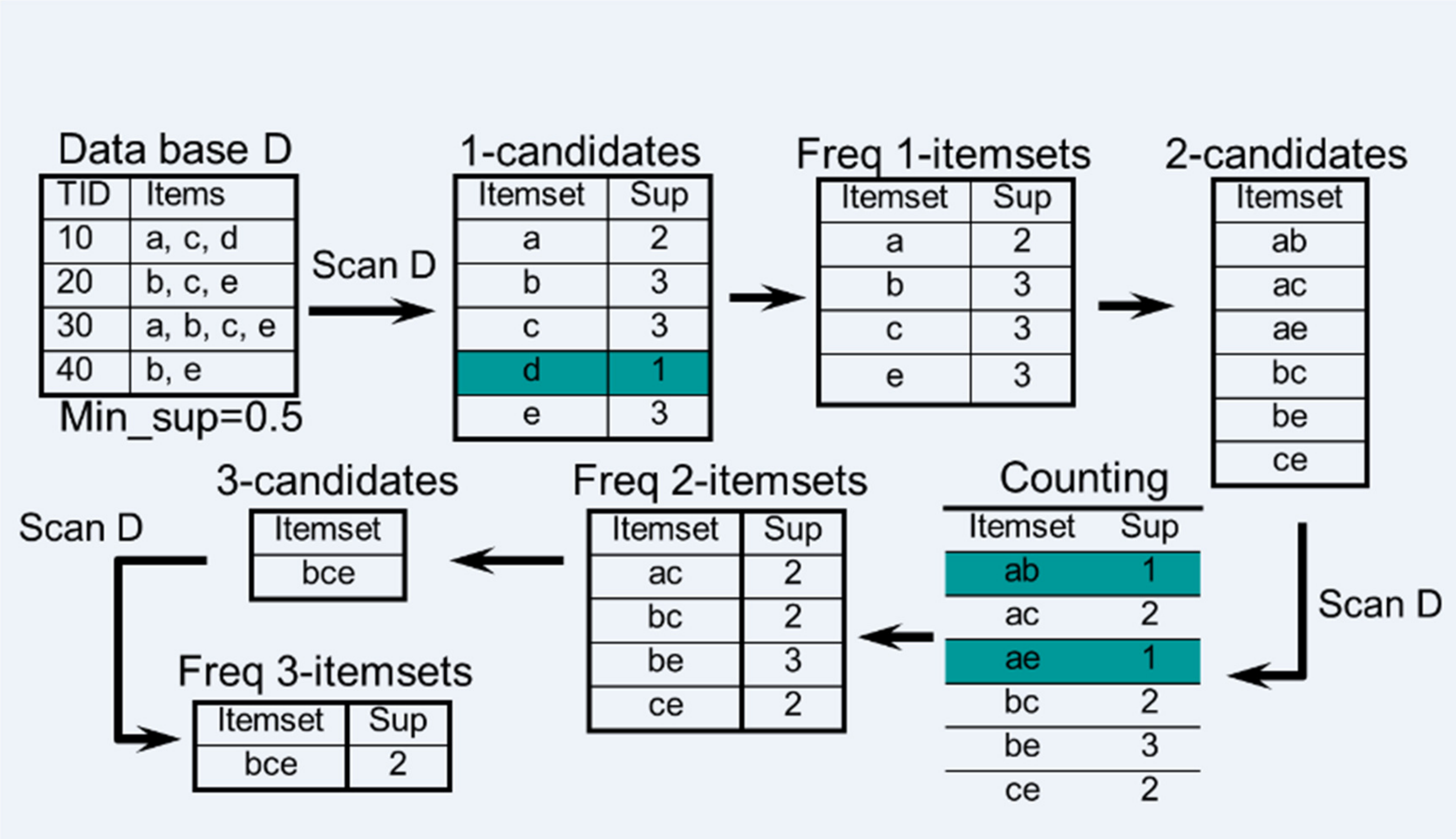
CSE 633 Fall 2012
Parallel Apriori Algorithm and Frequent itemsets with Gene Expression Data

