Characterizing Data Mining Algorithms and Applications:
Do they impact processor and system design?

Speaker

Alok Choudhary

Dept. of Electrical & Computer Engineering
Northwestern University

Project Team Members

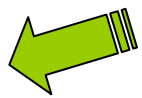
Jayaprakash Pisharath
Ying Liu
Wei-keng Liao
Gokhan Memik

Sponsored by Intel Corporation

Introduction

NORTHWESTERN UNIVERSITY

- Tremendous growth in data
- Sophisticated tools to analyze data
- Growth in systems not enough to keep up with it
- Need for smarter systems and algorithms
- Where to start?
- Analyze & Attack
  - **Investigate future mining needs**
  - **Analyze existing tools and algorithms**
  - Identify shortcomings and try to meet it



Analysis

NORTHWESTERN UNIVERSITY

- Very little is known
- Consistent, steady and phase-driven analysis
- Consider all domains
- First step: construct a benchmark
- To begin with, we look at
  - domains,
  - categories,
  - system features

MineBench

NORTHWESTERN UNIVERSITY

- Data mining benchmark
  - Application suite
  - Popular algorithms (domains)
  - Multiple categories (types) of mining
- Scalability of algorithms
  - Data
  - Algorithmic
- Evaluation options
  - Architecture
  - Measures of interest
- Expansion?

Methodology

NORTHWESTERN UNIVERSITY

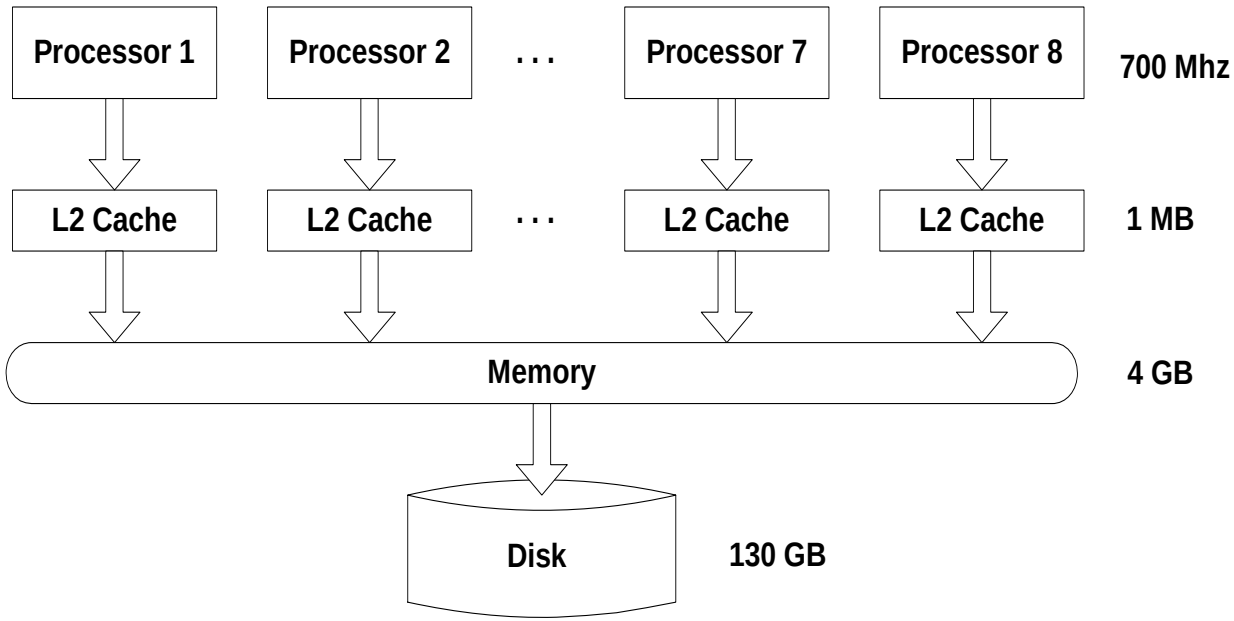
- Include things that have not been considered till now (No TPC-H/ SPLASH/ SPEC)
- Eclectic mix of algorithms
  - Clustering
  - Classification
  - Association Rules
- If scalable, improve performance through parallelization
- Study the characteristics

Benchmark

NORTHWESTERN UNIVERSITY

Algorithms	Category	Description	Lang.
ScalParC	Classification	Decision tree classifier	C
Naïve Bayesian	Classification	Statistical classifier based on class conditional independence	C++
K-means	Clustering	Partitioning method	C
Fuzzy K-means	Clustering	Fuzzy logic based K-means	C
BIRCH	Clustering	Hierarchical method	C++
HOP	Clustering	Density-based method	C
Apriori	ARM	Horizontal database, level-wise mining based on Apriori property	C/C++
Eclat	ARM	Vertical database, break large search space into equivalence class	C++

