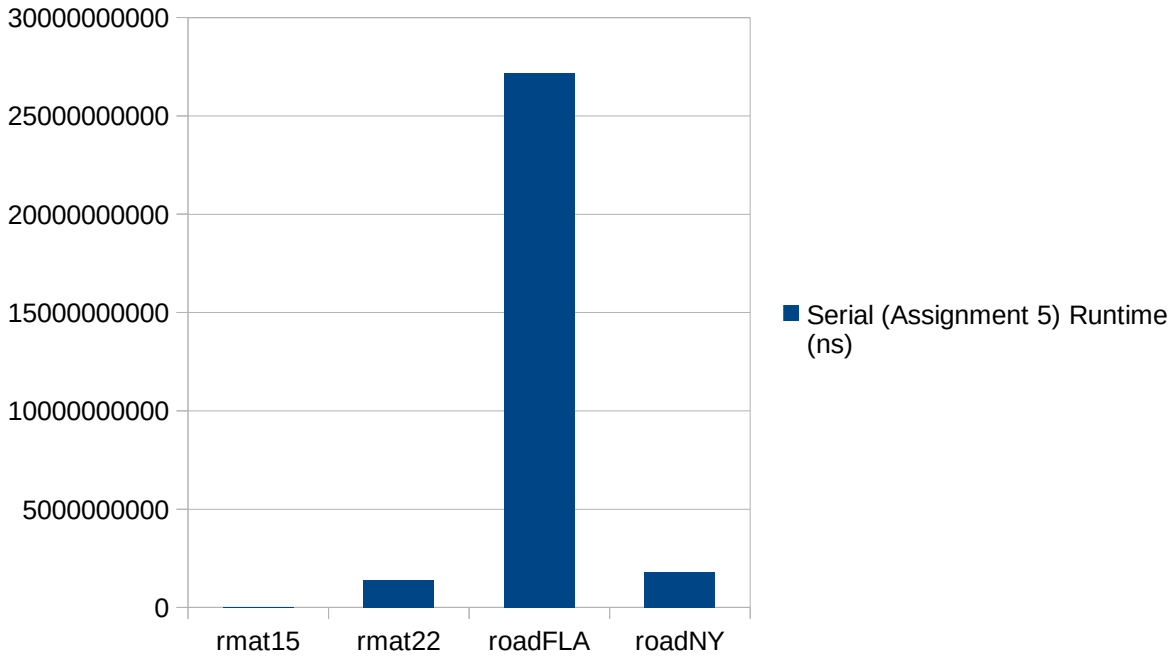
Savannah Smith – ses3964
Manasa Tipparam – mt32855

