Feature analysis for selection of implementations in an accurate matrix-matrix multiplication library

Shota Aoki*, Takahiro Katagiri,
Satoshi Ohshima, Toru Nagai
Nagoya University

*aoki.shota@k.mbox.Nagoya-u.ac.jp

Backgrounds and objectives

- Various social problem are caused by using prediction output by AI without enough verification.
- It is necessary to use with some verification, however it is difficult to verify output by AI manually.
- There are tools that explain output by AI.

- Software auto-tuning (AT) is required to perform numerical computation faster.

- We aim to develop a software auto-tuning (AT) mechanism with AI to reduce the man-hours required for performance tuning in the field of numerical computations.

Ozaki method

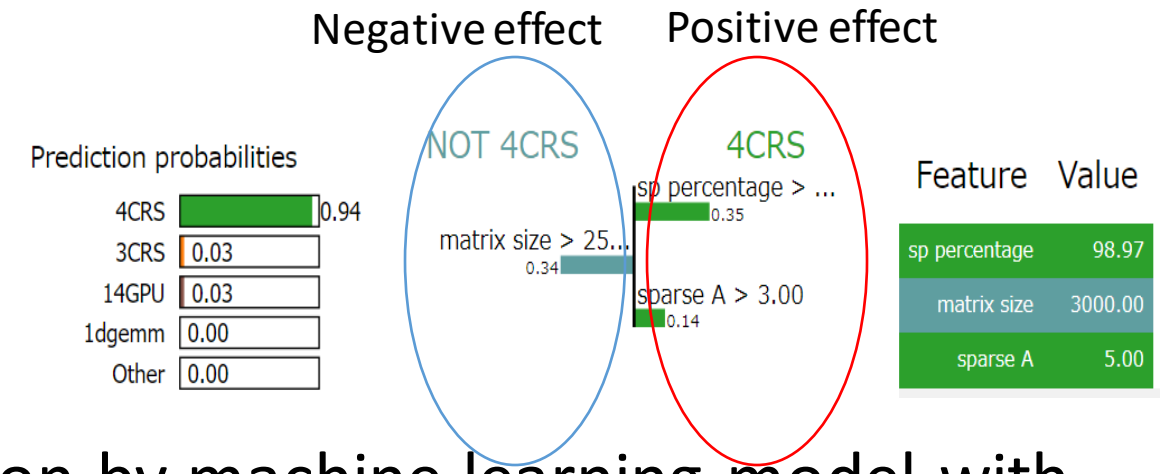
A numerical algorithm for accurate matrix-matrix multiplication (MMM) computation with Ozaki method is executed with the following below 3 steps:

1. Split the matrix $\boldsymbol{A}$ to $\boldsymbol{B}$ into $\boldsymbol{p}$ and $\boldsymbol{q}$ matrices respectively:
   $\boldsymbol{A = A^{(1)} + A^{(2)} + \ldots + A^{(p)}}$
   $\boldsymbol{B = B^{(1)} + B^{(2)} + \ldots + B^{(q)}}$
2. Compute split matrices' products:
   $\boldsymbol{AB = (A^{(1)} + A^{(2)} + \ldots + A^{(p)})(B^{(1)} + B^{(2)} + \ldots + B^{(q)})}$
3. Add them using the accurate sum algorithm:
   $\boldsymbol{AB = A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B^{(1)} + \ldots + A^{(p)}B^{(q)}}$

On the splitting process, some sparse matrices are generated according to range of elements of the input matrices.

The execution of computation process with Sparse matrix-vector multiplication (SpMV) is faster in some cases.

LIME



- The library for XAI.
- LIME explains a result of prediction by machine learning model with showing prediction probabilities and contribution of explanatory variables.
- Users can decide the number of displayed explanatory variables in decreasing order of high contribution.
  - 3 explanatory variables in this research.

Execution environment

Supercomputer “Flow” Type II Subsystem installed at Information Technology Center, Nagoya University
（FUJITSU Server PRIMERGY CX2570 M5） https://www.icts.nagoya-u.ac.jp/ja/sc/overview.html