Hyunwoo Sung

Generating Frequent Itemsets



Data base D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=0.5

Scan D

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

3-candidates

Itemset
bce

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2

Gene Expression Data Used For Input

Sample1	UP	UP	Down	UP	ALL
Sample2	UP	Down	UP	Down	ALL
Sample3	Down	Down	UP	UP	ALL
Sample4	Down	Down	Down	Down	AML
Sample5	UP	UP	Down	Down	ALL
Sample6	UP	Down	Down	Down	ALL
Sample7	UP	UP	Down	UP	AML
Sample8	Down	Down	UP	UP	ALL

11	21	30	41	51
11	20	31	40	51
10	20	31	41	51
10	20	30	40	50
11	21	30	40	51
11	20	30	40	51
11	21	30	41	50
10	20	31	41	51

When transmitting data between processors, they exchange integers that are transformed instead of String. UP becomes 1 and Down becomes 0. Also, the name of the cancers such as ALL or AML becomes 1 or 0, respectively.

Sequential Apropri Algorithm Implementation in C++

```
ItemsetIndex = 1;

while(itemsetIndex < numberOfItems );
{
    generateCandidates(itemsetIndex);
    generateFrequentItemsets(itemsetIndex);
    print(frequentItemsetTableElements);
    itemsetIndex++;
}
```

Continued from Implementation

```
GenerateCandidates(int sequence)
for I = 0 to candidates.size()
    for j = I + 1 to candidates.size()
        construct the candidates through self-joining.
        store the candidates in the map. (key = candidate, value = 0)
```

Worst case: O(n^2) $when\ sequence > 2,\ C$*: number of candidates*

I: $sequence$ (i:1,2,3,…number of transactions - 1)

Continued from Implementation

```
generateFrequentItemsets(sequence);
    for I = 0 to numberOfTransactions(N) in a transaction matrix
        for j = 0 to numberOfItems(W)
            iterate through HashMap
            get the key(item) and compare the key with the actual transaction items in order to count the # of transactions
            update the value that maps to the key in HashMap

    if the support of itemsets >= MIN_SUPPORT
        place the selected itemsets on CandidateItemsetTable
```

- Best: O(N^2*C) when sequence == 1, C = # of candidates
- Worst: O(N^2*C' *C'') $when$ sequence $> =2$, $i:itemsetIndex$
- C': $reduced\ \#\ of\ items\ in\ a\ map$
- C'': $increased\ \#\ of\ items\ to\ compare$