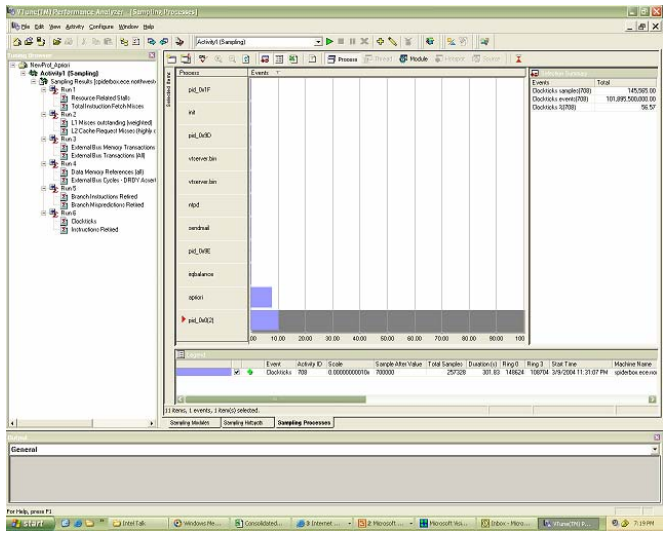
System Architecture



Evaluation Setup

NORTHWESTERN UNIVERSITY

Data Mining Applications



redhat

Red Hat Enterprise Linux AS

Evaluation Metrics

NORTHWESTERN UNIVERSITY

- Evaluation perspectives
  - Algorithmic
  - Architectural
- Scalability Study
  - Single Processor, Multiprocessor Analysis
  - Scalable Data Analysis

- Algorithmic
  - Execution Times
  - Operating System Overhead
  - I/O Overhead
  - Synchronization Overhead
- Architectural
  - L1 Cache Analysis
  - L2 Cache Analysis
  - Memory Access Pattern Study
  - CPI Behavior
  - Instruction and Branch Behaviors

Data Set

NORTHWESTERN UNIVERSITY

- Clustering
  - HOP: ENZO data
    - Small: 61440 particles, Medium: 491520 particles and Large: 3932160 particles.
  - K means, Fuzzy K means
    - Real image database of 17695 pictures
    - 2 features: Color (9 floating points) and Edge (18 floating points)

Dataset	Classification		Association Rule Mining (ARM)	
	Parameter	DB Size(MB)	Parameter	DB Size(MB)
Small	F26-A32-D125K	27	T10-I4-D1000K	47
Medium	F26-A32-D250K	54	T20-I6-D2000K	175
Large	F26-A64-D250K	108	T20-I6-D4000K	350

Fx-Ay-DzK denotes a dataset with Function x, Attribute size y, and Data comprising of z*1000 records.

D denotes the number of transactions, T is the average transaction size, and I is the average size of the maximal potentially large itemsets.

Functional Analysis

(Single Processor Analysis)

ScalParC (Decision Tree Classifier)

NORTHWESTERN UNIVERSITY

A decision tree is a class discriminator that recursively partitions example set until each partition consists entirely or dominantly of examples from one class. Each non-leaf node of the tree contains a split point which is a test on one or more attributes and determines how the data is partitioned.

Partition (Example Set S)

- if (all examples in S are of the same class) then return;
- for each attribute A do
  evaluate splitting point on attribute A;
- use the best splitting found to partition S into S_1 and S_2;
- Partition (S_1);
- Partition (S_2);

* Use Gini index for splitting point selection measure:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2 \qquad Gini(S_1, S_2) = \frac{n_1}{n} Gini(S_1) + \frac{n_2}{n} Gini(S_2)$$

- Find splitting point with minimum Gini

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Instructions Retired	Clockticks
ParClassify	49.78	67.45	73.32	74.71	72.98	50.28	46.04	36.94
Calculate_Gini	33.11	22.21	15.99	14.32	15.52	27.28	31.11	35.65
VRCompare	16.96	10.33	10.68	10.96	11.47	19.62	19.66	26.15
main	0.15	0.01	0.01	0.01	0.02	2.82	3.19	1.26

- Splitting procedure (ParClassify) causes the maximum bottleneck (high resource and data stalls)
- Simple GINI index calculation could get complex (second most resource consuming, and time consuming operation)
- Index comparison (VRCompare) can also be optimized further (less instructions retired, but comparatively longer clock ticks)
- Main does just the delegation work

Bayesian (Statistical Classifier)

NORTHWESTERN UNIVERSITY

Naïve Bayesian classifier, a statistical classifier, assumes the effect of an attribute on a given class is independent of the other attributes. It predicts the probability that a given example belongs to a particular class.

Suppose there are m classes, C_1, C_2, …, C_m, assume $P(C_1) = P(C_2) = \ldots = P(C_m)$, data example $X = (x_1, x_2, \ldots, x_n)$, s_i is the number of training examples belonging to C_i, s_{ik} is the number of training examples of class C_i having the value x_k for attribute k.

- Scan the training data set, calculate s_i and s_{ik} for categorical attributes, calculate mean μ_{Ci} and standard deviation σ_{Ci} for continuous attributes.

- To classify an unknown example X, assign X to the class C_i with maximum $P(X|C_i)P(C_i)$.

Functional Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
nbc_add	24.87	87.22	30	53.59	43.2	13.84	9.96		13.65	11.69
att_valadd	28.87	3.62	35	23.04	25.49	17.15	16.06	7.42	12.67	13.53
as_read	19.51	4.19	10	11.11	14.81	20.77	16.8	6.22	18.26	15.43
tfs_getfld	26.34	3.51	10	7.52	9.47	48.01	56.93	74.63	55.2	59.02
nbc_setup	0.01	0.79	5	0.82	0.97					0.2

- nbc_add is used while scanning the dataset. A tuple is read (added to the list), an update is done to the class distribution list, and counters (sum, mean). Even though this operation has less data ref (13%), it results in high L1 misses (87%) and significant L2 misses. Memory is accessed too often => room for cache optimization.
- tfs_getfld retrieves fields (columns). This takes up considerable amount of the computation time. Lot of fields are retrieved (reason for high data references). Whereas reading attributes (as_read + att_ualadd) is much quicker – but can be improved still (misses).

K–Means (Clustering)

NORTHWESTERN UNIVERSITY

The k-means algorithm takes the input parameter, *k*, and partitions a set of *n* objects into *k* clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low.

