Real World HPC Systems for Big Data/AI Research

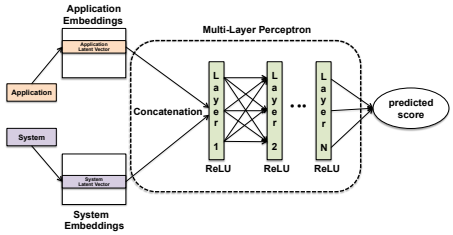
Learning Neural Representations for Predicting GPU Performance[1]

Problem

- Modeling performance of applications across systems with different GPU microarchitectures

Proposal

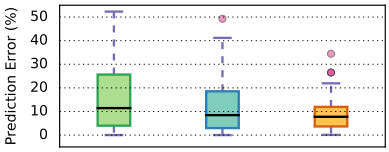
- A collaborative filtering based matrix factorization (MF) model to automatically learn latent features describing performance of applications on systems
- A multi-layer perceptron (MLP) to model complex non-linear interactions between applications and systems



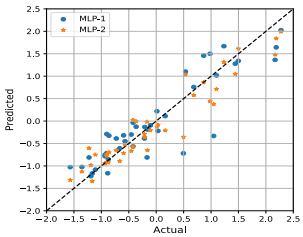
Multi-layer perceptron model using latent features

Results

- MLP with 2 layers (MLP-2) achieves 90.6% accuracy when predicting instructions per second (IPS)



Accuracy of the models using IPS dataset



Actual vs. Predicted normalized IPS values

Efficient 2D Convolution on CUDA-enabled GPUs[2]

Problem

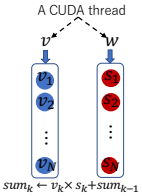
- 2D convolution computation on CUDA-enabled GPUs

Proposal

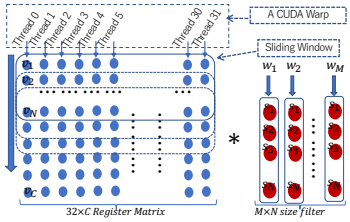
(1) Using registers as a cache
- Lower latency, high throughput
- High data reuse

(2)Computing partial sums in parallel

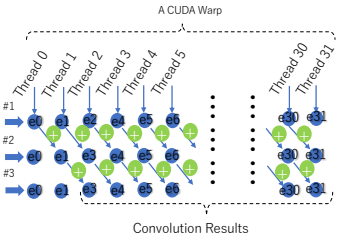
(3)Transferring partial sums in parallel
- No stalling within thread
- Efficient communication between threads by shuffle



(2) Compute partial sums

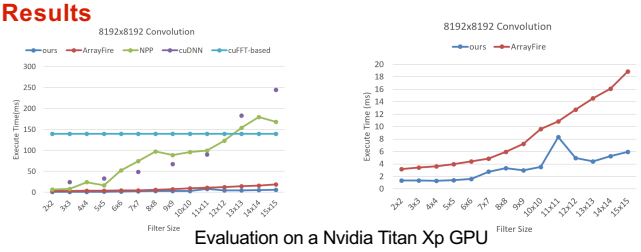


(1) Register Cache (32×C)



(3) Transfer partial sums

Results

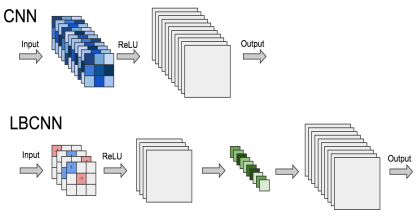


Evaluation on a Nvidia Titan Xp GPU

Can Local Binary Convolutions Make Neural Networks Models Smaller?[3]

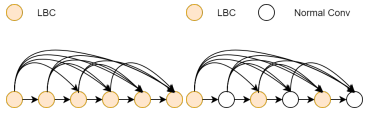
Background

- Local Binary Convolution Neural Networks(LBCNN)
  - models use Local Binary Convolution(LBC) layers
  - (+)less learned parameters
  - (-)less accuracy



Problem

- Not applicable to Large Model
  - Too slow
  - Too much Acc. loss

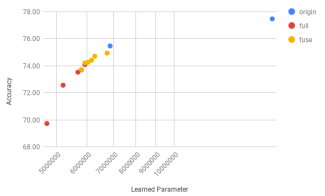


Proposed Method

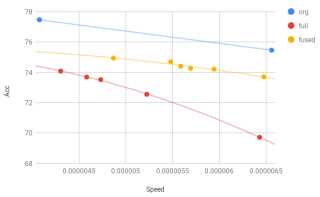
Only replace half of the normal convolution layer by LBC

Results

Results of models for different number of difference maps generated by LBC filters



Accuracy and Speed of different models



Framework for Transpliation Between Python and Fortran[4]

Problem

Fortran
+ top performance
+ HPC legacy
- hard to maintain

Python
+ ease of programming
+ general-purpose tools
- big runtime overhead

Dillema
- Performance or ease-of-programming?

Solution
- Python for development, Fortran at runtime.

take legacy Fortran
- reuse legacy code
- keep battle-tested implementations
- will be translated by our framework

migrate to Python
- happens once
- semi-automatic or automatic
- performance-critical data is retained (via Type Hints)
- user can easily extend functionality

translate performance-critical kernels to Fortran
- JIT, at runtime
- fully automatic
- original performance is retained
- user doesn't interact with Fortran

Results

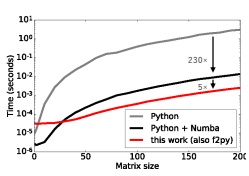


Figure 1

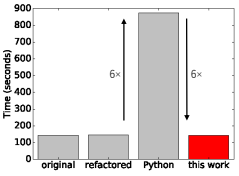


Figure 2

Figure 1: DGEMM performance the same as Fortran. 5× better than Numba.

Figure 2: Migrated Miranda IO benchmark retains original performance.

[1] Shweta Salaria, Aleksandr Drozd, Artur Podobas, Satoshi Matsuoka, Learning Neural Representations for Predicting GPU Performance, ISC'19

[2] Peng Chen, Mohamed Wahib, Shin'ichiro Takizawa, Ryousei Takano, Satoshi Matsuoka, Efficient Algorithms for the Summed Area Tables Primitive on GPUs. IEEE CLUSTER'18

[3] Haoyu Zhang, Mohamed Wahib, Satoshi Matsuoka, Can Local Binary Convolutions Make Neural Networks Models Smaller?, Submitted to ICPP'19 (Poster)

[4] Mateusz Bysiek, Aleksandr Drozd, and Satoshi Matsuoka. Migrating Legacy Fortran to Python While Retaining Fortran-level Performance Through Transpilation and Type Hints, 6th Workshop on Python for High-Performance and Scientific Computing, co-held with SC16