A Parallel Implementation

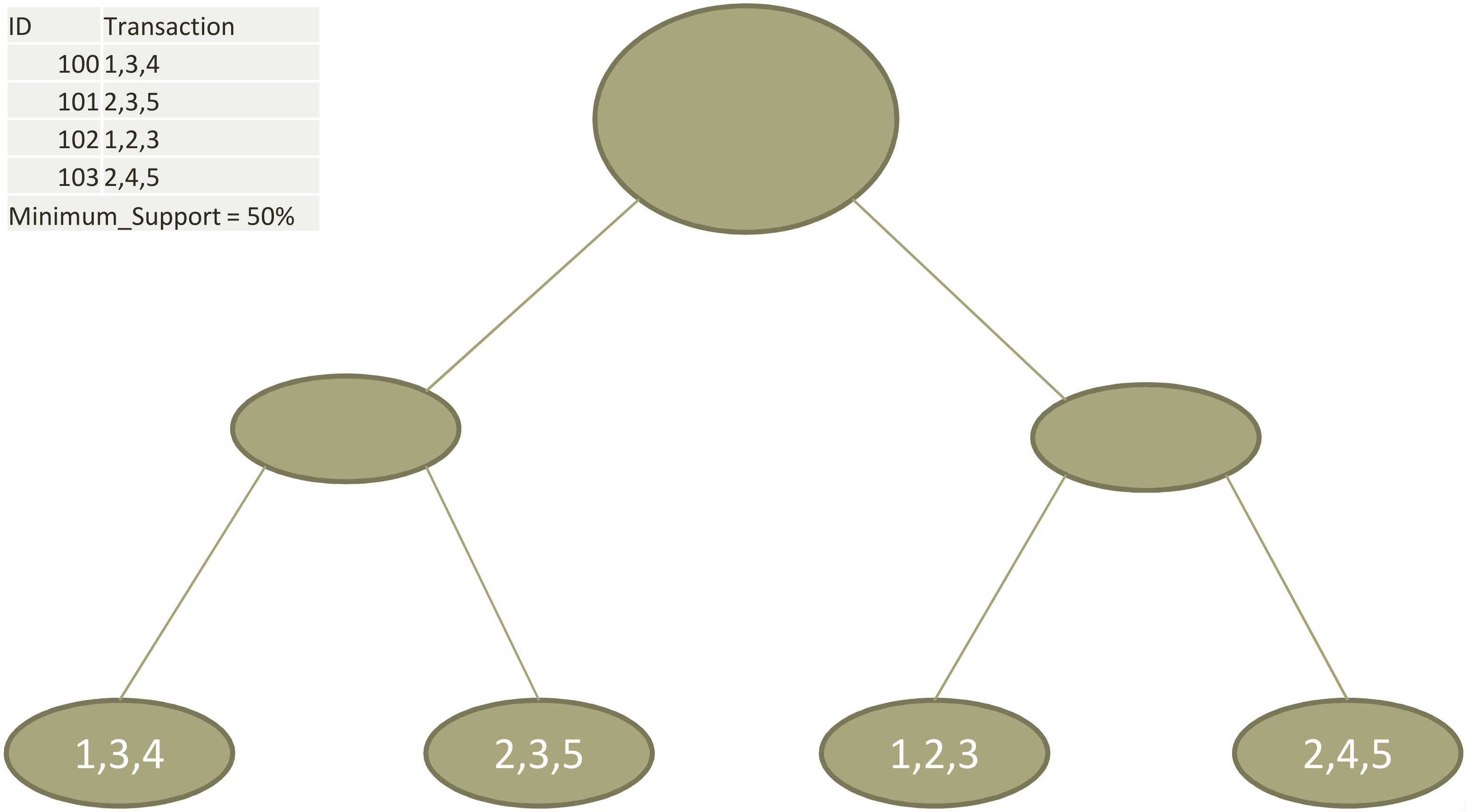
```
bool *alive = new bool[NUM_PROCESSORS];
for( int i = 0; i < NUM_PROCESSORS; ++i )
        alive[i] = true;
get_initial_data ( RANK, globalInput, input );
for( int i = 0; i < log2(NUM_PROCESSORS); ++i)
{
    if( alive[RANK] )
     {
            instance.process(input, output);
            if( should_request( alive, i ) )
           {
                  int neighbour = RANK + (int) pow(2.0,i);
                  request_data( neighbour, neighbor_output );
                  merge_results( output, neighbor_output, input);
                  alive[neighbour] = false;
             }
```

Implementation(Continued)

```
         else
        {
            alive[RANK] = false;
            int neighbour = RANK - (int)pow(2.0,i);
            send_data( neighbour, output);
            delete[] alive;
            MPI::Finalize();
            exit( 0 );
        }
    }
}
  delete[] alive;
  MPI::Finalize();
   return 0;
}
```

