A Processor Selection Method based on Execution Time Estimation for Machine Learning Programs

Kou Murakami, Kazuhiko Komatsu,
Masayuki Sato, Hiroaki Kobayashi

Tohoku University, Japan

Background

- Growing demand of Machine Learning (ML)
  - Statical machine learning; data classification, regression, etc.
  - Deep learning; image recognition, natural language processing, etc.
- ML framework
  - ML frameworks provide APIs that make it easy to implement machine learning programs
  - Frameworks with similar APIs have been developed for each processor
- A problem of machine learning : long execution time
  - Increasing the amount of data to be analyzed and upgrading of algorithms
- Accelerator
  - Accelerators are used to accelerate ML programs
  - Run ML frameworks in accelerators
  - Need to use an accelerator that is appropriate for calculations
  - GPU, TPU, Vector processor, etc.

Objective & Approach

- Objective
  - Accelerate ML programs
    - target : Statistical machine learning

- Approach
  - Selecting a suitable processor for each ML programs
    - A processor selection based on the estimation of execution time

ML frameworks

- ML frameworks
  - Provide algorithms of ML with APIs
  - There are several different frameworks for different functions and processors
    - The same types of frameworks have almost the same APIs
    - Can be run on different processors with a few changes to programs

	x86	GPU	Vector Processor
Numerical framework	NumPy	Cupy	NLCPy
Statistical ML framework	CuPy	RAPIDS	Frovedis
Deep learning framework	TensorFlow	TensorFlow	TensorFlow
Situations for good performance	Small data size	Big data size Compute-bound	Big data size Memory-bound

Overview of the proposed method

- Key idea
  - Decide the processor on which to run a program by selecting a framework
- Steps

1. Estimate the execution time by breaking it down into three components
   - Calculation time, Data transfer time, Setup time
2. Estimate the calculation time : T_{cal}
   - Calculation time of a program
   - Approximate using General Matrix-Matrix Multiplication (GEMM) or STREAM performance
3. Estimate the Data transfer time : T_{trans}
   - Time to transfer data between a host and an accelerator
4. Estimate the Setup time : T_{setup}
   - Setup time for a processor to prepare an execution of a program
5. Select a framework based on the estimation of execution time on each processor

$$execution\ time = T_{cal} + T_{trans} + T_{setup}$$

Estimation of execution time

1. Calculation time : T_{cal}
   - Calculate arithmetic intensities of processors, called a *processor intensity*, by using the fundamental benchmarks in advance
   - Calculate an arithmetic intensity of a target ML program, called an *application intensity*
   - Determine a bottleneck of the target program by comparing the application intensity with the processor intensity
   - Calculate the calculation time according to the bottleneck using equations that are explained later

2. Data transfer time : T_{trans}
   - Measure the *data transfer time* depending on the amount of data in advance
   - Measure the data transfer time based on the amount of data

3. Setup time : T_{setup}
   - Measure the setup time for each processor in advance by approximating the execution time of a small calculation
   - The measure result is directly used

Estimation of the calculation time

- Calculate a *processor intensity*
  - Memory bandwidth : STREAM
  - Performance : GEMM

- Determine a bottleneck of the target program

application intensity > processor intensity : compute-bound
processor intensity > application intensity : memory-bound

compute-bound

$$T_{cal\_comp} = \frac{FLOP_{count}}{FLOPS_{GEMM}}$$

or

memory-bound

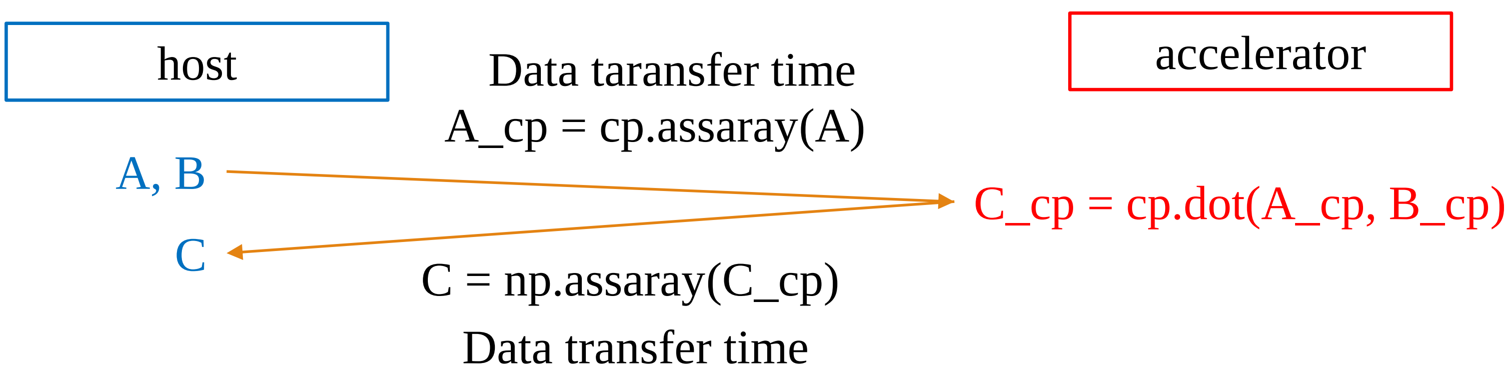
$$T_{cal\_mem} = \frac{datasize}{BW_{STREAM}}$$