- arbitrarily choose *k* objects as the initial cluster centers;
- repeat
  - for each object p_i do
    - for each center c_j do
      - calculate distance between p_i and c_j;
    - find the cluster center *c* nearest to p_i;
    - assign p_i to the cluster centered by *c*;
  - update the cluster center (calculate the mean value of objects in each cluster);
- until no change;

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
euclid_dist	56.4	86.55	88.76	89.38	89.64	54.72	69.24	72.14	68.63	67.68
kmeans_clustering	28.43	6.92	10.63	10.03	9.79	27.71	19.27	27.47	21.04	21.85
find_nearest_point	15.09	6.18	0.09	0.01	0.06	17.56	11.48	0.37	10.33	10.43
main	0.06	0.27	0.25	0.34	0.31	0.01				0.04

- Euclidean distance calculation leads the pack.
  - Too many cache misses (no locality in access pattern)
  - Many data memory references as point coordinates are stored in memory (and hence max bus transactions)
  - Lot of these instructions (68.3%)
  - More room for automation of distance calculation. No need for big cache. Memory resident distance calculation would work best.
- Finding nearest point (center) is cache efficient (significant data ref, but less L2 misses). Lot of points are read at any given point of time (array).
- Clustering function does the work of assigning a point to cluster and recalculating the center. Main is just a block that delegates work.

Fuzzy K–Means (Clustering)

NORTHWESTERN UNIVERSITY

Rather than having a precise cutoff that an object is or is not a member of a particular cluster, Fuzzy K-means assumes that an object can have a degree of membership in each cluster.

- The Fuzzy K-means assigns each pair of object and cluster a probability that indicates the degree of membership of the object in the cluster.
- For each object, the sum of the probabilities to all clusters equals to 1.
- The assignment iterates until the sum of all membership values converges.

The flexibility of assigning objects to multiple clusters is necessary to generate better clustering qualities.

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
fuzzy_kmeans_cluster	54.21	83.04	5.22	6.42	5.36	59.97	51.48	3.67	63.69	57.61
euclid_dist	39.9	11.08	88.75	88.22	88.49	32.93	38.33	94.43	27.22	32.89
sum_fuzzy_members	5.86	5.4	5.49	4.89	5.89	7.09	10.19	1.88	9.08	9.47
main	0.01	0.39	0.36	0.23	0.08					

- Different from K-means even though goal is the same (bottlenecks have changed)
- fuzzy_kmeans_cluster involves fuzzy membership calculation. Lot of computation involved (high # of instructions retired) due to varying membership of each object.
- Euclid distance calculation is not the main bottleneck (but still significant). Cache is still a bottleneck. Lot of branch mispredictions as well – can be avoided if compiler assembles code more smartly.
- Convergence and check for convergence is really not a bottleneck (as we would expect). Implies, a lot more room for parallelization.

BIRCH (Clustering)

NORTHWESTERN UNIVERSITY

BIRCH is an integrated hierarchical clustering method, effective for incremental or dynamic clustering. *Clustering Feature tree (CF tree)* is used to summarize cluster representations. The non-leaf node in *CF tree* stores the *clustering feature (CF)* of its descendents.

$$CF = (N, \overrightarrow{LS}, SS)$$

N	number of objects in the subcluster
$\overrightarrow{LS}$	linear sum on N objects
SS	square sum of N objects
B	branching factor
T	threshold

Phase 1: for each object *p* in database do
insert *p* into the closest leaf node (subcluster) *l*;
if the diameter of the subcluster stored in $l > T$ then
split *l;*
update the information in *l* and its ancestors;

Phase 2: apply a clustering algorithm (i.e. K-means) to cluster the leaf nodes of the *CF tree*.

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
operator|	25.84	13.17	2.04	6.31	4.35	37.24	32.4	15.52	34.2	28.6
operator&&	25.69	6.32	9.6	18.81	13.18		6.64	8.58	7.15	13.28
operator&	2.62			0.74						1.29
operator||	2.55	3.24		6.33	7.78	2.28	2.87	6.64		2.78
operator+=	1.29	0.43	0.45	0.93	4.17		2.58	2.11	2	3
operator=	0.78	14.06	23.06	17.83	17.71				1	2
RedistributeB	8.03	17.15	20.33	1.04	13.1	5.2	10.05	17.01	10.31	9.13
ClosestOne	7.6	8.48	0.57			3.76	2.94		10	6.5
distance	6.39	2.88	0.48	2.72	0.34	4.46	2.32	0.75	4	4.78
Decode	2.74	1.41			0.7	8.18	8.24	17.19	7	6.47

- Software Operators (overridden in C++) are used for distance calculation purposes. Clearly indicates that software operators can be a considerable overhead. Hardware optimization could help (hardware vector processing).
- Very high misses during redistribution of points to its closest seed (to obtain a new set of clusters). Caused due to the nature of tree access.

HOP (Clustering)

NORTHWESTERN UNIVERSITY

HOP, a clustering algorithm proposed in astrophysics, identifies groups of particles in N-body simulation. HOP clustering can be applied to applications involving neighborhood searching geology, astronomy, molecular pattern Recognition, etc.

