# Apriori

For assignment four I decided to implement the Apriori algorithm in sequence, and then parallelize it using OpenMP. I chose OpenMP because while the problem is somewhat recursive in nature, I found it easier to write iteratively, ie I was originally going to write the implementation in Cilk, but later decided not to.

## The Algorithm

The apriori algorithm is used to find links between items in a transactional database. For example, say you had a database to keep track of items purchased by customers. It would be useful to find the items that are most purchased together (ie customers who buy bread usually buy milk, as well). We could keep track of this relation with the items SKUs.

When running the Apriori algorithm on the database, we must specify a support threshold, ie, in how many transactions must a link appear for us to want to see it? This is useful because in many scenarios there are many outliers. For example, there could be one person who bought a mop and a gatorade, most people would not buy this combination.

Say our database looked something like this

| **Itemsets** |
|---|
| {1,2,3,4} |
| {1,2,3} |
| {3,4,5} |
| {6} |

Where each item in the itemset represents a SKU. From there we could check how often combinations are bought together, starting with the items purchased on their own:

| Sku | Support |
|---|---|
| {1} | 2 (SKU 1 appears in 2 different transactions) |
| {2} | 2 |
| {3} | 3 |
| {4} | 2 |
| {5} | 1 |
| {6} | 1 |

Once we establish that list then we will then check combinations
{1,2},{1,3},{1,4},{1,5},{1,6},{2,3},{2,4} .. and so on

{1,2} appears in 2 sets
{1,5} never appears (these items are never bought together)
{2,3} appears in 2 as well,
etc

we keep doing this process until no more combinations meet the support threshold.

More information can be found at http://en.wikipedia.org/wiki/Apriori_algorithm

## System Created

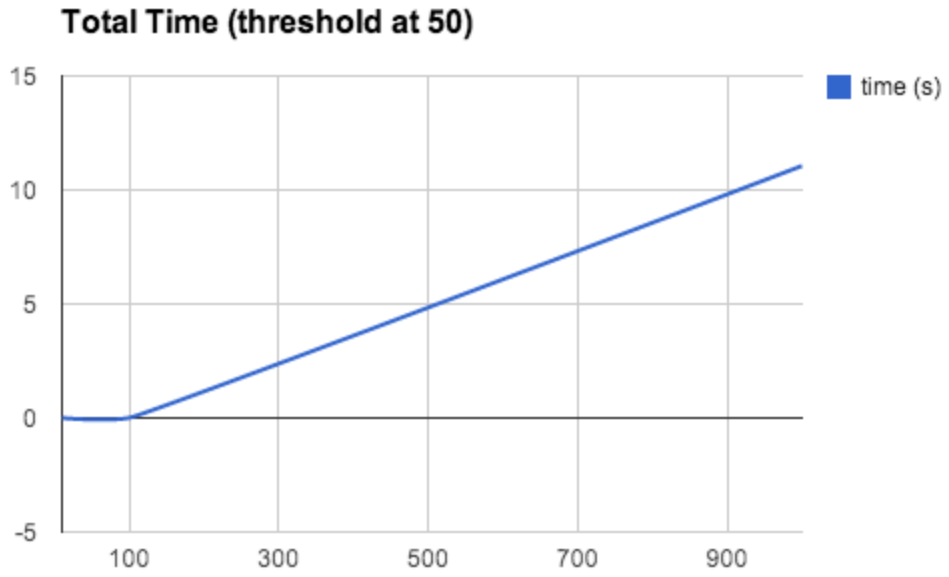
As mentioned above, the system created uses OpenMP to parallelize the task of check the support of itemset combinations versus the transactions. For ease of understanding and development, the transactional database is represented as a C struct, and is randomly generated at runtime. Items in the database are integers. The number of transactions generated can be specified as a command line options.

The support threshold that the user algorithm will look for is also specified at run time. Once the program is running, it will print out itemsets and itemset combinations that meet the threshold as they are found.

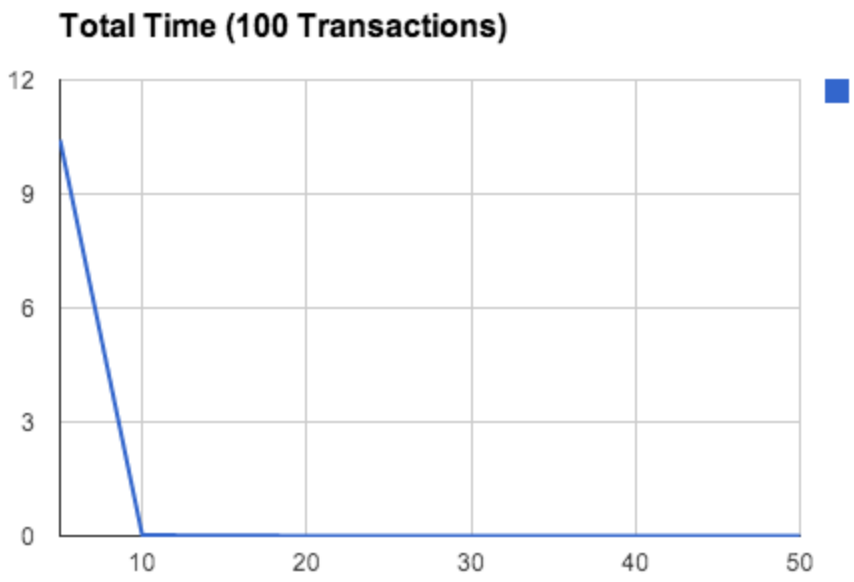
# Speedup

Time taken to run the algorithm varies greatly depending on how many rounds the algorithm must do to find the largest combinations that meet the minimum support threshold.  I have used an average of 5 runs where the maximum number of runs must execute (ie generation was lucky and large combinations meet support threshold).