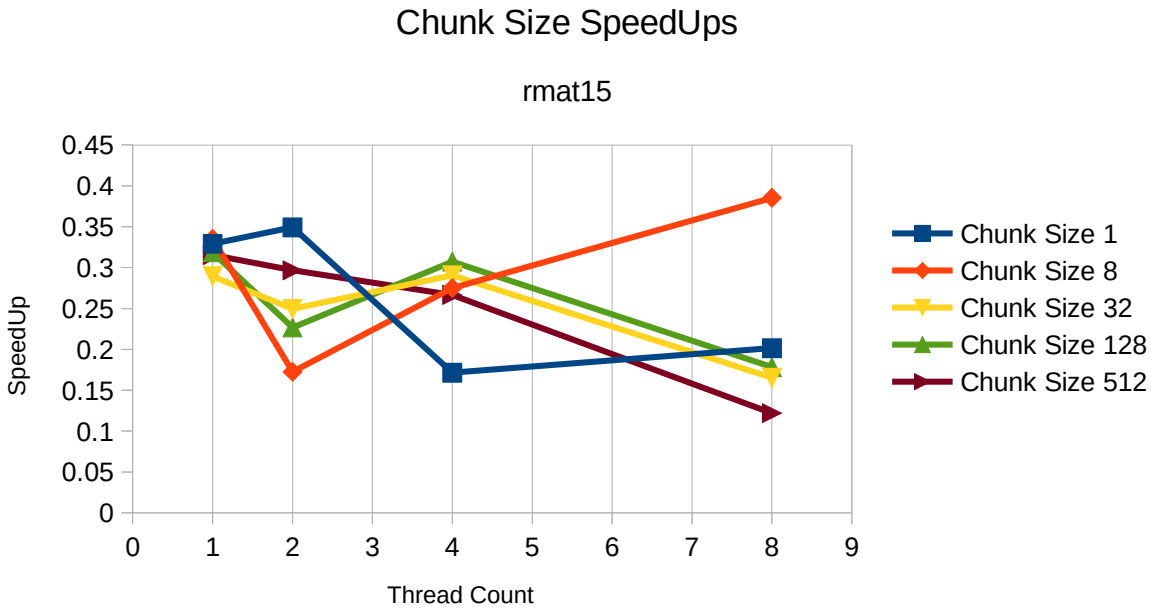
---Serial Runtimes---

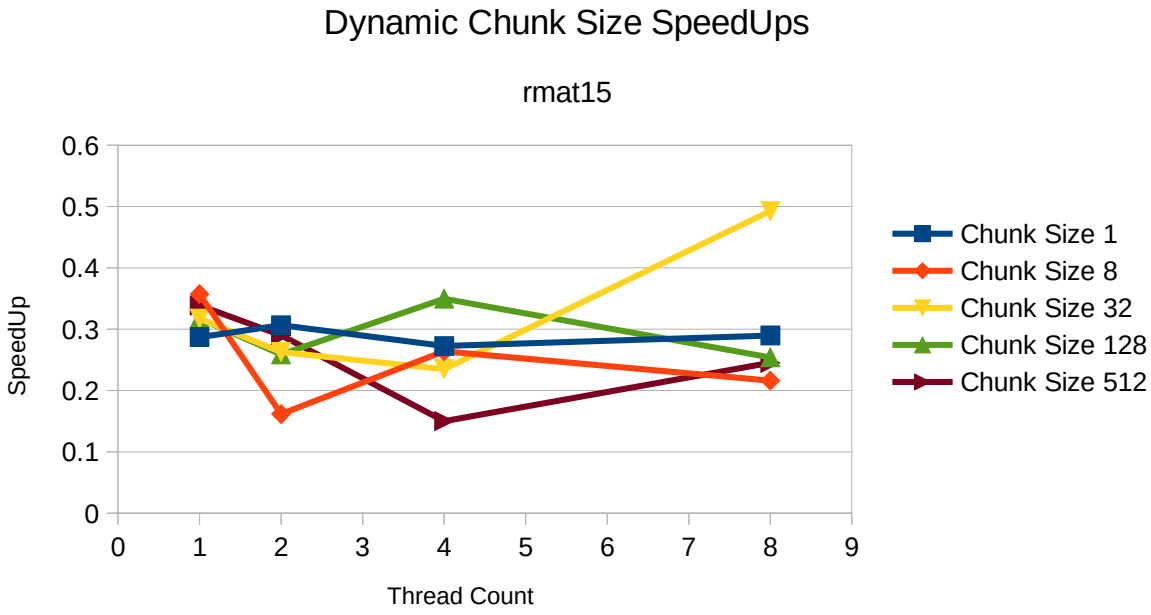


rmat15 has a very low runtime. Too low to see.

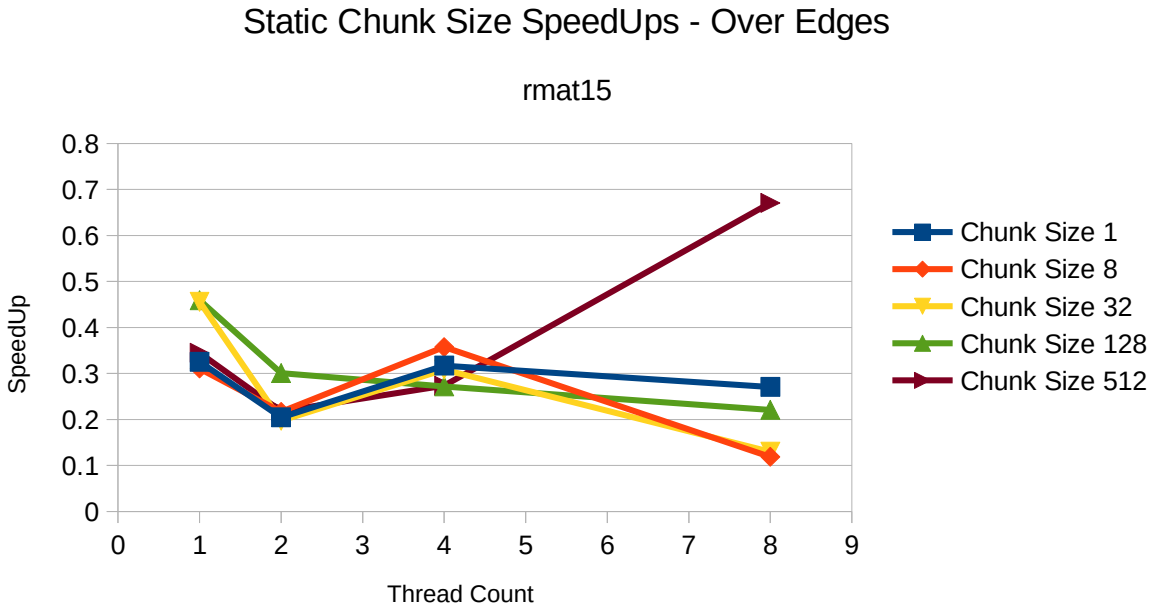
Savannah Smith – ses3964
Manasa Tipparam – mt32855