- Constructing a KD tree:

  Recursively bisect the particles along the longest axis, and move nearby particles into the same sub-domain;

- Generating density:

  for each particle *p* do

  traverse the tree to find N_{dens} neighbors; assign an estimated density to *p*;

- Hopping:

  for each particle *p* do

  associate *p* with its densest neighbor; hop to its densest neighbor till *p* reaches a particle that is its own densest neighbor;

- Grouping:

  Define particles associated to the same densest neighbor as a group;

  Refine and merge groups;

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
smBallGather	38.72	27.03	13.29	12.2	11.39	48.8	4.18	0.44	6.83	22.53
smBallSearch	25.57	32.25	32.61	28.59	28.4	27.36	15.49	15.67	27.65	29.05
smDensitySym	22.48	4.23	3	2.59	2.55	8.4	77.65	81.55	61.27	38.44
kdMedianJst	6.5	24.88	32.33	35.25	36.23	2.88	2.56	2.24	4.13	9.8
PrepareKD	0.82	1.09	2.99	3.05	3.07	0.07	0.01	0.01		0.04
kdUpPass	0.53	0.99	1.95	1.74	1.75	0.09		0.03		0.03
FindGroups	0.52	2.02	2.05	3.71	3.71	0.02	0.01	0.01	0.01	0.01
main	0.86	1.38	2.82	3.74	3.78	0.24	0.01	0.01	0.01	0.01
SortGroups	0.29		3.22	3.47	3.49	0.07				

- smBallGather: Gathering densest and nearby neighbors in ("Hopping" step) stalls for resources to the worst extent (lot of data ref).
- But execution time of smBallSearch (finding nearest neighbor and generating density) is more than smBallGather. They are similar in functionality. Creates maximum misses.
- Density calculation takes the maximum time (used by smBallSearch). But data efficient as such – less misses. But not instruction efficient – lot of room for improvement in branch mispredictions (88% of mispredictions).

Apriori (ARM)

NORTHWESTERN UNIVERSITY

Apriori is a level-wise search method to find the frequent itemsets in database records, where *k*-itemsets are used to explore (*k*+1)-itemsets. *Apriori property*, every subset of a frequent itemset has to be frequent, is used to prune many Candidate itemsets.

- L_1 = frequent 1-itemsets;
- for ($k = 2$; $L_{k-1} \neq \varnothing$; k++) do
  - L_{k-1} is used to generate candidates set C_k;
  - eliminate those candidates c having a subset that is not frequent from C_k;
  - for each transaction $t \in$ Database
    - for each $c \in C_k$ do
      - if (c is a subset of t) c.count++;
  - $L_k = \{c \in C_k \mid c.\text{count} \geq min\_sup\}$
- return $L = \cup_k L_k$;

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
Database_readfrom	43.82	59.3	51.67	66.67	66.67	15.26	67.95	62.8	13.86	13.86
subset	18.74	21.65	18.44			52.89	22.17	28.36	57.54	57.54
main_proc	30.41	5.37	18.17			20.62	1.64	1.79	17.05	17.05
increment	2.44	3.87	3.29			4.22	1.82	2.72	4.55	4.55
find_in_list	0.54	3.95	4.06			0.68	3.91	0.05	0.73	0.73
rehash						0.01	0.08		0.01	0.01
string				33.33	33.33					

- Database_readfrom produces the maximum stalls (lesser execution times). Data ref are less, but still misses and stalls are a lot (due to absence of locality as the access pattern is uniform). Even after modifying the access to bulk-load data to the memory (instead of contiguous access), it still remains a bottleneck.
- Subset calculation is the most active routine. Max data references, but comparatively less L2 misses.
- c.count++ is implemented as a separate function (increment) – takes up 4% of the time and considerable overheads. Even if a function is simple, compiler overheads do arise and hit the architecture.

ECLAT (ARM)

NORTHWESTERN UNIVERSITY

Eclat is an association rule mining algorithm based on equivalence classes. An equivalence class is a potential maximal frequent itemset. Efficient lattice traversal techniques are used to identify all the true maximal frequent itemsets. All the frequent subsets of the maximal frequent itemsets are generated during traversing.

Phase 1: finding equivalence classes

- generate frequent k-itemsets L_k (i.e. k=2);
- create equivalence classes F_k by joining itemsets in L_k, based on their common k-1 length prefix;

Phase 2: bottom-up traversing lattice

Bottom_Up(input: F_k, Output: $L = \cup L_j\ (j >= k)$)

- generate candidate set C_{k+1} based on F_k;
- for each transaction $t \in$ Database do
  for each $c \in C_{k+1}$ do if (c is a subset of t) c.count++;

$L_{k+1} = \{c \in C_{k+1} \mid c.\text{count} \geq min\_sup\}$

create equivalence classes F_{k+1};

- if ($F_{k+1} \neq \varnothing$) then Bottom_Up(F_{k+1});

Function Profiling

NORTHWESTERN UNIVERSITY

Function	Resource Related Stalls	L1 Misses	L2 Misses	Bus Transactions	Bus Memory Transactions	Data Memory References	Branch Instructions	Branch Mispredictions	Instructions Retired	Clockticks
add	30.7	28.82	29.49	33.26	34.22	8.37	4.18	0.44	6.83	22.53
get_intersect	26.34	21.06	23.5	22.53	22.34	55.5	77.65	81.55	61.27	38.44
process_invert	12.36	8.77	15.62	16.47	16.39	5.57	2.56	2.24	4.13	9.8
partition_get_lidxsup	0.04	0.05	0.04	0.04	0.03		0.01	0.01		0.04

- add routine creates an equivalence class (used multiple times in both phases). Less data ref, but most stalls and misses. Better creation of equivalence class is needed.
- get_intersect involves finding common items in order to create a longer itemset. Lot of branches and many are mispredicted as well. Better reorganization of instructions might help.
- process_invert and partition_get_lidxsup are used during the subset calculation (not much of a bottleneck).