## Linear Code

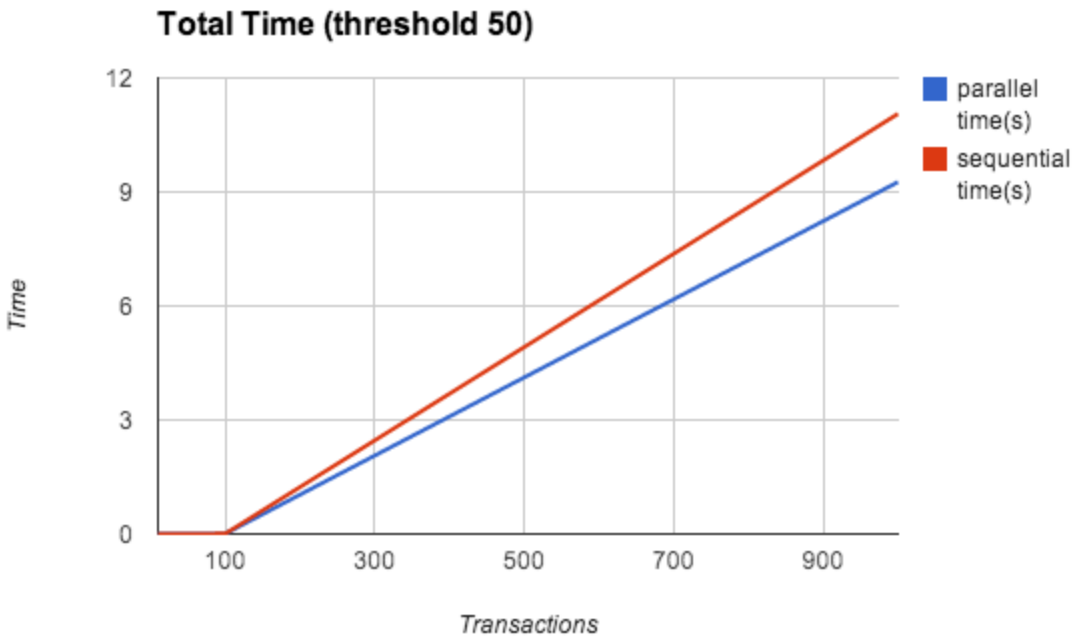


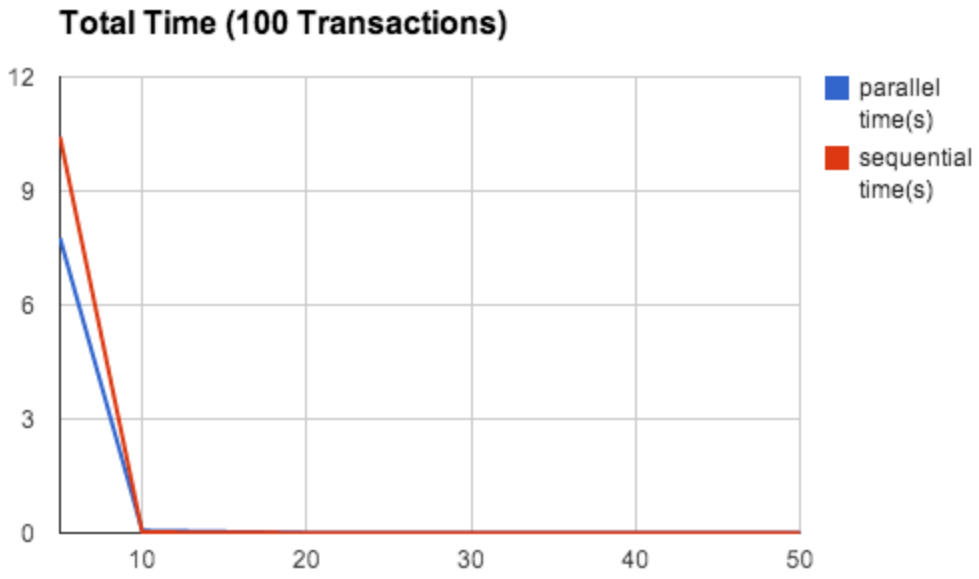

x-axis - total number of transactions
y-axis - total time in seconds




x-axis - support threshold
y-axis - total time

## Parallel Code



As you can see, the parallel time is slightly improved whenever the time taken is not trivial.