SGEMM Product Analysis

Roger Pujol
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
roger.pujol.torramorell@est.fib.upc.edu

ABSTRACT

General Matrix Multiplications (GEMM) are a key component in the rising world of Deep Learning. Most of the computational complexity of Deep Neural Networks can be reduced to mutliple huge GEMMs. Since any small optimization in GEMM can affect positively in many applications, our objective in this project will be to find how the parameters affect the performance of this kind of operations. Also we will try to get a regressor model capable to predict the time that will be needed to compute the GEMM given the parameters used.
The code of this project is Open Source and can be found in: https://github.com/rogerpt32/adm_paper1

1 INTRODUCTION

2 DATA SET

The dataset used is from the UCI Machine Learning Repository [1] [2]. This data set measures the running time of a matrix-matrix product A*B = C, where all matrices have size 2048 x 2048, using a parameterizable SGEMM (the "S" stands for "Single precision", meaning 32 bits floating point) GPU kernel with 241600 possible parameter combinations. For each tested combination, 4 runs were performed and their results are reported as the 4 last columns. All times are measured in milliseconds*.
There are 14 parameter, the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary. Out of 1327104 total parameter combinations, only 241600 are feasible (due to various kernel constraints). This data set contains the results for all these feasible combinations.
The values of the data set were taken using a desktop workstation running Ubuntu 16.04 Linux with an Intel Core i5 (3.5GHz), 16GB RAM, and a NVidia Geforce GTX 680 4GB GF580 GTX-1.5GB GPU. The tool used was the "gemm_fast" kernel from the automatic OpenCL kernel tuning library "CLTune".

2.1 Attribute information

Independent variables:

(1) MWG: per-matrix 2D tiling at workgroup level: 16, 32, 64, 128 (integer)
(2) NWG: per-matrix 2D tiling at workgroup level: 16, 32, 64, 128 (integer)
(3) KWG: inner dimension of 2D tiling at workgroup level: 16, 32 (integer)
(4) MDIMC: local workgroup size: 8, 16, 32 (integer)
(5) NDIMC: local workgroup size: 8, 16, 32 (integer)
(6) MDIMA: local memory shape: 8, 16, 32 (integer)
(7) NDIMB: local memory shape: 8, 16, 32 (integer)
(8) KWI: kernel loop unrolling factor: 2, 8 (integer)
(9) VWM: per-matrix vector widths for loading and storing: 1, 2, 4, 8 (integer)
(10) VWN: per-matrix vector widths for loading and storing: 1, 2, 4, 8 (integer)
(11) STRM: enable stride for accessing off-chip memory within a single thread: 0, 1 (categorical)
(12) STRN: enable stride for accessing off-chip memory within a single thread: 0, 1 (categorical)
(13) SA: per-matrix manual caching of the 2D workgroup tile: 0, 1 (categorical)
(14) SB: per-matrix manual caching of the 2D workgroup tile: 0, 1 (categorical)

Output:

(15) Run1: performance times in milliseconds for an independent run using the parameters described in the previous attributes.
(16) Run2: another independent run (see Run1).
(17) Run3: another independent run (see Run1).
(18) Run4: another independent run (see Run1).

The range of the running times is between 13.25 and 3397.08.

3 PREPROCESS

Since this dataset is very synthetic because of its computational nature, there isn't much things to preprocess. The only thing that we have done to preprocess the data, has been to separate each Run in a unique row, which only has one Run. This means that after the pre process we have 3 less columns but 4 times more rows. This new dataset have a lot of redundand data compared to the original version but is easier to handle, since now there is only one target variable.