Scalability Analysis

Multiprocessor, Scalable Data Analysis

Parallelization

NORTHWESTERN UNIVERSITY

- Goal: to test the scalability of data mining applications on SMPs
- Parallel versions of our benchmark applications.
- Currently 5 parallel applications (out of 8)
  - ScalParC (Zaki's SMP algorithm),
  - K-means (data parallelism),
  - Fuzzy K-means (data parallelism),
  - HOP (data parallelism), and
  - Apriori (common candidate partitioned database strategy).
- Data sets – as described earlier (Small, Medium, Large).

Execution Times

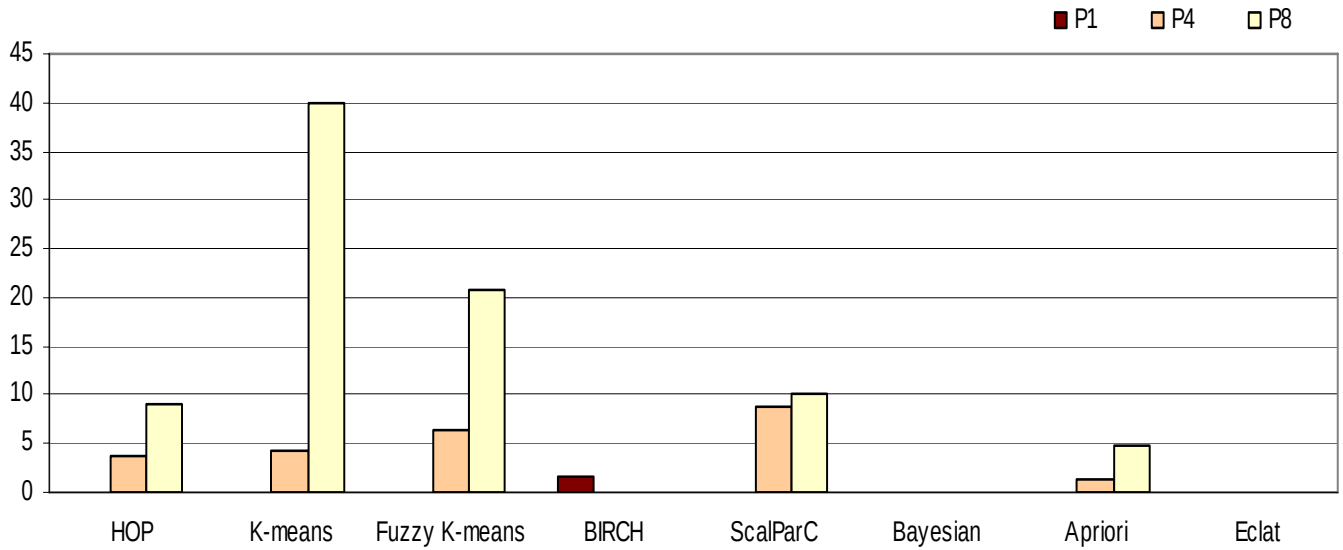
NORTHWESTERN UNIVERSITY

Program	Data set = S			Data set = M			Data set = L		
	P1	P4	P8	P1	P4	P8	P1	P4	P8
HOP	6.30	3.50	5.25	52.70	1.92	6.06	435.30	3.40	5.34
K-means	5.70	2.85	4.38	12.90	3.91	4.96	-		
Fuzzy K-means	164.10	3.01	6.22	146.80	3.44	5.42	-		
BIRCH	3.50			31.70			172.60		
ScalParC	51.00	3.78	4.90	110.60	3.88	5.12	225.90	4.02	6.19
Bayesian	12.60			25.10			51.50		
Apriori	6.10	2.03	2.35	102.70	2.66	3.37	200.20	2.76	3.18
Eclat	11.80			81.50			127.80		
	Seconds	Speedups		Seconds	Speedups		Seconds	Speedups	

- ScalParC scales very well, Fuzzy K-Means performs well.
- Balanced partitioning of data in ScalParC => very less memory contention (for shared variables). SMP architecture is utilized.
- Apriori has issues with SMP: hash tree is common and is accessed too frequently.

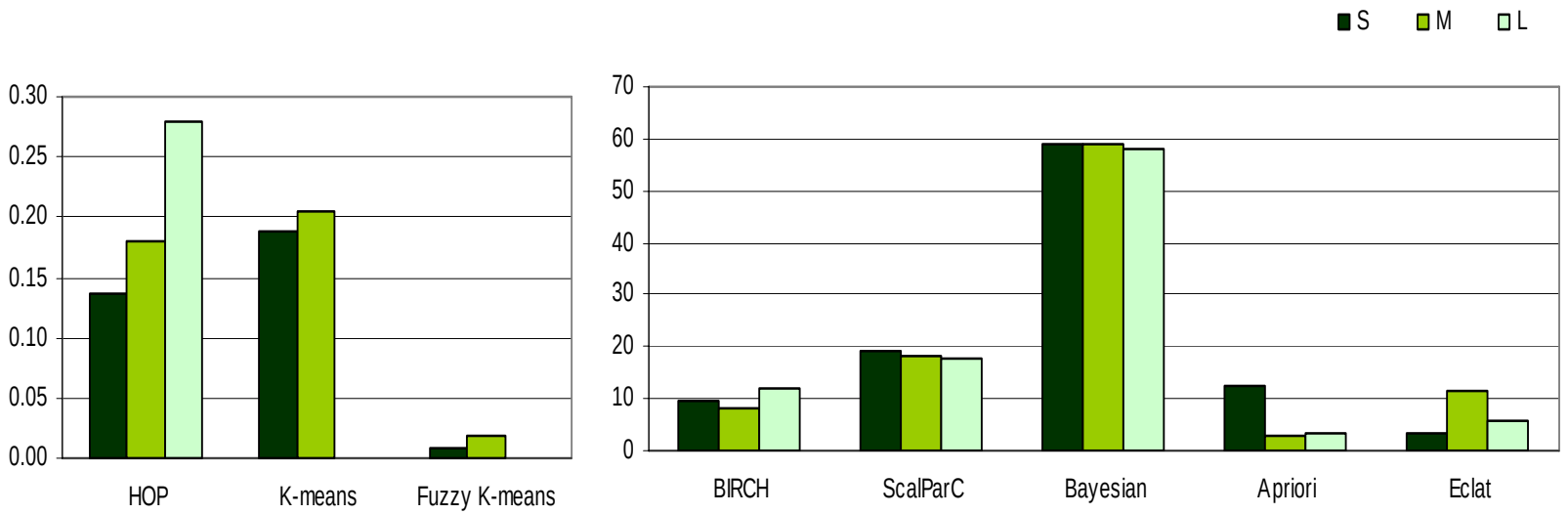
OS Overheads (%)