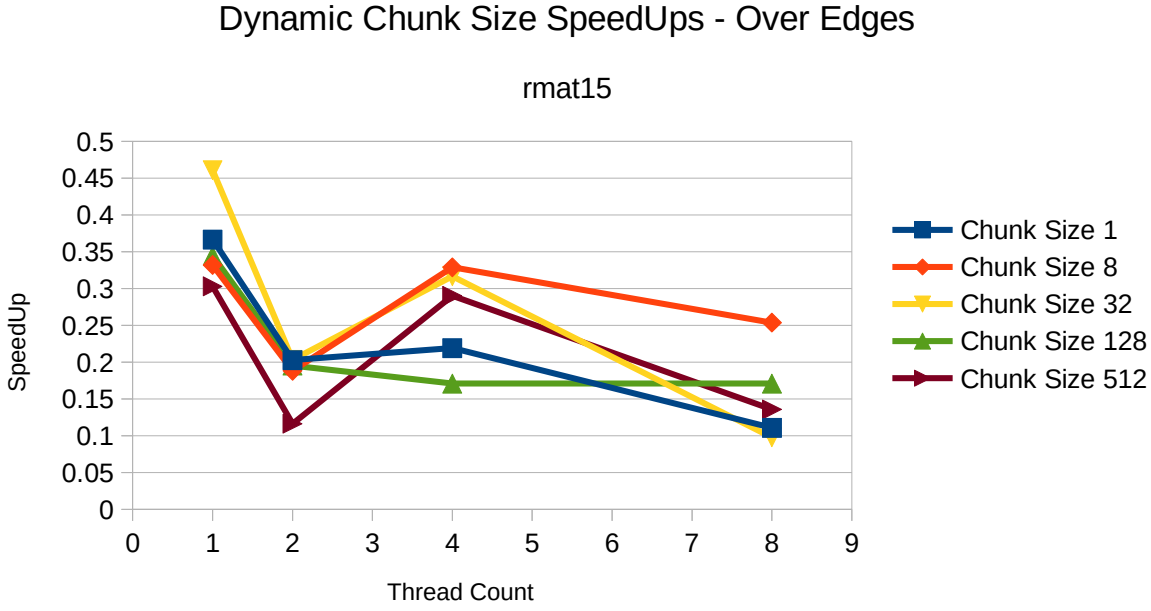
---RMAT15 Graphs---





Savannah Smith – ses3964
Manasa Tipparam – mt32855



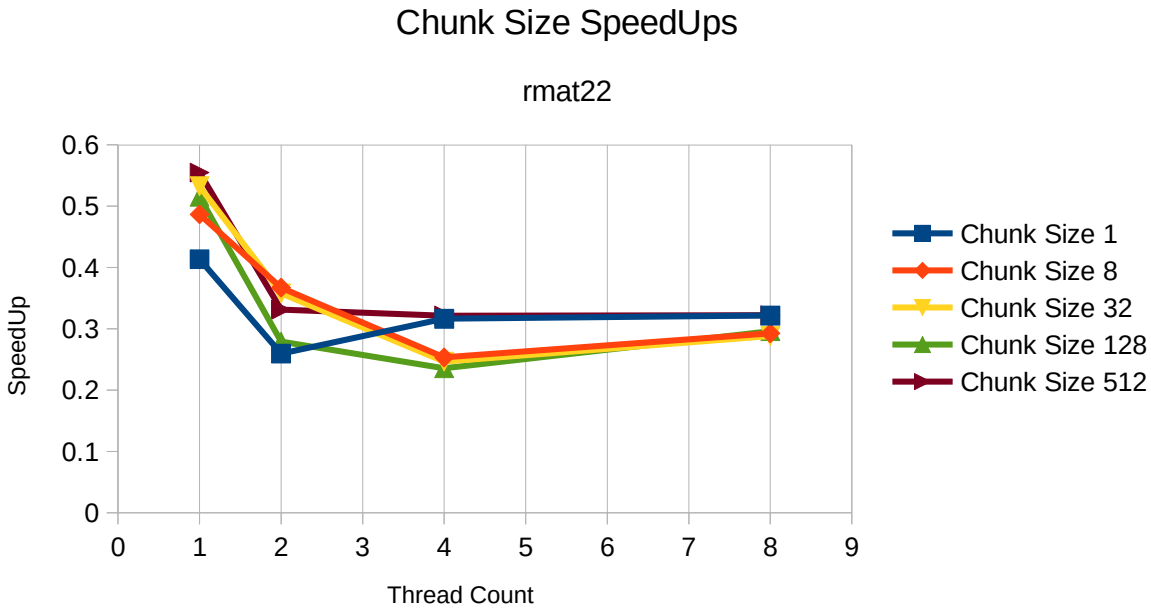


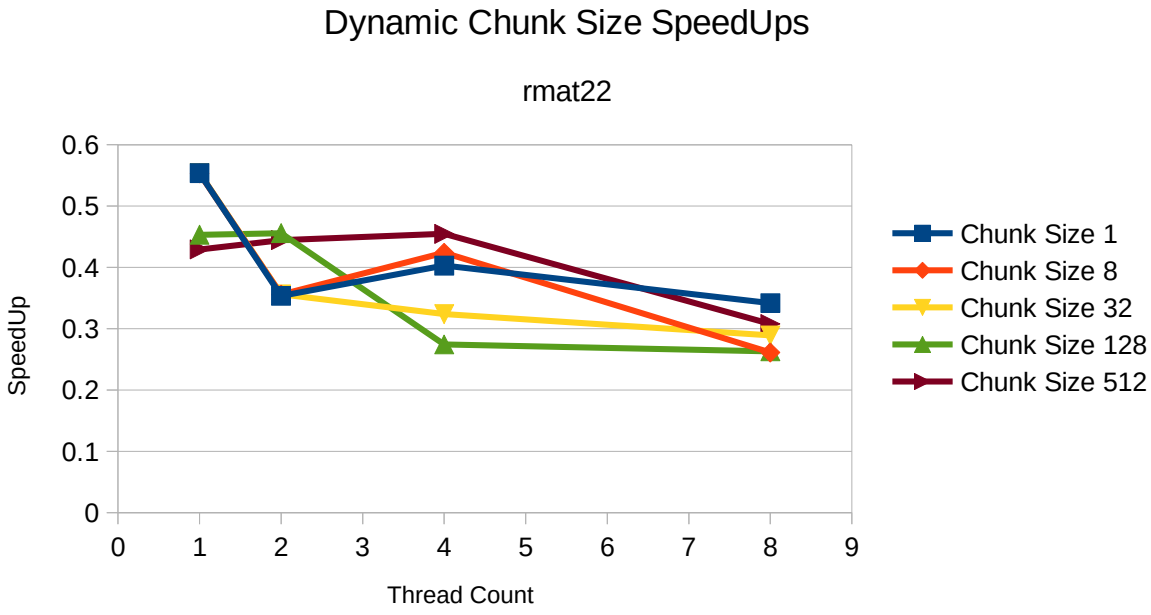
In none of the rmat15 graphs do we see any speed up. As the number of thread size increases performance typically continues to decrease although there are exceptions in static-8, dynamic-32, and static-512. I believe with rmat15 it isn't that helpful to thread our code because our code is written in a way that has a lot of true/false sharing. With the exceptions I believe that the chunk sizes in those situations allow for the situation where the dependencies do not cause as many conflicts and thus allowed for greater performance than other chunk sizes. In this case I believe dynamic is too hurtful during runtime to be useful. There is the example of dynamic-32 that had the best performance, but typically I believe this process does not benefit from a dynamic scheduling of chunk iterations, and it performance wise except for dynamic-32 on average does much worse. This graph does not seem to

benefit from the over edges approach to the algorithm. This shows that in this graph load balancing is not a problem, meaning that this is not a power law graph or that the true/false sharing is wasting enough time that it the sharing does not matter. A power law graph would typically benefit from dividing by edges rather than nodes as that would divide up the number of operations more equally between threads. This is most beneficial to power law graphs as uniform degree ones would end up naturally dividing the edges fairly equally on their own.