4 ANALYSIS

In this section we will do a very brief analysis of the impact of the variables to the target variable (Run). To do this analysis, we have plots with all the dataset points where the Y-axis is always the target variable and then in each plot the X-axis is a different variable.
With this we can see that MWG 1a and NWG 1b have a big impact where a smaller values will always have small results but bigger values have a bigger variance in the results. This same effect can be seen but less clearly in KWG 1c, KWI 1h, SA 1m and SB 1n, which could mean that this variables are a bit less important and depend more on other variables. Then in MDIMC 1d and NDIMC 1e have the opposite impact to MWG and NWG, here the bigger values implies small results but smaller values have bigger variance in the results.
Finally the other variables (MDIMA 1f, NDIMB 1g, VWM 1i, VWN 1j, STRM 1k and STRN 1l) doesn't seem to have a direct implication since for any value have a similar distribution in the target. This

doesn't mean that this variables are useless, this means that their value alone doesn't clarify anithing about the target but it might be meaningful only when combined with other variables.

5 REGRESSION LINE MODEL

The first approach that we used to try to predict the execution time of the SGEMM given the parameters, is a simple regression line.

5.1 Implementation

To get the model, first we split the model in two blocks: one for training (in this case 90% of the data set) and the other to test (in this case 10% of the data set) the accuracy later. Then we use the training part to fit the regression line model. Finally we use that model to predict the execution times of the test examples and we compare the results with the real values.

5.2 Results

The results from the Regression Line are not very satisfying. The mean square error that this model has in the training data is 80322.28, and obviously the mean square error in the testing data is even worse with a value of 82841.39. This lack of accuracy using a linear regression, might probably mean that our data is not linear. If we check the variance using the R^2-score, where a 1 means that the data fits perfectly to the regression line with no variance and a 0 means the opposite, we get a 0.41 in both train and test data. This confirms that this regression might not be a good solution.
But in order to benefit from this model, we can take a look to the coefficients that define the model. This coefficients multiply the correspondent input variables described in the Data Set section.

- 3.3280371
- 3.07243576
- 5.1729495
- -16.67627391
- -16.33800475
- 1.01720931
- 1.02988438
- 3.98709287
- -1.52596645
- -3.01457498
- -9.44404773
- -0.18646408
- 37.88273766
- 46.99629411

With this we can see that as we predicted in the analysis of the data: MWG, NWG, KWG, KWI, SA and SB have a positive impact in the target (bigger input means bigger execution time), whereas MDIMC and NDIMC have a negative impact in the target (bigger input means smaller execution time).

6 DECISION TREE REGRESSOR MODEL

The second approach to fit the data to a model is using decision trees.

6.1 Implementation

The implementation of this method has followed a similar steps to the Regression Line Model. First we split the data in training (75%) and test (25%). Then with the training data we fitted two decision tree models, one without any restrictions and another with the depth limited to 5. After that we test the accuracy of both models with training and test data (separately).

6.2 Results

	Full DT model	DT max depth 5
Max Depth	25	5
Node Count	481085	63
MSE train	3.55	31092.63
R^2 train	1	0.77
MSE test	8.39	30345.53
R^2 test	1	0.77

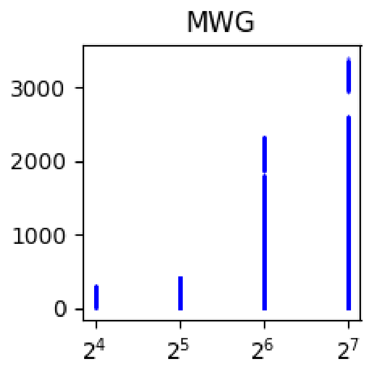
Table 1: Results from the decision tree models

The model trained without restrictions have a great accuracy since the mean square error in training is only 3.55 and in test is 8.39. But this impressive accuracy is basically consequence of a huge overfitting. If we check the depth and the number of nodes of the model (see table 1), we can see that basically the model has probably almost one leaf for each possible configuration. Technically since we have all the possible configurations encoded in this decision tree, the results will be always accurate, but if we want to extract useful information from this model it will be very hard.
To analyse easily the impact of the variables we use the model trained with the maximum depth fixed to 5. This restriction reduce the depth from the previous model to 5 and the node count to only 63. Taking a look to the values of each decision node (see the file *doc/tree/tree.pdf* from the githb repository given in the abstract) we can see that only a few variables are used. The variables used are: MWG, NWG, MDIMC, NDIMC and SA. These confirm again what we saw at the analysis of the data where the mentioned variables were the ones with more importance on the final result. The accuracy of this reduced model is worse than the full one (see table 1), which is obvious since we are only using 5 variables and not checking all the configurations, but still it has a better accuracy than the simple regression line.