Estimation of the transfer time and the setup time

- Data transfer time
  - Evaluate the relationship between data size and data transfer time in advance
    - Calculate the slope for each processor
  - Predict the data transfer time from data size using the relationship revealed by the preliminary evaluation

$$T_{trans} = slope^{proc} \times DATA_{size}$$

- Setup time
  - Substitute setup time with the execution time of 2×2 GEMM



Summary : Estimation of the execution time

- compute-bound

$$T_{exe\_comp}^{proc} = \frac{FLOP_{count}}{FLOP_{GEMM}^{proc}} + slope^{proc} \times DATA_{size} + T_{setup}^{proc}$$

- memory-bound

$$T_{exe\_mem}^{proc} = \frac{DATA_{size}}{BW_{STREAM}^{proc}} + slope^{proc} \times DATA_{size} + T_{setup}^{proc}$$

The parameters in red are clarified in preliminary evaluations!

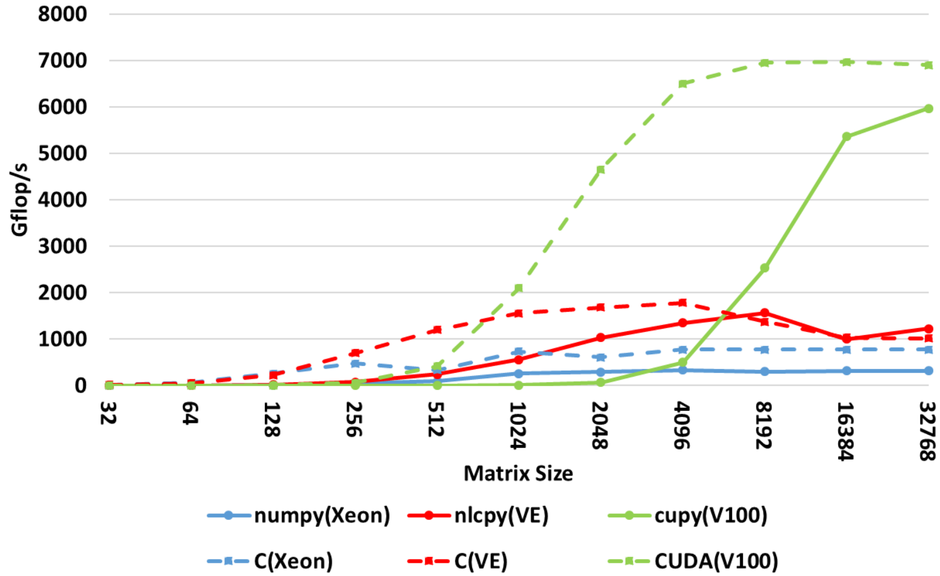
Preliminary evaluation for parameters of the estimation

- Evaluation Items
  - Calculation time : GEMM (sustained performance), STREAM benchmark (sustained memory bandwidth)
  - Data transfer time : Relationship between matrix size and data transfer time
  - Setup time : 2×2 GEMM
- These evaluations use a ML framework
  - Consider the overhead of the ML framework
- Environments

Processors	Intel Xeon Gold 6126	SX-Aurora TSUBASA Type 10B	NVIDIA Tesla V100
Performance (double)	0.883 TFLOPS	2.15 TFLOPS	7.8 TFLOPS
Memory bandwidth	128 GB/s	1.22 TB/s	0.90 TB/s
Frameworks	NumPy, scikit-learn	NLCPy, Frovedis	CuPy, RAPIDS