CPU		GPU	
Processor	Intel Xeon Gold 6230	Processor	NVIDIA Tesla V100 SXM2
The number of cores	20core	The number of cores	2,560 FP64 core
Processor frequency	2.1 GHz - 3.9 GHz	Processor frequency	～1,530 MHz
The number of the processor	2	The number of the processor	4
FLOPS	1.344 TFLOPS / CPU	FLOPS	7.8 TFLOPS / GPU
Memory capacity	192 TiB / CPU	Memory capacity	32 GiB / GPU
Memory bandwidth	141 GB/sec / CPU	Memory bandwidth	900 GB/sec / GPU
Compiler	Intel C++ Compiler (ver.19.5.281)	Compiler	Intel C++ Compiler (ver.19.5.281) CUDA 10.2.89_440.33.01
Option of the compiler	-xHost -O3 –qopenmp –mkl	Option of the compiler	-xHost -O3 -fPIC -DNDEBUG -DADD_ -fp-model strict -qopenmp -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 –lpthread -lcublas -lcudart -lcusparse
Execution conditions	OpenMP, on 20 core of 1 CPU	Execution conditions	OpenMP, on 20 core of 1CPU, 1GPU

Implementation of the computation

Split matrices are computed with below 9 implementations (dgemm and 8 SpMV) on CPU and 4 implementations (dgemm and 3 SpMV)on GPU.

A split matrix of ***A*** is transformed into CRS or ELL when its sparsity is 90 or more, and a pair of matrices including this matrix are computed with SpMV or SpMM. (a split matrix of ***B*** is not transforemed)

A split matrix of ***A*** is computed with dgemm on CPU when its sparsity is less than 90.

- Implementations on CPU
  1. dgemm
  2. SpMV with CRS (internal parallelism)
  3. SpMV with CRS (external parallelism)
  4. SpMV with CRS (multiple right-hand-sides, internal parallelism)
  5. SpMV with CRS (multiple right-hand-sides, internal parallelism, blocking)
  6. SpMV with ELL (internal parallelism)
  7. SpMV with ELL (external parallelism)
  8. SpMV with ELL (multiple right-hand-sides, internal parallelism)
  9. SpMV with ELL (multiple right-hand-sides, internal parallelism, blocking)
- Implementations on GPU
  1. dgemm
  2. SpMV with CRS
  3. SpMV with ELL
  4. SpMM (sparse matrix-matrix multiplication) with CRS

Machine learning model

- We constructed xgboost (ver.1.4.2) classifier as Machine learning (ML) model to predict which implementation is the fastest.
- 7 explanatory variables:
  1. Matrix size
  2. Sparsity of matrix before that is split (the percentage of 0 in ***A***)
  3. The maximum element of ***A***
  4. The minimum element of ***A***
  5. Number of sparse split matrices of ***A*** (We treat a matrix which has 90% or more sparsity as a sparse matrices and which does NOT as a dense matrix)
  6. Number of dense split matrices of ***A***
  7. Number of sparse split matrices of ***B***
- Objective variable: The fastest implementation.

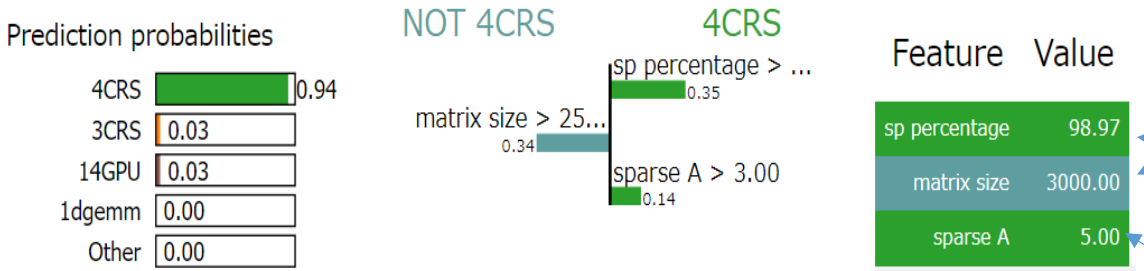
Data

- Number of training data: 199
- Number of test data: 23
- Matrix size: 1000,1500,2500,3000,3500,4000
- Input data:
  1. Matrices whose elements are generated in range of 0 to 1, and values of pow (10, rand()%Φ) are inserted for some sparsity elements. (the upper limit is Φ=15);
  2. A unit matrices in which values in the range 0 to 1 are inserted for some sparsity elements. (sparsity 90 to 98);
  3. A unit matrices in which values pow(10, rand()%Φ) are inserted for some sparsity elements. (upper limit is Φ=15);

A result of prediction by ML model

- Accuracy of the model prediction is from 83 % to 96 %, according to how the data is taken.
  - Low accuracy may be caused by the number of data, which is a little.