NORTHWESTERN UNIVERSITY



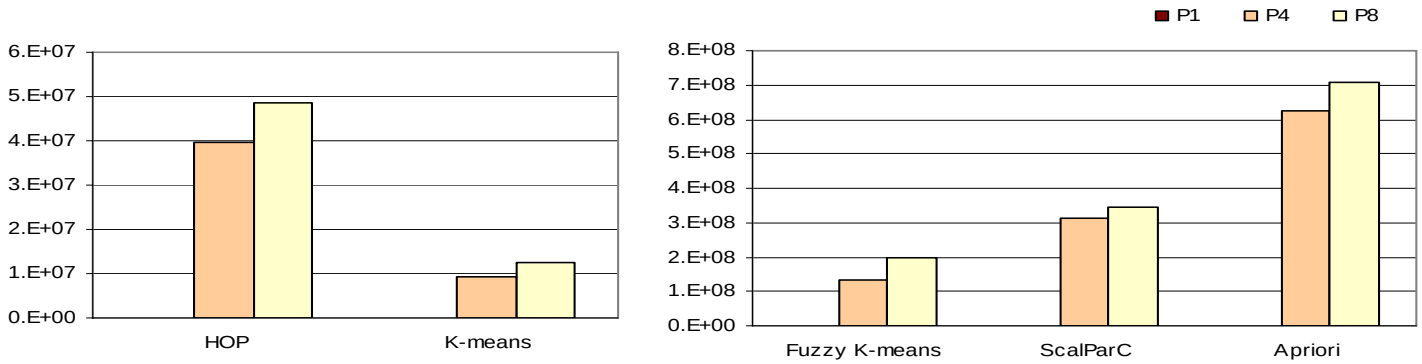
- Overheads increase on parallelization to more processors.
- Mainly from OpenMP overheads, program locks.
- OpenMP programming environment adds extra execution cycles => affects execution times. Unless speedup is good, OpenMP could be destructive.

I/O Time (%)

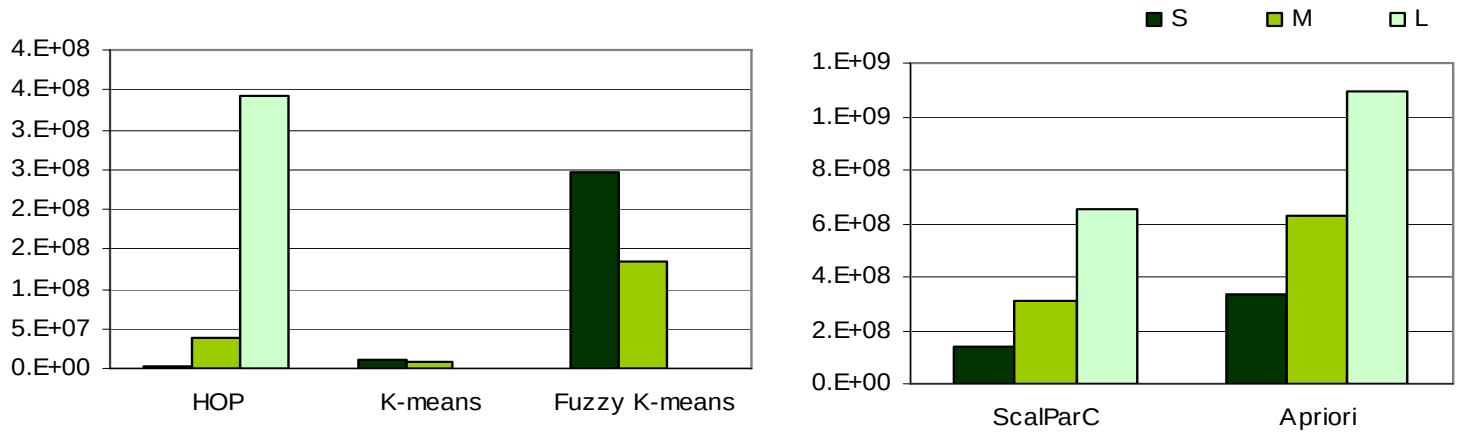


- Very small I/O overheads in the program – Bayesian is an exception.
- Bulk read (ScalParC) vs. Character based (Bayesian) read operation. Character reads prove to be costlier in terms of I/O.
- I/O overheads increase with data size

Synchronization Time (cycles)