A Parallel Solution Illustration

ID	Transaction
100	1,3,4
101	2,3,5
102	1,2,3
103	2,4,5
Minimum_Support = 50%	



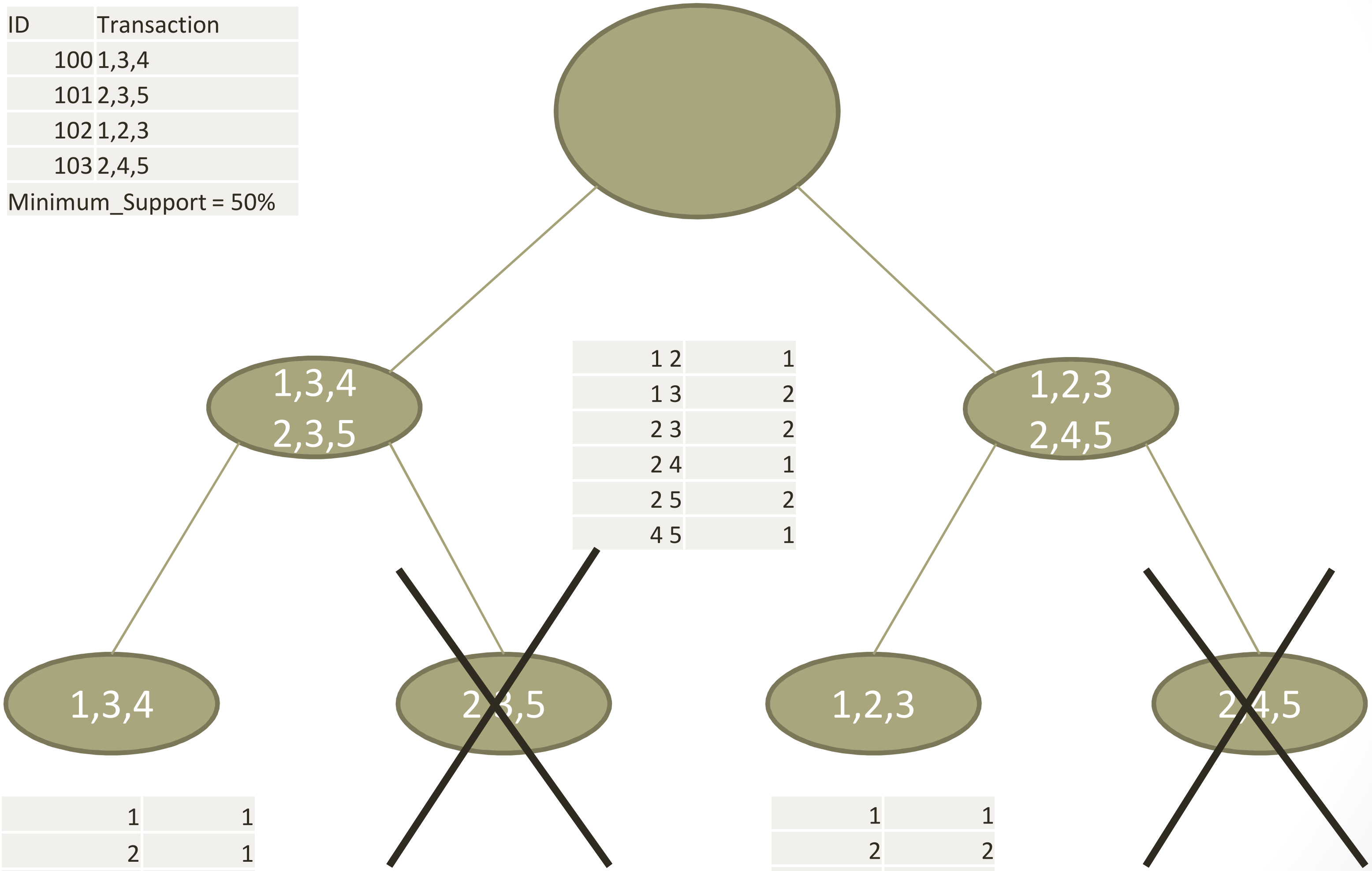
1	1
2	1
3	2
4	1
5	1

1	1
2	2
3	1
4	1
5	1

A Parallel Solution(continued)