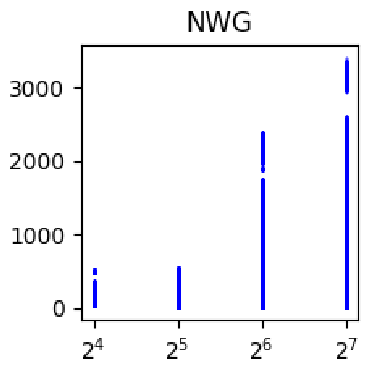
7 CONCLUSIONS

REFERENCES

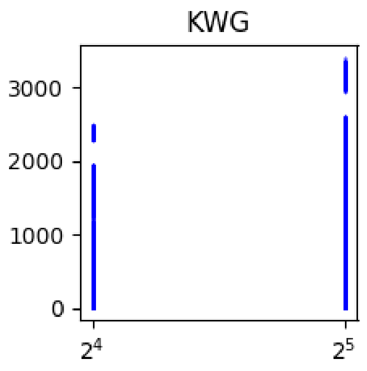
[1] E. G. Paredes and R. Ballester-Ripoll. Sgemm gpu kernel performance data set, 2018.
[2] C. Nugteren and V. Codreanu. Cltune: A generic auto-tuner for opencl kernels. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 195–202, Sep. 2015.