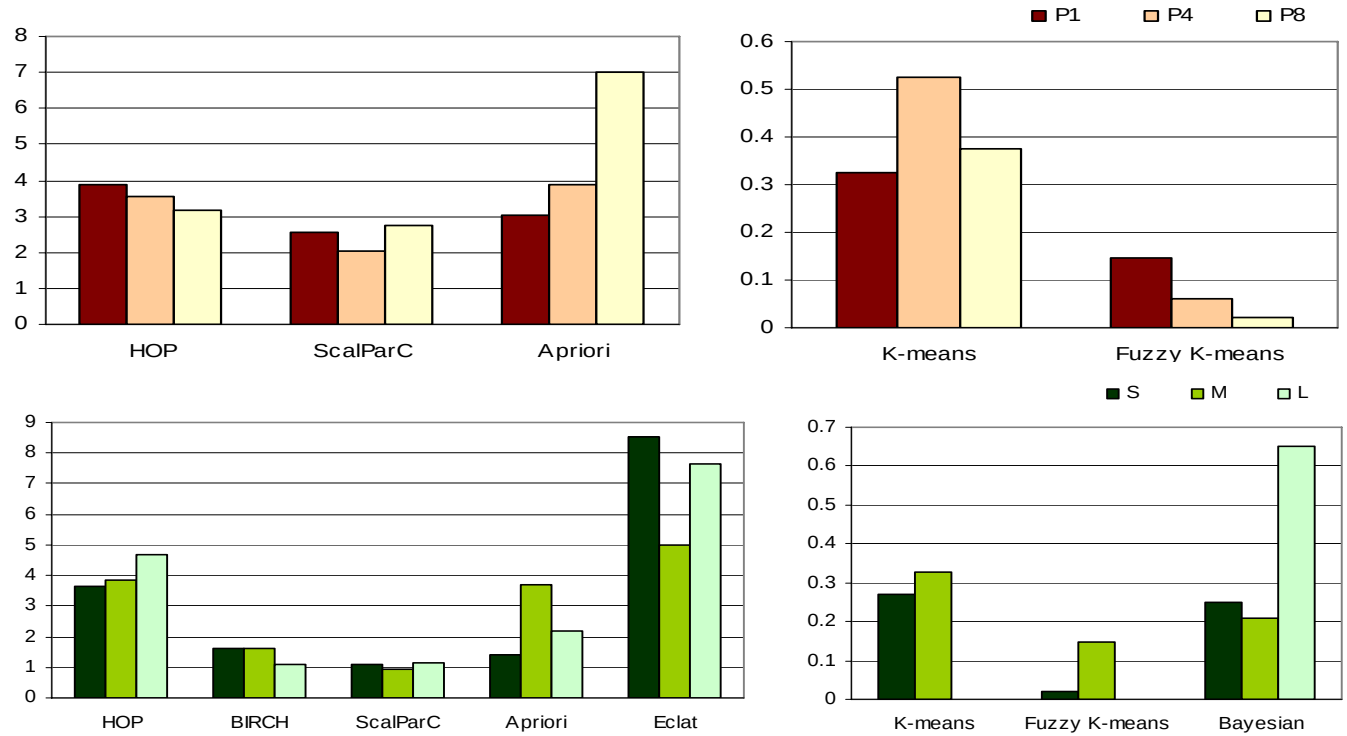


Synchronization time in CPU cycles for all applications. The synchronization time increases when computation is scaled to multiple processors due to increased contention for shared variables.



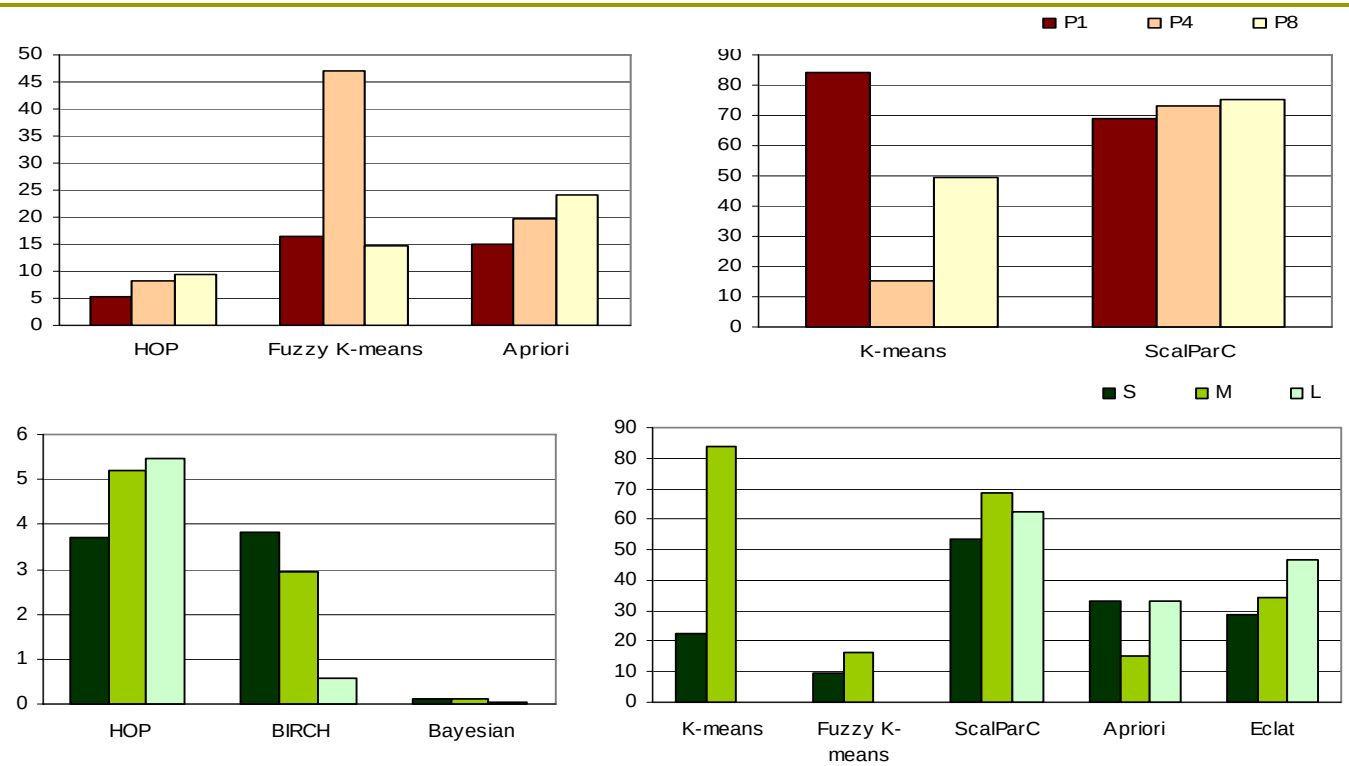
Synchronization time in CPU cycles for all applications for different datasets. The synchronization time increases when data size is increased (as shared data also relatively increases).