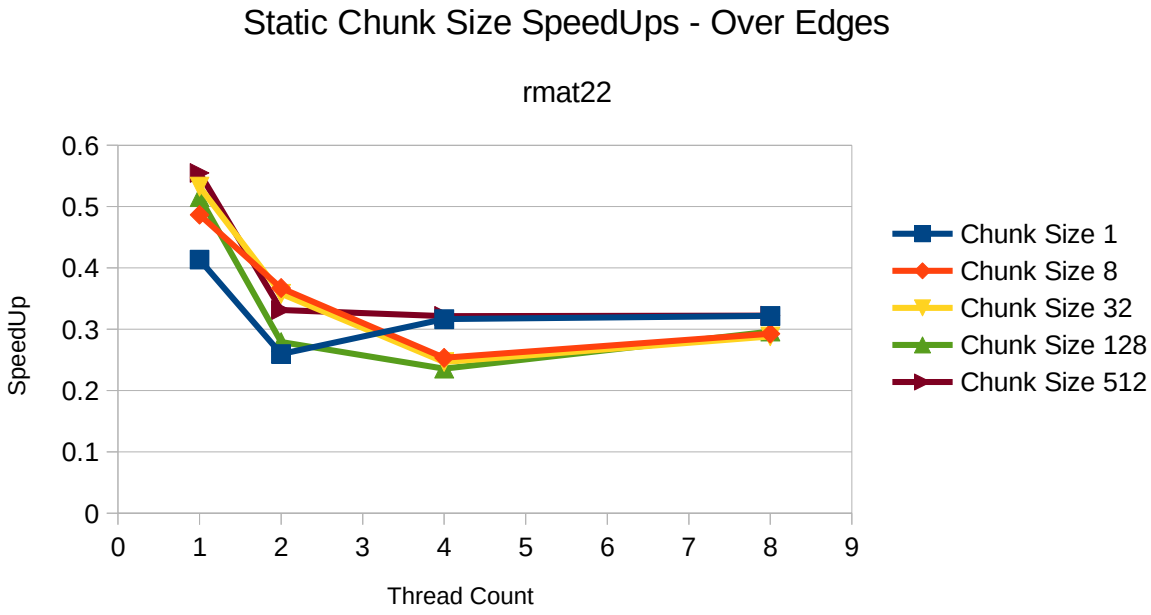
Savannah Smith – ses3964
Manasa Tipparam – mt32855