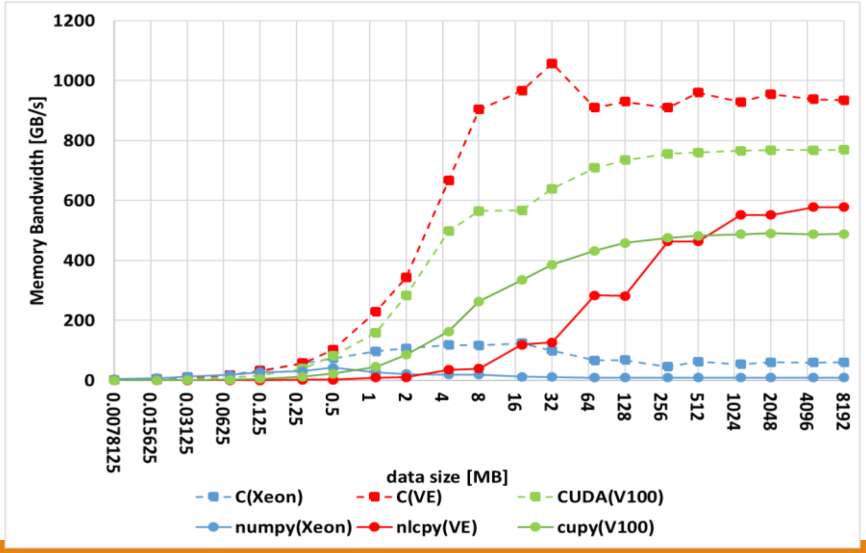
Performance Evaluation of GEMM: $FLOP_{GEMM}^{proc}$

- Evaluate by changing the matrix size from 32 to 32768
  - Compare what is implemented using the framework with what is implemented in C and CUDA
- Computation time is approximated by dividing $FLOP_{count}$ estimated from the code by the corresponding computing performance
  - The computing performance changes depending on the $FLOP_{count}$



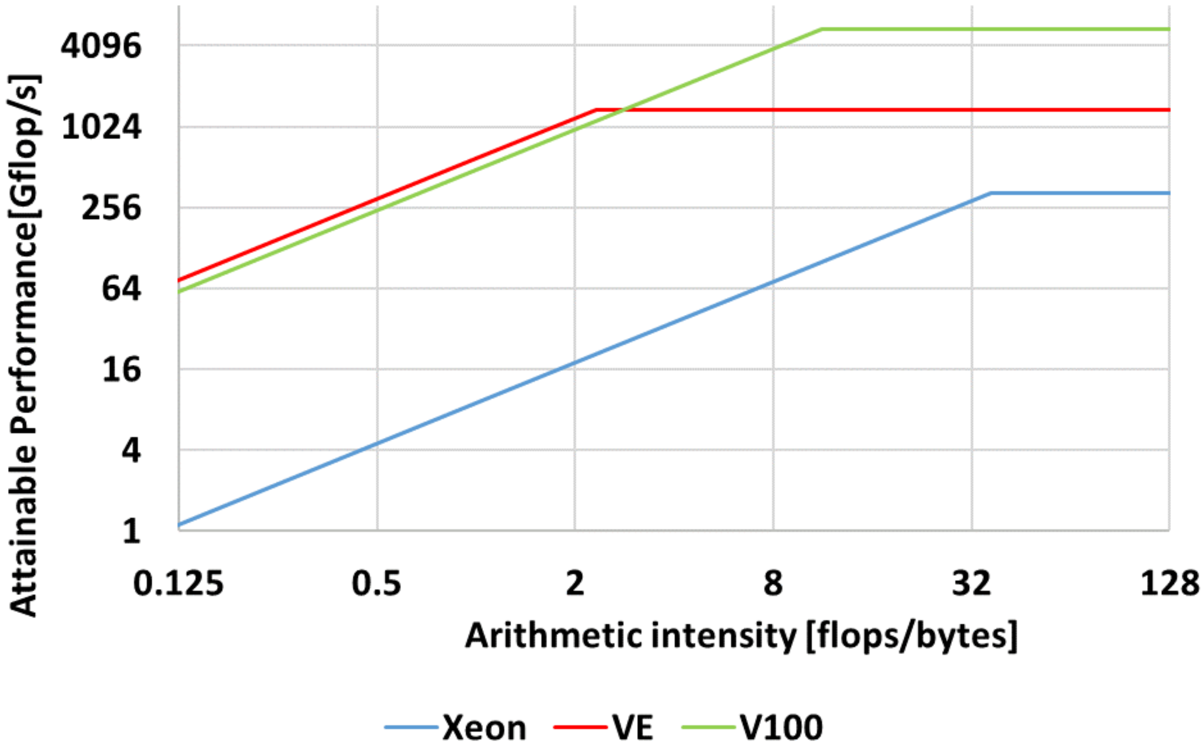
Performance Evaluation of STREAM: BW^{proc}_{STREAM}