ID	Transaction
100	1,3,4
101	2,3,5
102	1,2,3
103	2,4,5

Minimum_Support = 50%



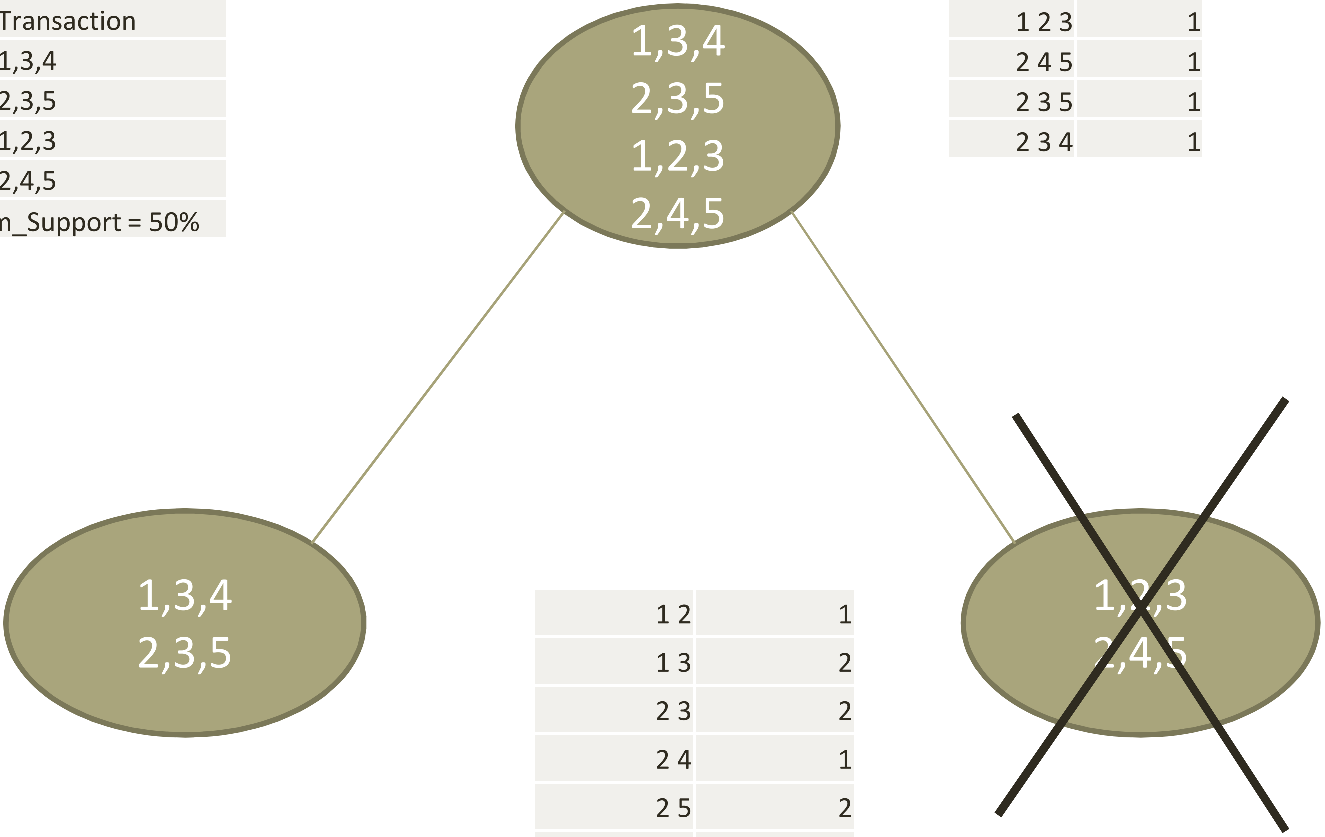
1 2	1
1 3	2
2 3	2
2 4	1
2 5	2
4 5	1

1	1
2	1
3	2
4	1
5	1

1	1
2	2
3	1
4	1
5	1

A Parallel Solution(continued)

ID	Transaction
100	1,3,4
101	2,3,5
102	1,2,3
103	2,4,5
Minimum_Support = 50%	



1 2 3	1
2 4 5	1
2 3 5	1
2 3 4	1

1 2	1
1 3	2
2 3	2
2 4	1
2 5	2
4 5	1

A Parallel Solution(continued)

ID	Transaction
100	1,3,4
101	2,3,5
102	1,2,3
103	2,4,5
Minimum_Support = 50%	

1 2 3	1
2 4 5	1
2 3 5	1
2 3 4	1

1,3,4
2,3,5
1,2,3
2,4,5

The row of the input size grows exponentially as the tree gets trimmed(the processor starts dying. In this example, the final input size becomes 4 by 3 matrix. It started with 1 row. It becomes 2 rows and ends with 4 rows.

Result

- Use the gene expression data that consists of the cancer-causing gene structure information and the name of the cancers.
- There are two different inputs. One is the numbers stored in the two dimensional matrix and the other is the minimum support rate.
- The input size for the runs is 100 * 100, 150*150, 200 * 200, 250 * 250, 300 * 300, 350 * 350, 500 * 500, and 1000 * 1000
- The min_support is set to 50% for all of the runs.