L1 Cache Misses (%)



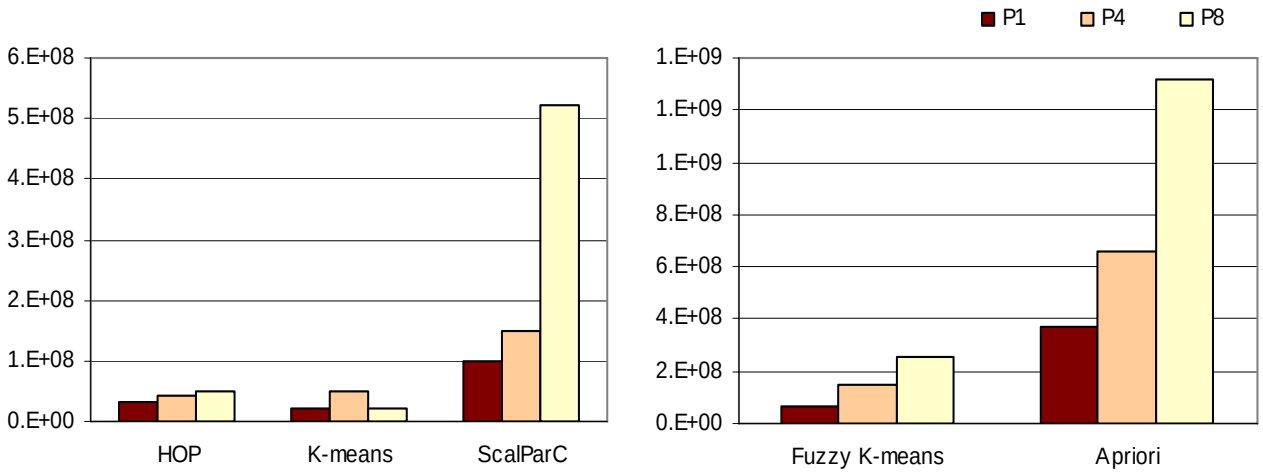
- Applications are drastically different in their L1 cache behavior (two types). Random data distribution is seen.
- Individual processor study suggested master processor is at times overloaded.
- Data size is proportional to data misses (due to limited size of L1).

L2 Cache Misses (%)



- Erratic behavior
  - Data distribution is random as we use dynamic scheduling for parallelization of our applications.
  - Load is also unbalanced
- Misses increase as data sizes are increased (cache not evenly used). There is a lot of contention.

Memory References (cycles)



- Memory access times increase when more processors are used due to the repeated data accesses arising from
  - Locks
  - L2 cache misses
  - Synch time

CPI

NORTHWESTERN UNIVERSITY

Programs	P1	P4	P8
HOP	1.53	1.36	1.45
K-means	1.82	1.56	1.72
Fuzzy K-means	1.36	1.53	1.61
BIRCH	1.29	-	-
ScalParC	2.96	2.63	2.61
Bayesian	1.20	-	-
Apriori	3.83	2.66	3.44
Eclat	9.59	-	-

- CPI is less for some applications. Attributed to contentions (program locks, OMP barriers).
- Some applications are able to overcome it by reducing total synch times.

Preliminary Conclusions

NORTHWESTERN UNIVERSITY

- MineBench is representative.
- MineBench is diverse: two association rule mining algorithms, two classification algorithms, and four clustering algorithms.
- Data mining algorithms are scalable (SMP).
- OS overhead, the synchronization overhead, and the I/O time are usually small in MineBench applications.
- L1 data cache miss rates are small.
- L2 cache miss rates are high, which results in small instruction-level parallelism (measured in CPI).
- Improvements in the performance of processors are likely to have a significant impact on the overall performance of data mining systems.
- Data fetch techniques like prefetching should also improve the performance of the processor considerably.
- Lot of room for algorithmic and architectural optimizations (targeting emerging scenarios) in existing data mining algorithms.

What more is needed?

NORTHWESTERN UNIVERSITY

- More evaluation – more architectures
- Newer measurement tools, measures of interest
- More algorithms
- More domains, categories

Where do we go after this?

NORTHWESTERN UNIVERSITY

- Function profiling
- Kernel identification
  - Individual
  - Common ones across categories/domains
- Kernel optimizations. Also, individual kernel parallelization.
- More detailed characterization
  - More measures of interest
  - More innovative evaluation schemes/tools

Questions?

Thank you