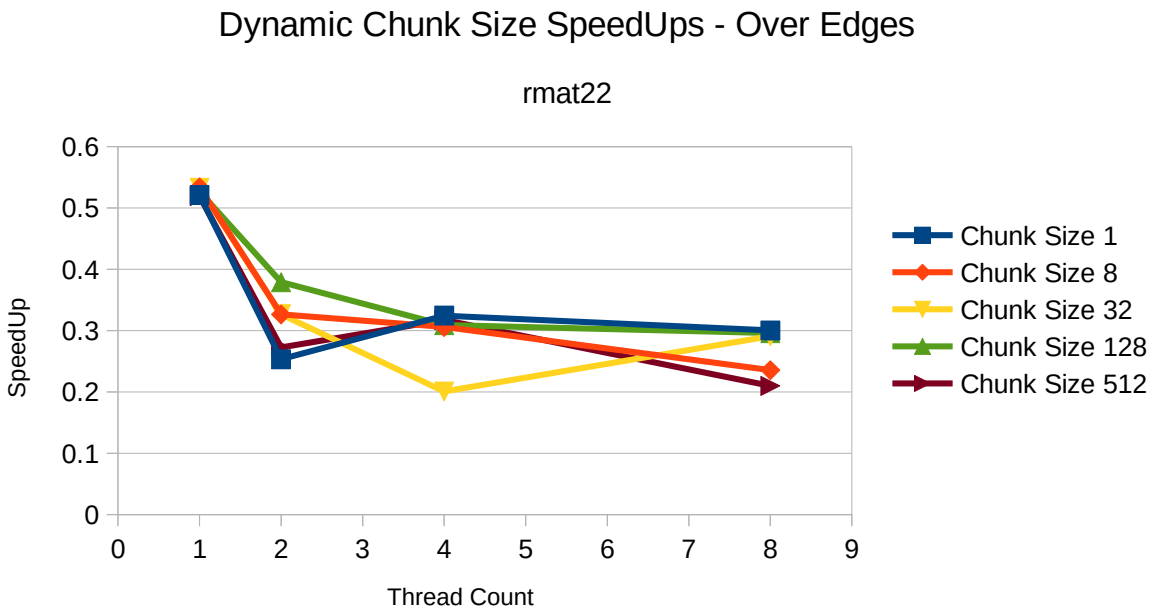
---RMAT22 Graphs---





Savannah Smith – ses3964
Manasa Tipparam – mt32855

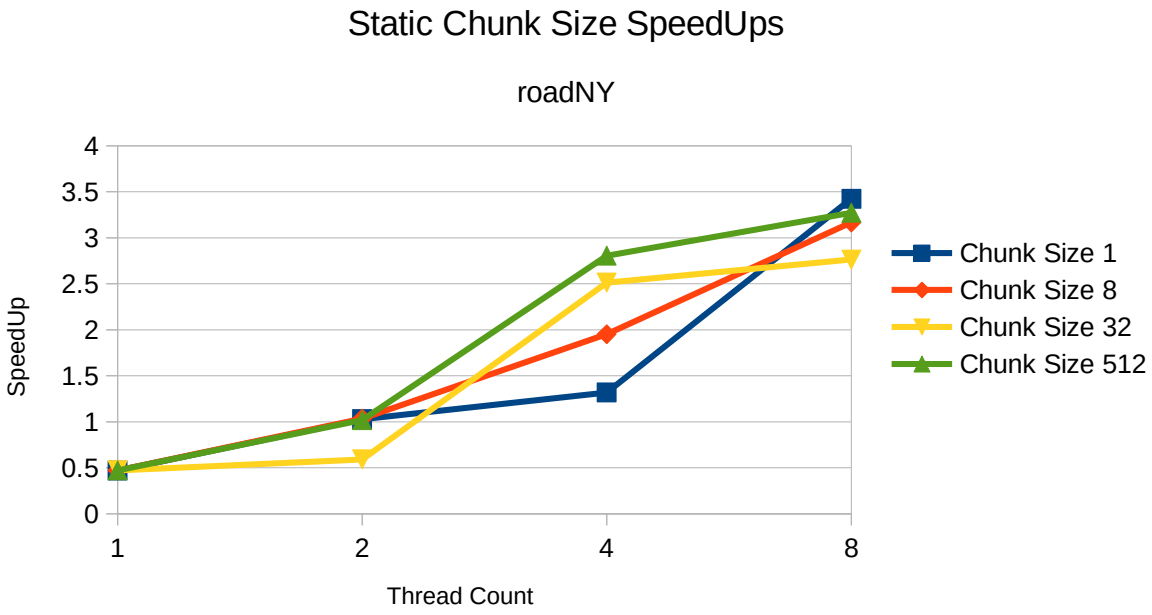


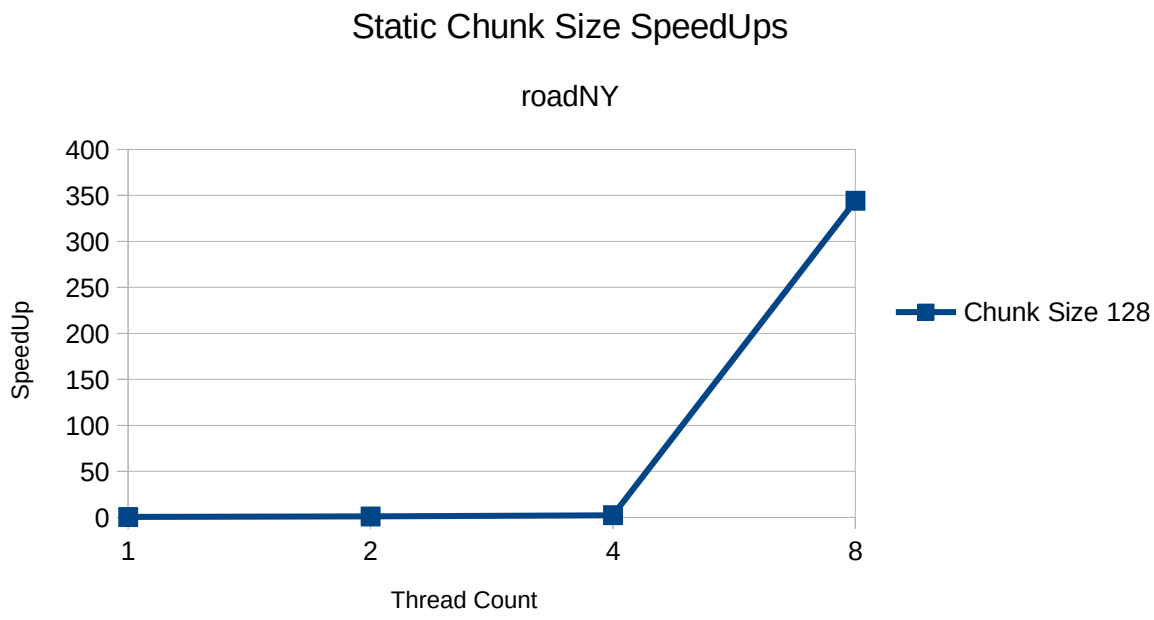


Again in the rmat22 which by the similar names and behavior, I assume are structured the same we see no actual speedup. Again I blame true/false sharing for this behavior. There are too many accesses to the same data (or same line) causing dividing the program into threads to not be helpful. In this program we do see performance increase using dynamic programming when dividing over nodes. This time it could be due to size of this program causing the dynamic operation to be helpful it dealing out chunks. Over nodes this program could have a lot of difference of when an chunk would finish, so dynamic is helpful dealing with this type of graph. Other than dynamically and over nodes we see very similar behavior between all of the variants.