- Evaluate triad (a[i] = b[i] +scalar*c[i])
  - Change the data size from 8KB to 8GB
  - Compare what is implemented using the framework with what is implemented in C and CUDA
- Computation time is approximated by dividing $DATA_{size}$ estimated from the code by the corresponding memory bandwidth
  - The memory bandwidth changes depending on the $DATA_{size}$



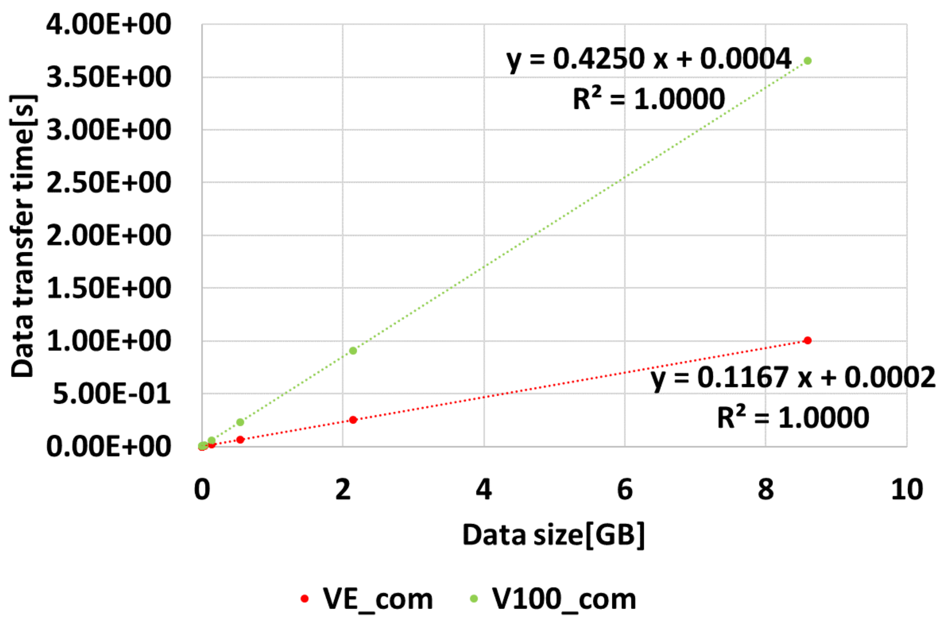
Roofline

- Calculate *processor intensity* from GEMM and STREAM evaluations
- Processor intensity
  - Xeon : 34.7, VE : 2.3, V100 : 11.2



The data transfer time between host and accelerator: $slope^{proc}$

- Evaluate by changing the matrix size from 32 to 32768
- Data size and the data transfer time are proportional



slope [s/GB]

Xeon	VE	V100
0	0.1167	0.4251

The setup time: T_{setup}^{proc}

- Substitute setup time with the execution time of 2×2 GEMM
  - Assume that the data transfer time and the calculation time are small and negligible
- V100, VE, and Xeon have longer setup time in that order

processor	Xeon	VE	V100
Setup time [s]	2.550×10^{-5}	2.519×10^{-4}	1.317×10^{-1}

Evaluation of the proposed method

- Evaluate the General Matrix-Vector Multiplication (GEMV) and an application
  - $FLOP_{count}$: $2n^2 - n$
  - $DATA_{size}$[Byte]: $8(n^2 + n)$

n: size of one side of the matrix

- Application : The liquid clustering application[1]
  - Consist of SOM part and clustering part
  - SOM part : find_bmu and neighbor are repetitive execution