Preliminary result

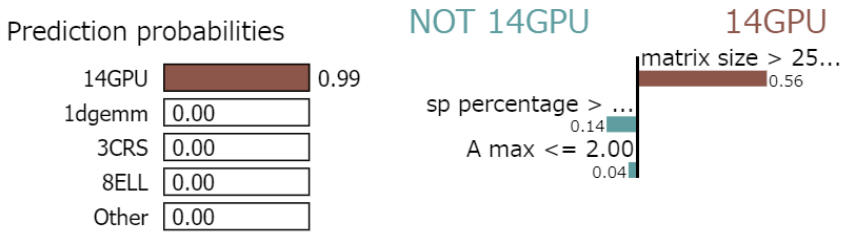


- Positive factor: sparsity of ***A***, the number of sparse split matrices of ***A.***
- Negative factor: matrix size.

We can know high sparsity has a large positive effect on SpMV with CRS. (multiple right-hand-sides, internal parallelism)
In addiction, we can know large matrix size has a large negative effect on it.
In fact, this implementation is faster in such as cases, thus these are reasonable explanations.

- Explanatory variables:
  - Matrix size: 3000
  - Sparsity of ***A***: 98.97%
  - The maximum element of ***A***: 1.00000000e+29
  - The minimum element of ***A***: 0.0
  - Number of sparse split matrices of ***A***: 5
  - Number of dense split matrices of ***A***: 0
  - Number of split matrices of ***B***: 6
- Actual result.
  - SpMV with CRS. (multiple right-hand-sides, internal parallelism)
- Prediction by ML model.
  - SpMV with CRS. (multiple right-hand-sides, internal parallelism) (correct)

Preliminary result

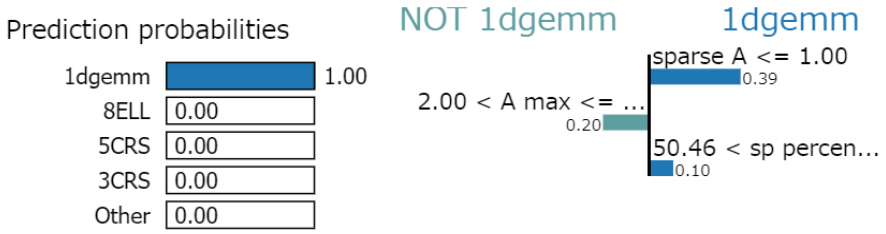


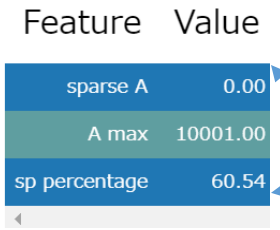
- Positive factor: matrix size.
- Negative factor: sparsity of ***A***, the maximum element of ***A.***

We can know large matrix size has a large positive effect on SpMM with CRS.
We can know relatively low sparsity that is 90 or more has negative effect on SpMM with CRS.
In fact, this implementation is faster in such as cases, thus these are reasonable explanations.

- Explanatory variables:
  - Matrix size: 3500
  - Sparsity of ***A***: 94.00%
  - The maximum element of ***A***: 1.0
  - The minimum element of ***A***: 0.0
  - Number of sparse split matrices of ***A***: 3
  - Number of dense split matrices of ***A***: 0
  - Number of split matrices of ***B***: 5
- Actual result
  - SpMM with CRS.
- Prediction by ML model
  - SpMM with CRS. (correct)

Preliminary result





- Positive factor: number of sparse split matrices of ***A***, sparsity of ***A.***
- Negative factor: the maximum element of ***A.***

We can know the number of sparse split matrices of ***A*** has large positive effect on dgemm on CPU.
It is a reasonable explanation because matrices are computed with dgemm when the number of sparse split matrices of ***A*** is 0.

- Explanatory variables
  - Matrix size: 1000
  - Sparsity of ***A***: 60.54%
  - The maximum element of ***A***: 10001.00
  - The minimum element of ***A***: 0.0
  - Number of sparse split matrices of ***A***: 0
  - Number of dense split matrices of ***A***: 1
  - Number of split matrices of B:1
- Actual result
  - Dgemm on CPU
- Prediction by ML model
  - Dgemm on CPU (correct)

Conclusion and feature work

- LIME gives us reasonable explanations about the fastest implementation in this research.
- The result includes a possibility that a software auto-tuning (AT) mechanism with AI to reduce the man-hours required for performance tuning in the field of numerical computations.

- Feature work
  - Adapting LIME or SHAP (that can explain same as LIME) to search procedure of performance parameters.
    - For example: blocking, the threshold sparsity of performing with SpMV or not, and so on