  - In the future, I will try the benchmarks with a lower min_support, since it is more likely to reveal the true speedup/efficiency.
- The number of the processors(cores) used is 1, 2, 4, 8, 16 and 32.
- Each test was run three times and obtained the average run

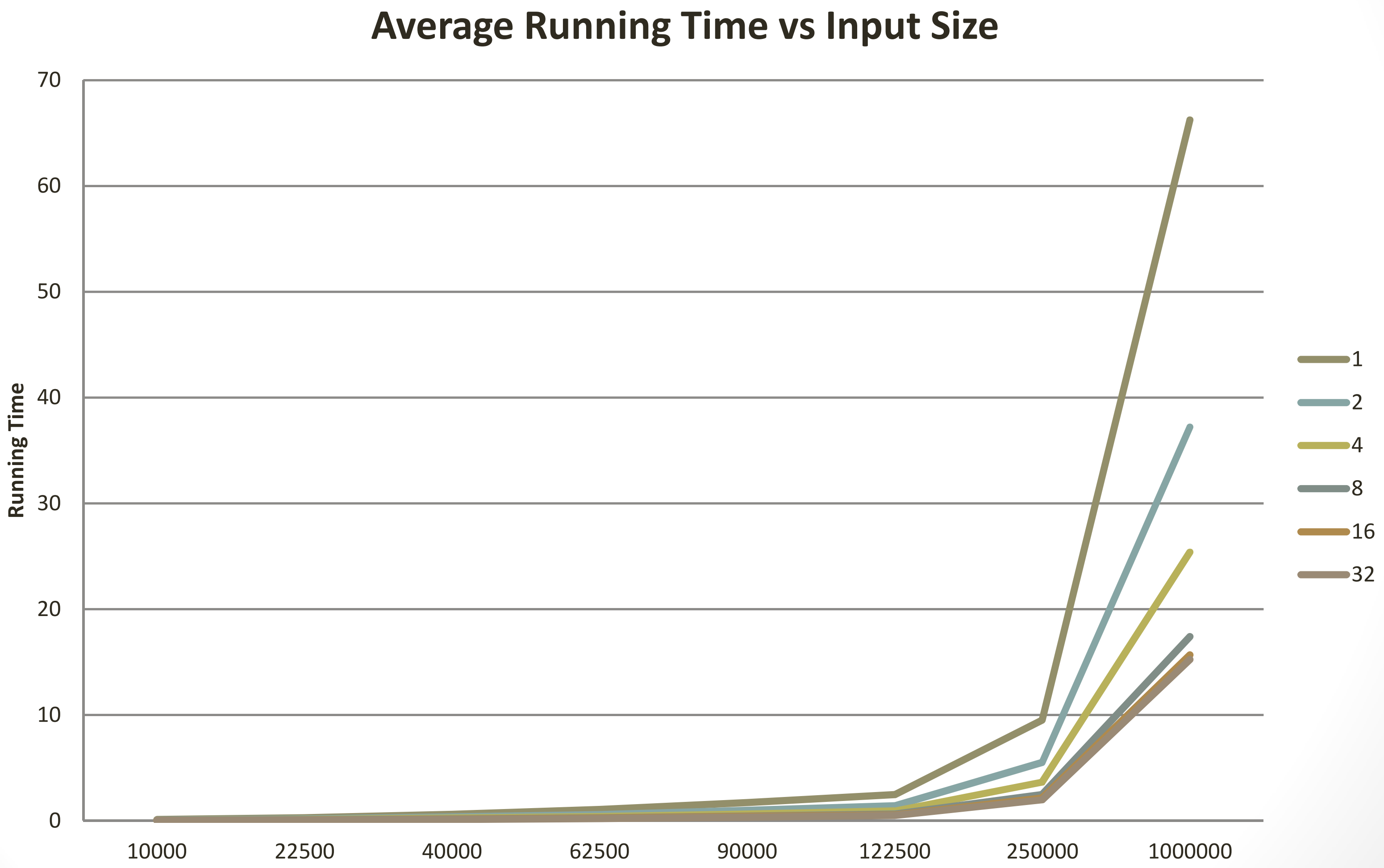
Average Runtime vs. Input Size

A Number of Processors

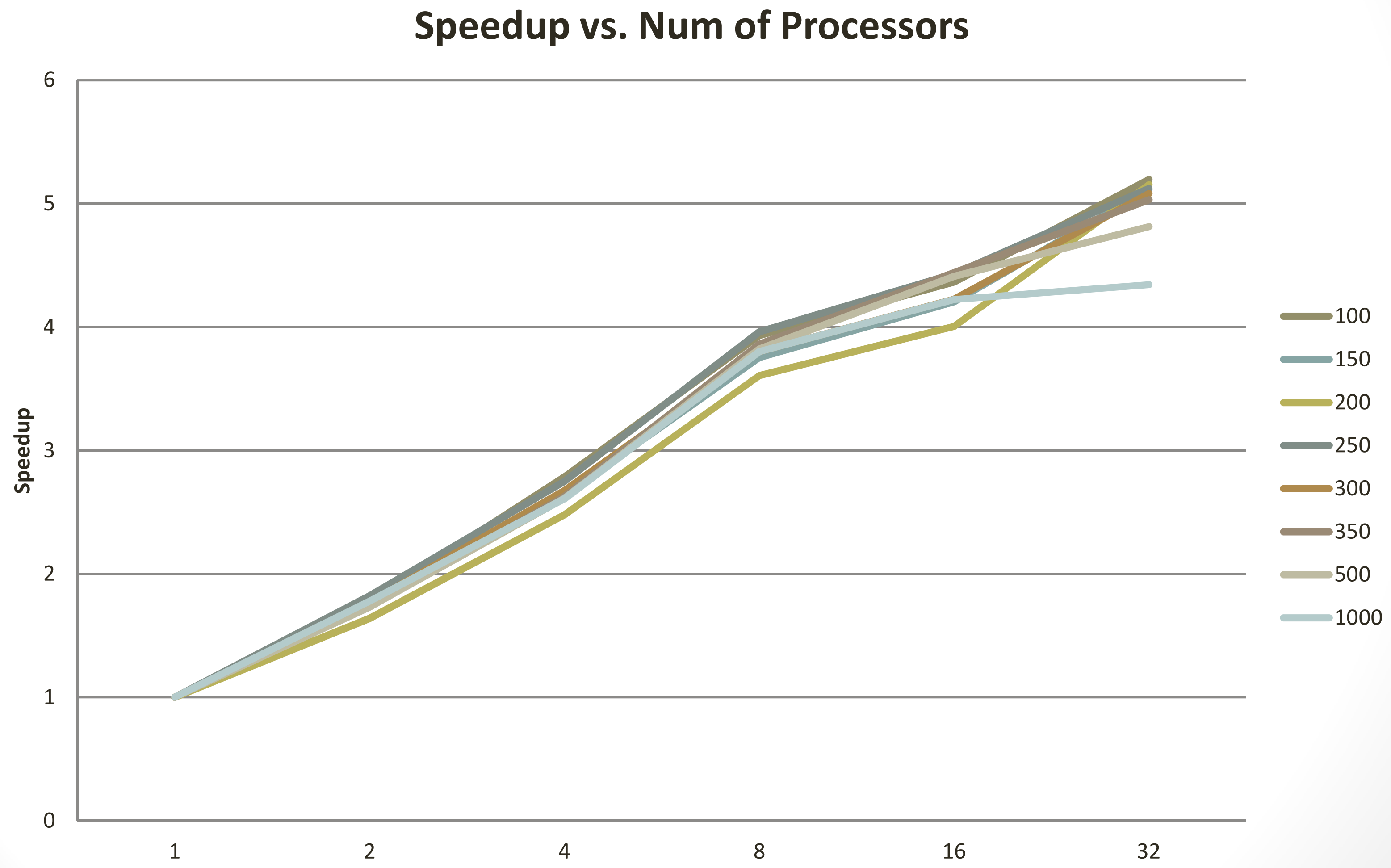
Input size	1	2	4	8	16	32
10000	0.126	0.071	0.045	0.032	0.029	0.024
22500	0.287	0.166	0.109	0.076	0.068	0.056
40000	0.606	0.370	0.245	0.168	0.151	0.118
62500	1.062	0.582	0.386	0.268	0.240	0.207
90000	1.713	0.968	0.640	0.451	0.405	0.337
122500	2.458	1.415	0.936	0.637	0.554	0.489
250000	9.531	5.506	3.646	2.494	2.161	1.980
1000000	66.223	37.225	25.383	17.421	15.686	15.247