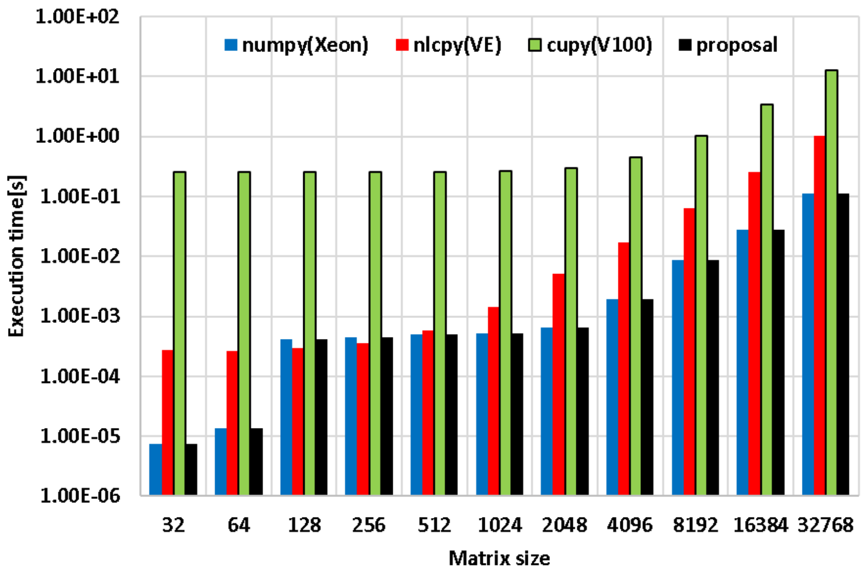
	Function	Calculation	$FLOP_{count}$	$DATA_{size}$[Byte]
SOM part	find_bmu	K-nearest neighbor	$2dnm$	$8d(n+m)$
	neighbor	Gaussian function	$5n^2$	$40n^2$
Clustering part	k-means	K-means	2nkdl	8nkdl

d: dimension, n: neuron, m: the number of data, k: the number of clusters， l: iteration

[1] G. Kikugawa; et al. Data analysis of multi-dimensional thermophysical properties of liquid substances based on clustering approach of machine learning. Chemical Physics Letters, Vol. 728, pp. 109-114, August 2019.

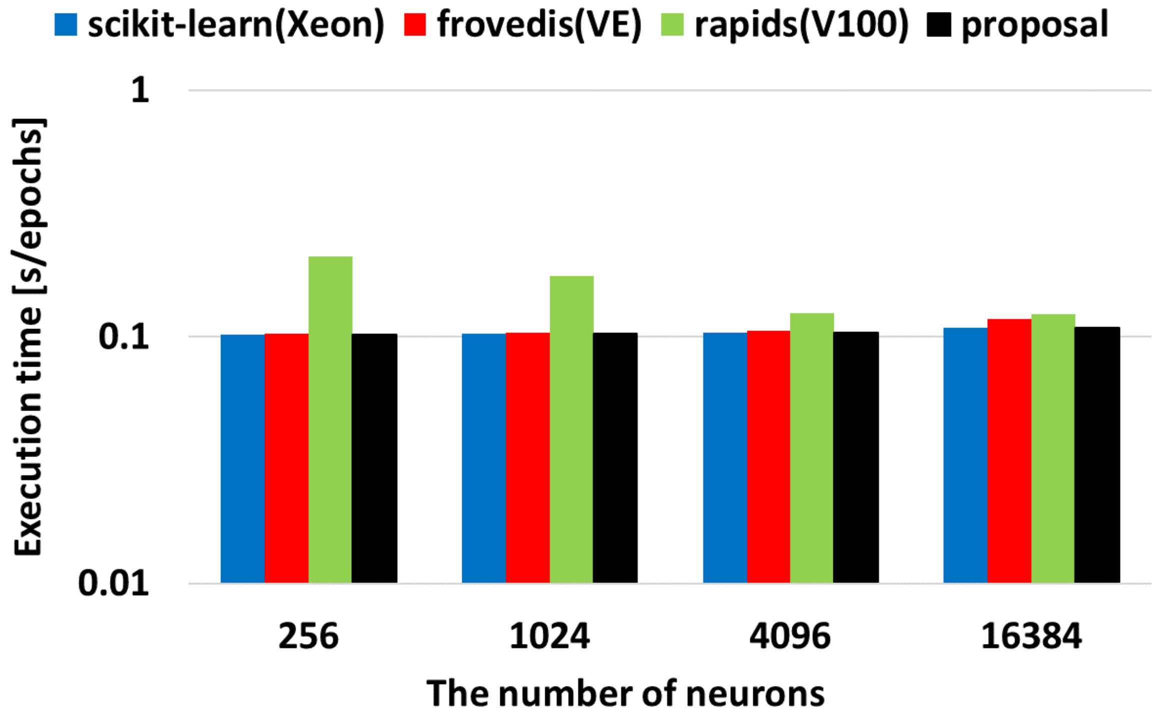
Evaluation: GEMV

- Evaluate by changing *n* from 32 to 32768
- GEMV is memory-bound
- Correctly selected except for n=128, 256
- Factor of false selection
  - Misestimation of the execution time of Xeon
    - The data size used for the estimation is different from the actual data size