Savannah Smith – ses3964
Manasa Tipparam – mt32855

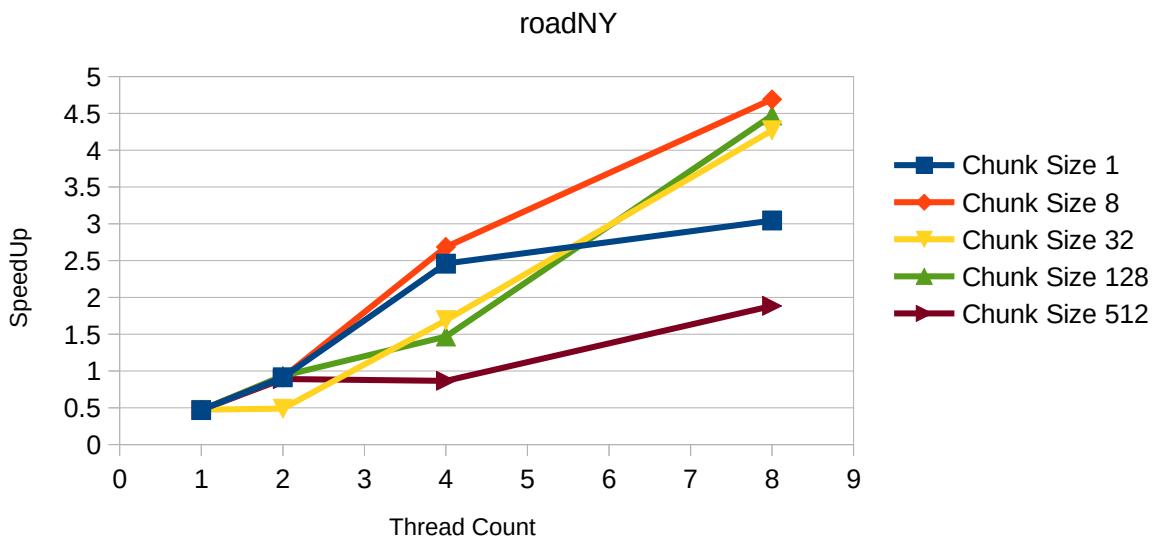
---RoadNY Graphs---



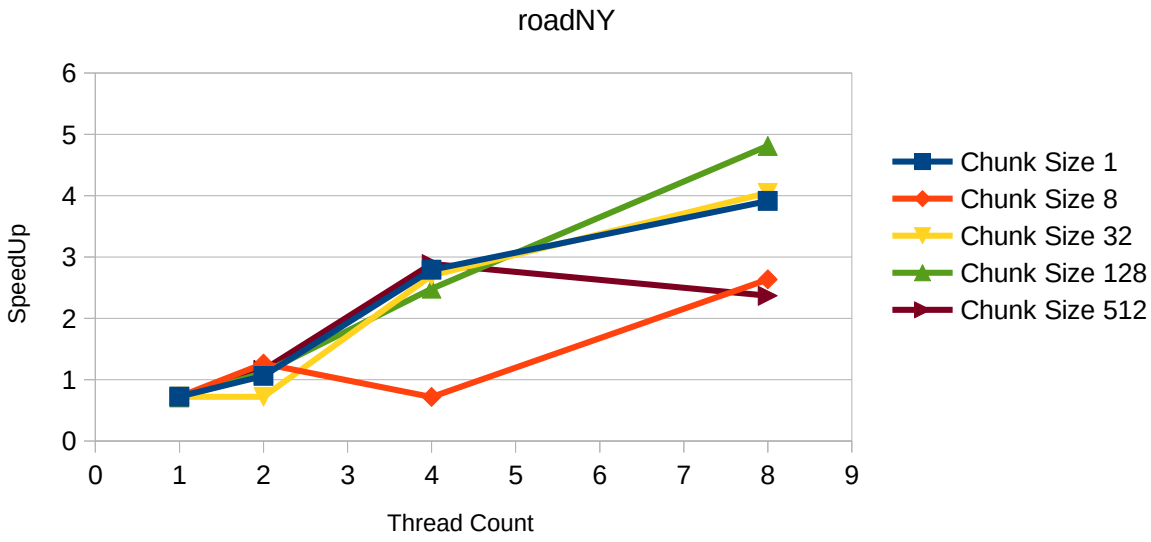


Savannah Smith – ses3964
Manasa Tipparam – mt32855

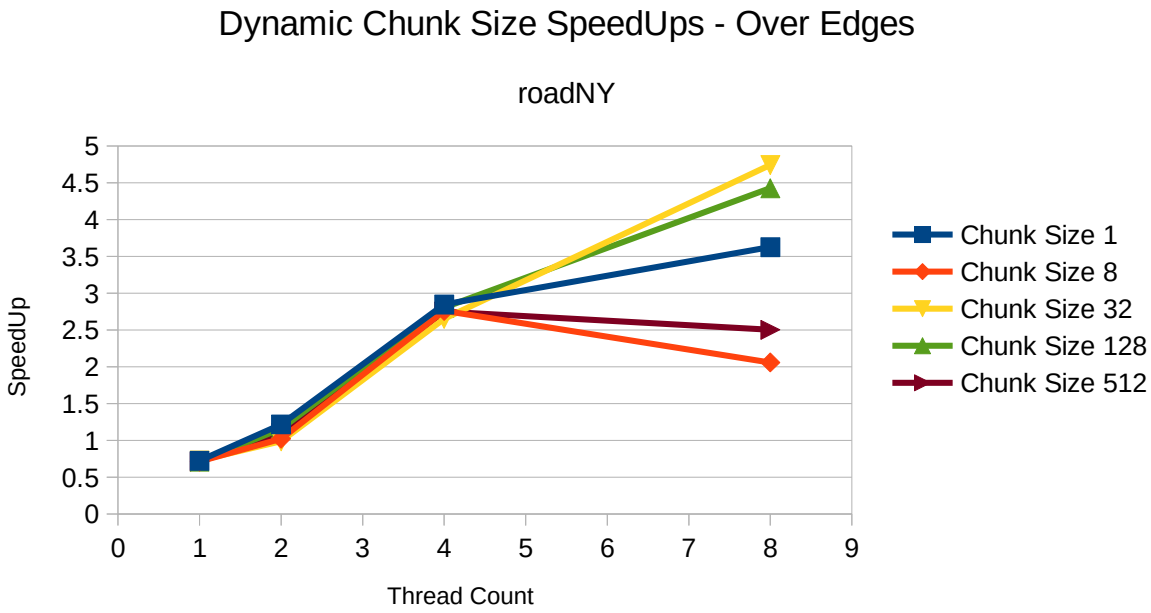
Dynamic Chunk Size SpeedUp



Static Chunk Size SpeedUps - Over Edges



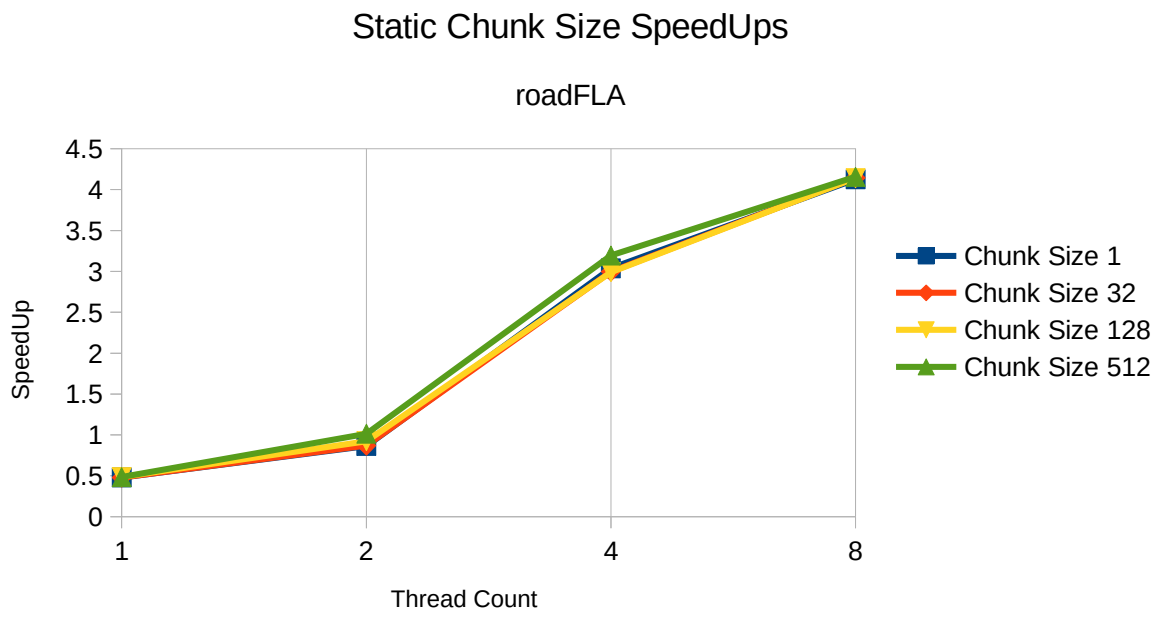
Savannah Smith – ses3964
Manasa Tipparam – mt32855