Runtime unit: Second

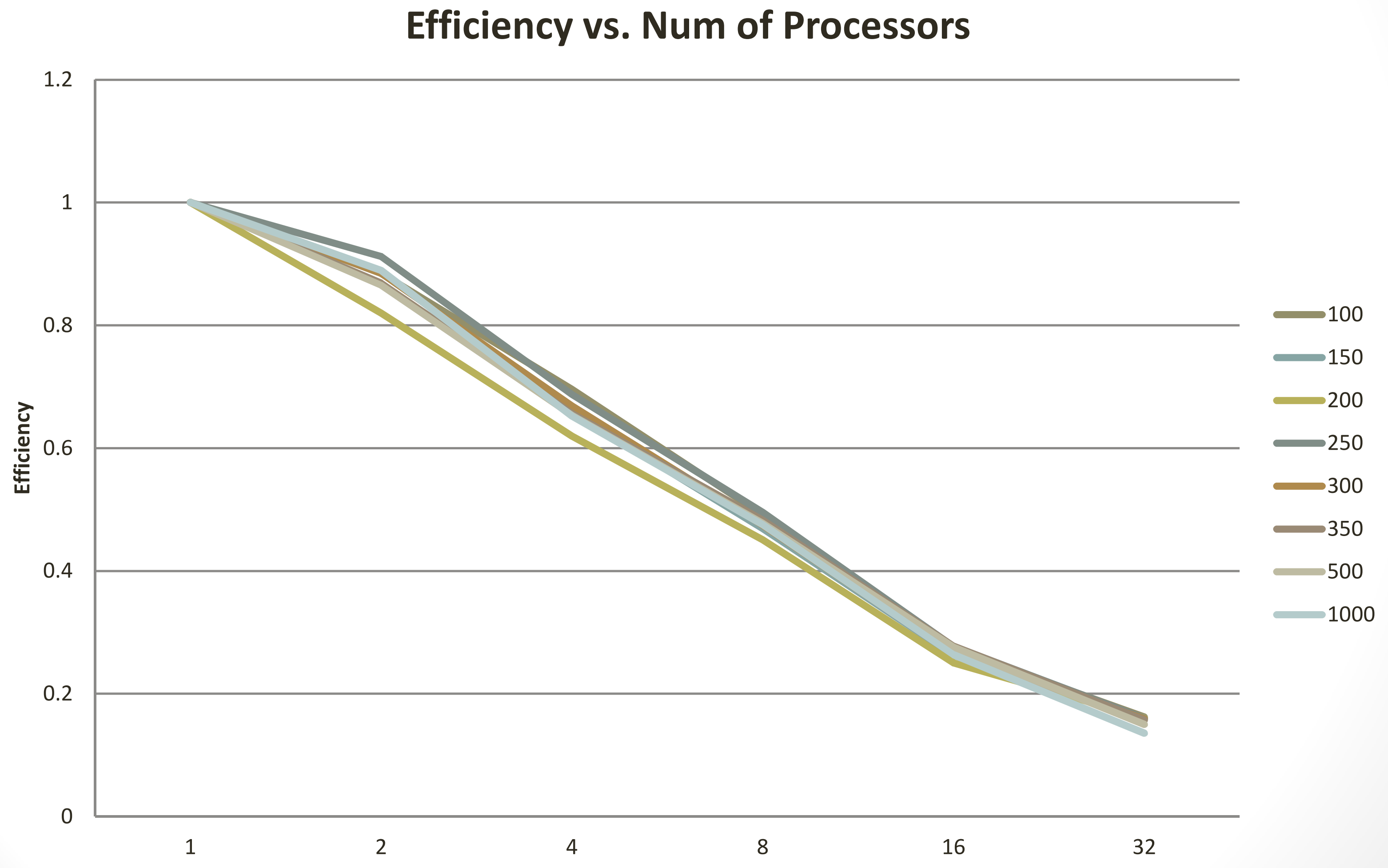
Average Runtime vs. Input Size



Speedup vs. Num of processors



Efficiency vs. Number of Processors



Observation

- Average run time grows exponentially as the input size grows
- Speedup grows linearly at the beginning. Then, it starts going down
- Efficiency drops as more processors are added. Even if more processors would take the divided inputs, their work would not affect the efficiency that much.

Future work

- Test with the various minimum support rate other than the fixed minimum support 50%
- Use various kinds of data such as Waltmart or Top's transactions
- Modify current implementations to achieve the better efficiency
- OpenMP implementation written in C

Reference

http://www.cse.buffalo.edu/faculty/azhang/cse601/cse601-associationrule.ppt