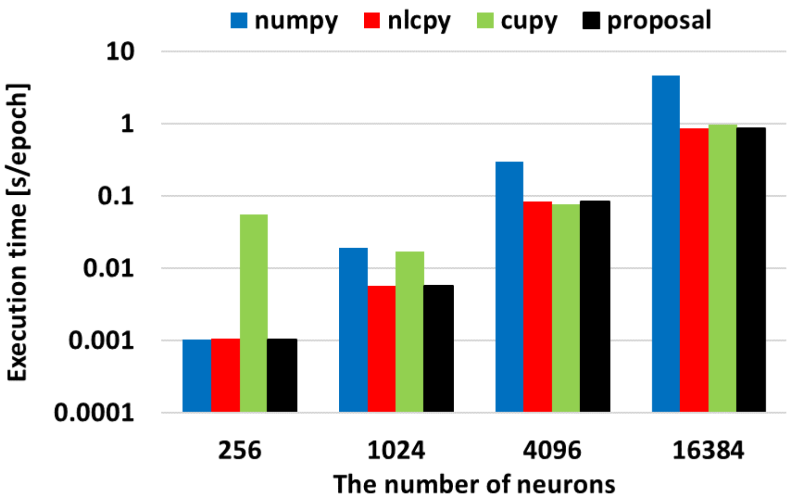
Evaluation: The find_bmu function

- Evaluate by changing n from 256 to 16384
  - $d = 9, m = 512$
- find_bmu is compute-bound
- Correctly select all data size



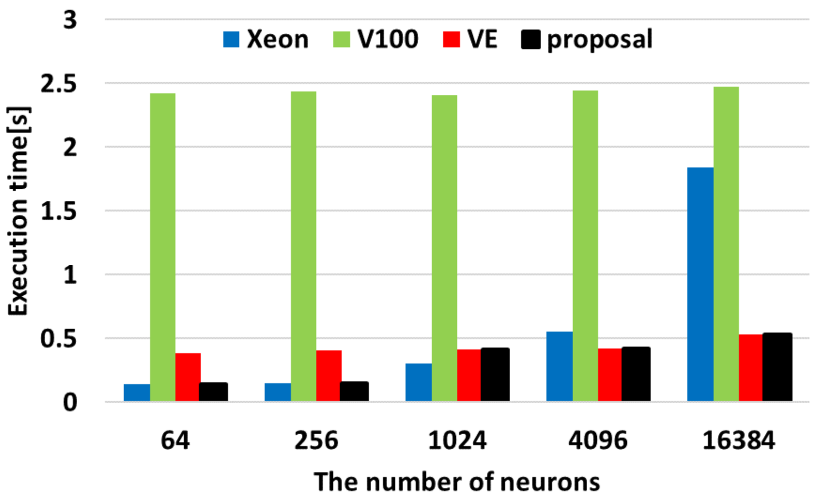
Evaluation: The neighbor function

- Evaluate by changing *n* from 256 to 16384
- neighbor is memory-bound
- Correctly select except for 4096
- Factor of false selection
  - The effect of the setup time on V100 estimation becomes small
    - The neighbor function is repeatedly executed



Evaluation: k-means

- Evaluate by changing n from 256 to 16384
  - $d = 9, m = 512, k = 9, l = 300$
  - l is assumed to be 300 since it cannot be estimated in advance
- k-means is memory-bound
- Correctly select except for 1024
- Factor of false selection
  - Cannot estimate the value of l correctly



Conclusions

- Objective
  - Accelerate ML programs
    - target : Statistical machine learning
- Approach
  - Selecting a suitable processor for each ML programs
    - Processor selection based on the estimation of execution time
- Evaluation
  - Correctly select a processor for almost all algorithms and data sizes
- Future Work
  - Improvement of the estimation
  - A system that automatically selects the processor