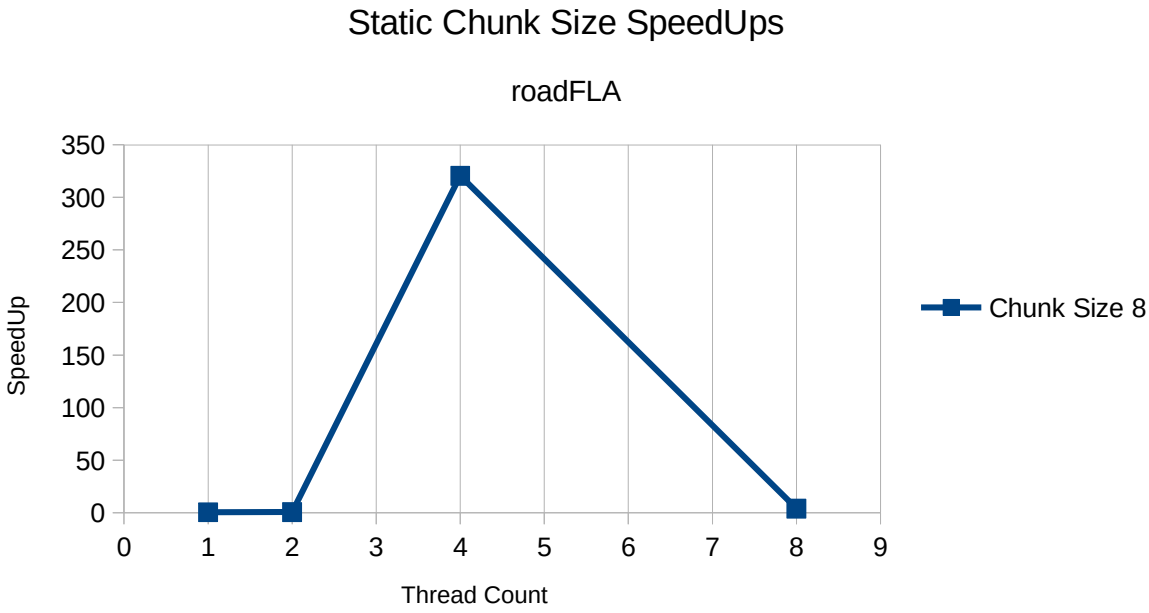


Now we see where threads are very beneficial! We even see a graph that I had to put by itself, because it blew the other variant's speed ups out of the water. These graphs are not as bothered by true/false sharing. So now as the threads aren't running into each other accessing the same data there can be a benefit in threading the program. We also see how over edges is beneficial as well, although we don't get the incredible speed up of ~300 with over edges. Over edges works well here as again distributing the workload more evenly is beneficial as all threads can take the same amount of time to finish. However I believe we don't get an astounding increase because dividing the edges equally does lead to true/false sharing again. Normally this is fine but for some reason it denied the incredible speedup that luckily occurred with static-128 sharing. Dynamic made a difference in the over node version as again it helped rectify an imbalanced load.

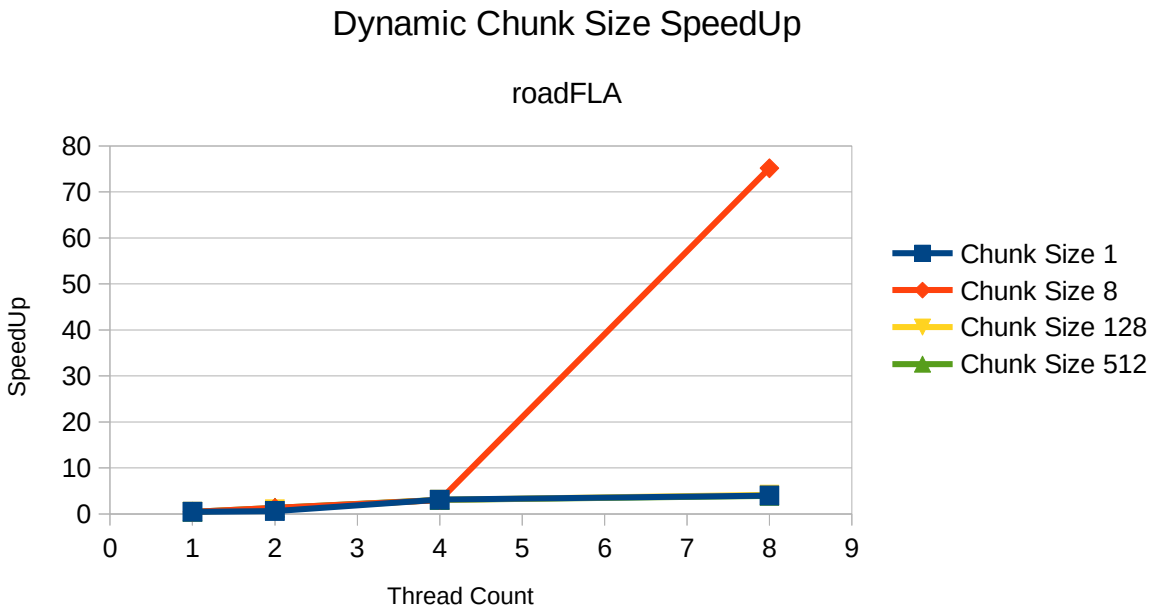
Savannah Smith – ses3964
Manasa Tipparam – mt32855