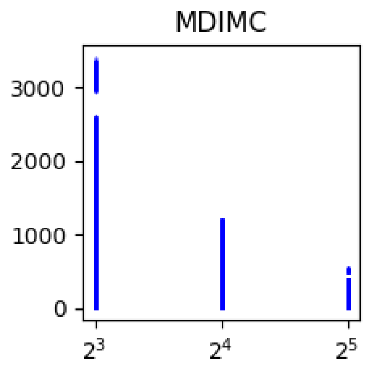
(a) MWG



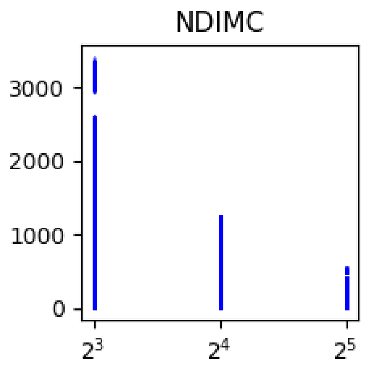
(b) NWG



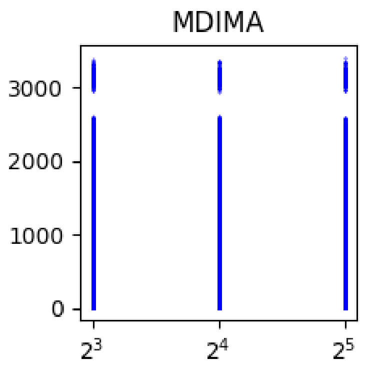
(c) KWG



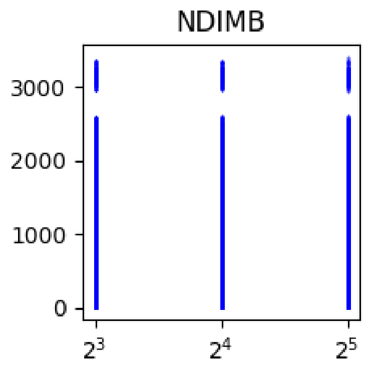
(d) MDIMC



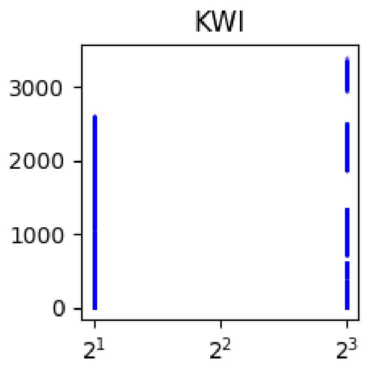
(e) MDIMC



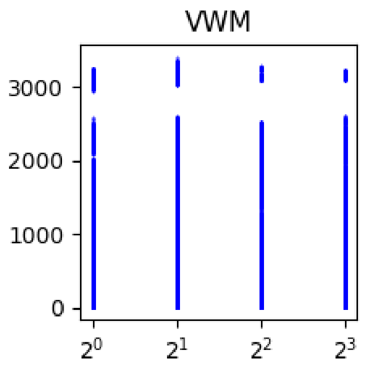
(f) MDIMA



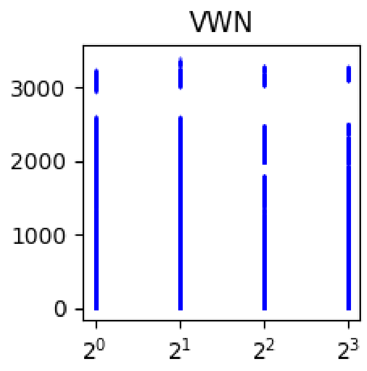
(g) NDIMB



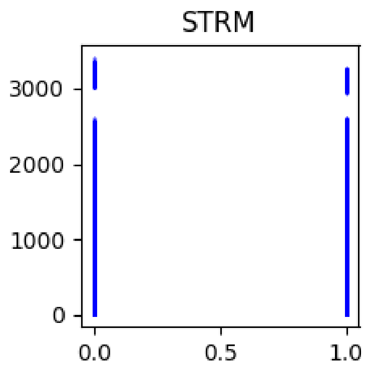
(h) KWI



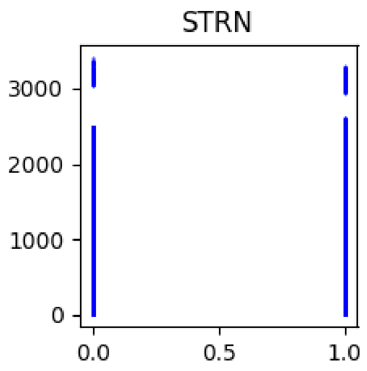
(i) VWM



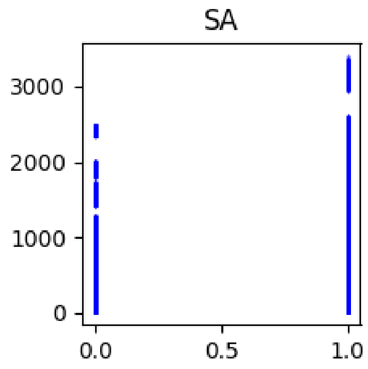
(j) VWN



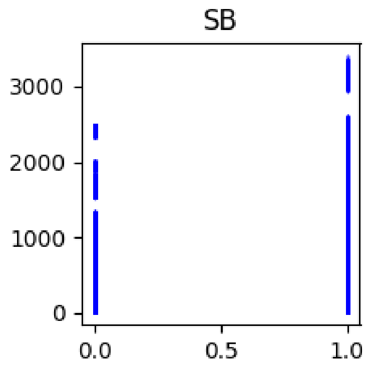
(k) STRM



(l) STRN



(m) SA



(n) SB

Figure 1: Plots of all variables contrasted with the target variable. Note that in most of the plots the X-axis is logarithmic.