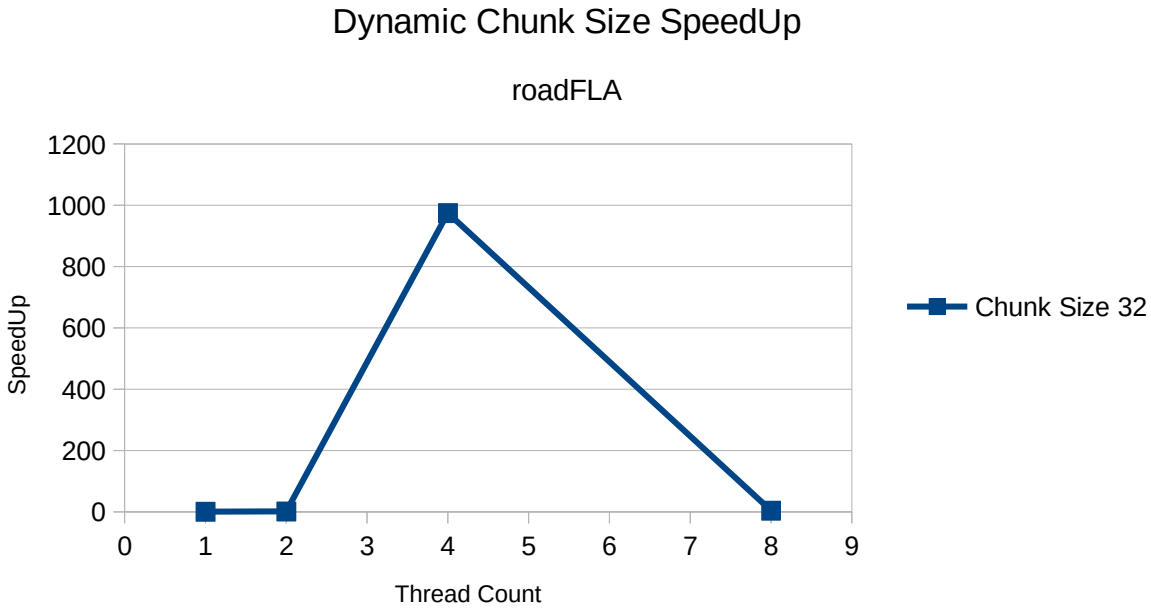
---RoadFLA Graphs---



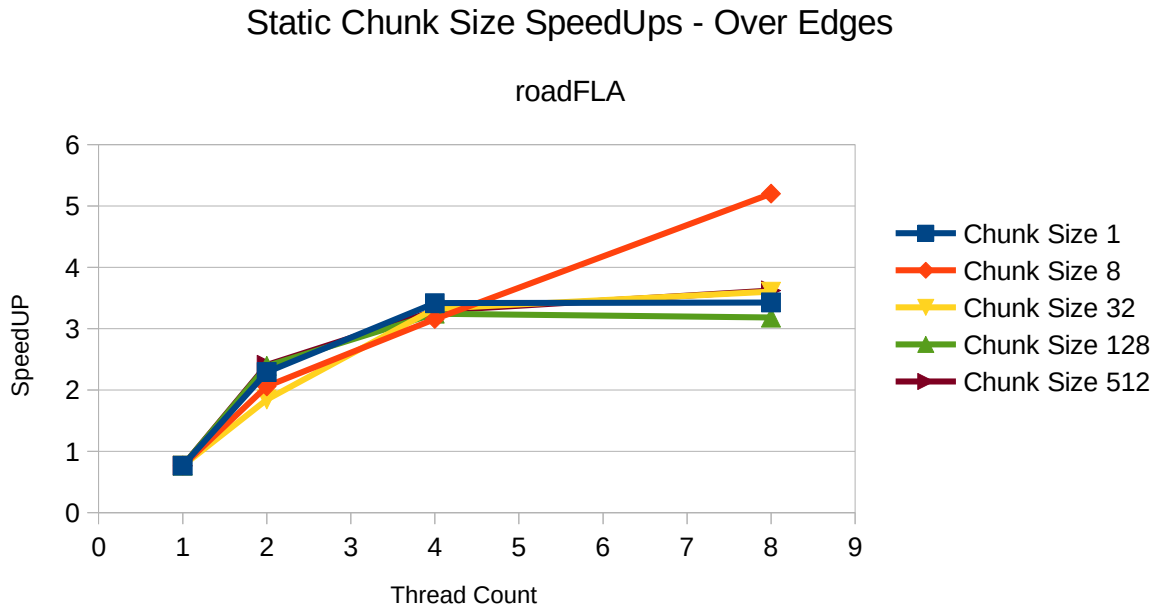


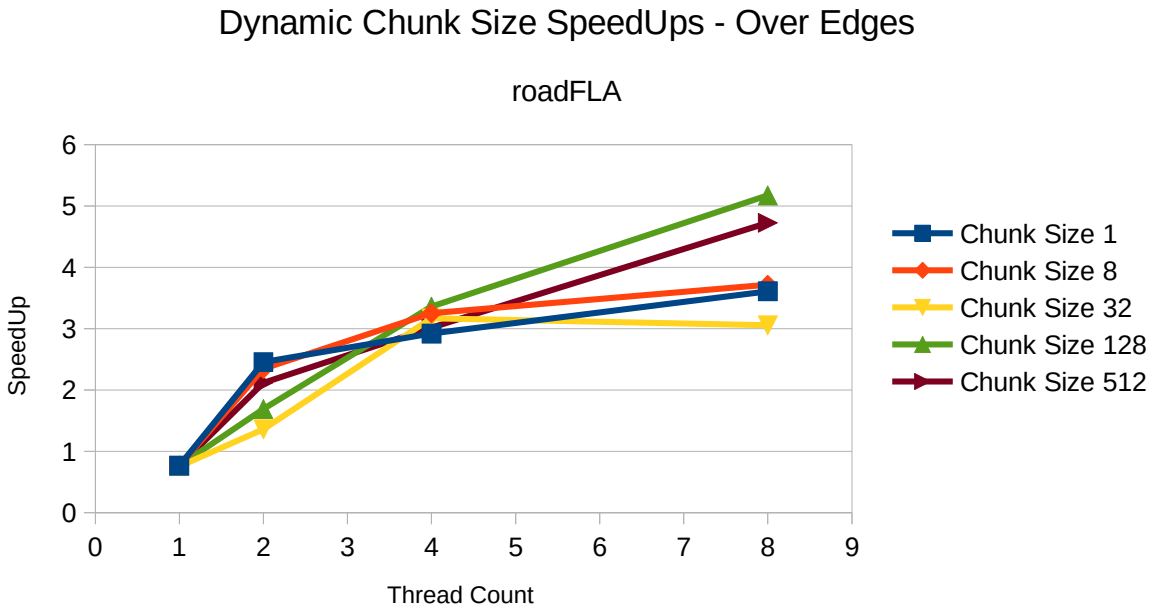
Savannah Smith – ses3964
Manasa Tipparam – mt32855





Savannah Smith – ses3964
Manasa Tipparam – mt32855





Again we see great speed up with threads in these versions. Over nodes we some incredible scenarios with speed ups such as ~75, ~300, and ~900!! We do not see that in over edges. Over edges is more fair but then it limits scenarios where there are no false/true sharing. With over nodes plus having the right selection of chunk size the processes can get lucky enough to not interfere with each other when accessing data.

Savannah Smith – ses3964
Manasa Tipparam – mt32855

In conclusion it seems that over edges is a more fair way to look at these nodes. It is the safer way to guarantee speed ups, but it also increases sharing and thus prevents the massive speedups we get when the right combination leads to very little interference in threads. These giant speed ups is why experimenting with chunk sizes is important to find what works the best for the size and organization of data.