GAMUT: Matrix Multiplication-like Tasks on GPUs

Xincheng Xie, Junyoung Kim, Kenneth Ross

Department of Computer Science, Columbia University

Matrix Multiplication in Data Science

Matrix multiplication is commonly used in data science

```
P[i][k]: Weight vector of person i's taste
R[k][j]: Style vector of restaurant j
C[i][j]: How much person i prefers to eat at restaurant j
```

```
for(i = 0; i < M; i++)
 for(k = 0; k < K; k++)
  for(j = 0; j < N; j++)
    C[i][j] += P[i][k]*R[k][j];
```

Data science task using matrix multiplication to calculate people's preferences for eating at different restaurants

Variations of Matrix Multiplication in Data Science

```
P[i][k]: Weight vector of person i's taste
R[k][j]: Style vector of restaurant j
Pzip[i]: Zipcode of person i
Rzip[j]: Zipcode restaurant j
C[i][j]: How much people at zipcode i prefers to eat at
         restaurants at zipcode j
```

```
for(i = 0; i < M; i++)
 for(k = 0; k < K; k++)
  for(j = 0; j < N; j++)
    C[Pzip[i]][Rzip[j]] += P[i][k]*R[k][j];
```

Data science task using matrix multiplication to calculate people's preferences for eating at different restaurants, grouped by zipcode

Variations of Matrix Multiplication in Data Science

```
A[i][k]: Weight of observation i for feature k
B[k][j]: Stength of feature k at location j
thres[j]: Threshold at which to amplify high single products
R[i][j]: Weighted stength for each observation i at location j
```

```
for(i = 0; i < M; i++)
 for(k = 0; k < K; k++)
  for(j = 0; j < N; j++)
    R[i][j] += A[i][k]*B[k][j] +
     (A[i][k]*B[k][j]>thres[i])*(A[i][k]*B[k][j] - thres[i]);
```

ML task that amplifies high signals in matrix multiplication

Motivation

Variations of matrix multiplication are useful in data science

However, performing such tasks is difficult as

- Libraries only support a limited class of manually tuned computations
- Deep learning compilers require significant time for optimizations

Motivation

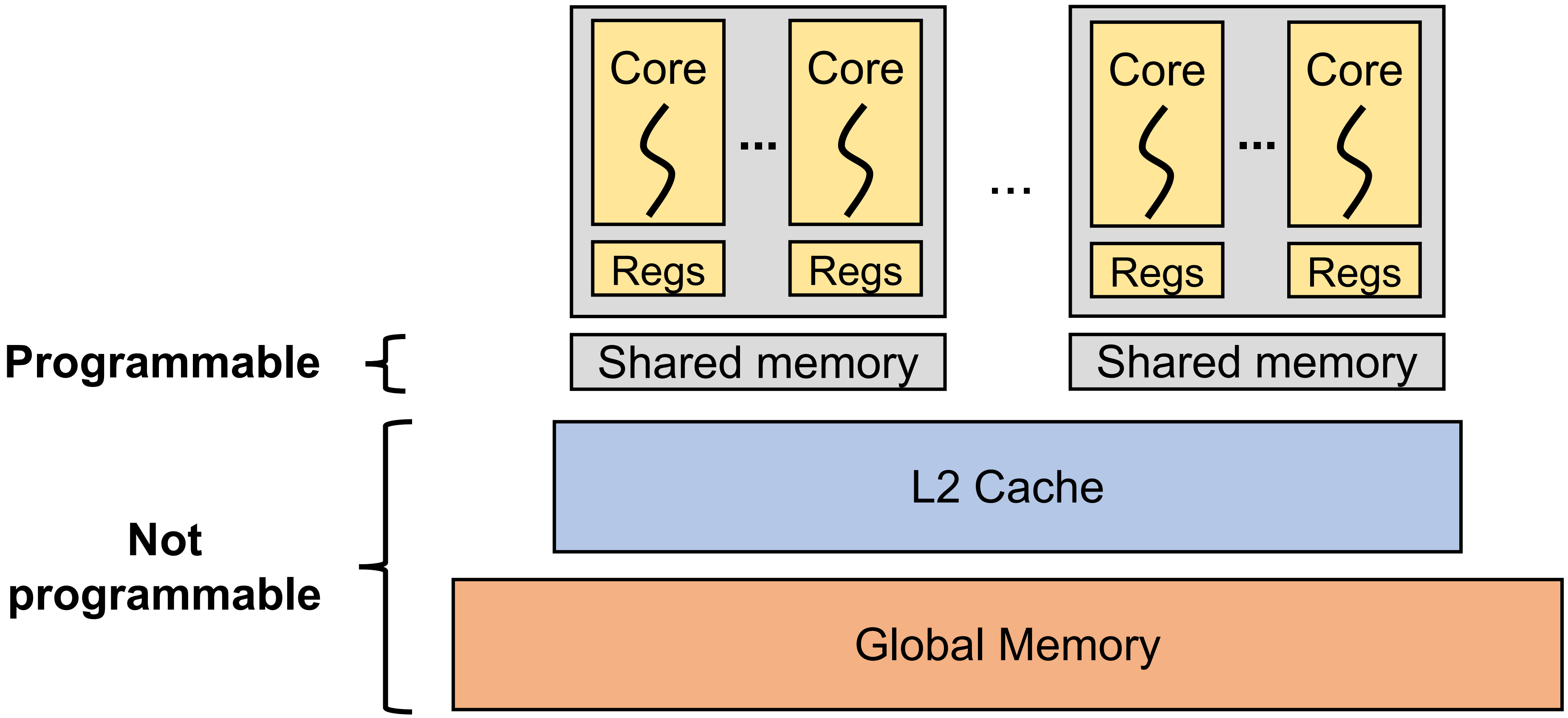
Variations of matrix multiplication are useful in data science

However, performing such tasks is difficult as
- Libraries only support a limited class of manually tuned computations
- Deep learning compilers require significant time for optimizations

If such tasks were easy and fast to execute, it would lead to **the discovery of more useful tasks and ML models**

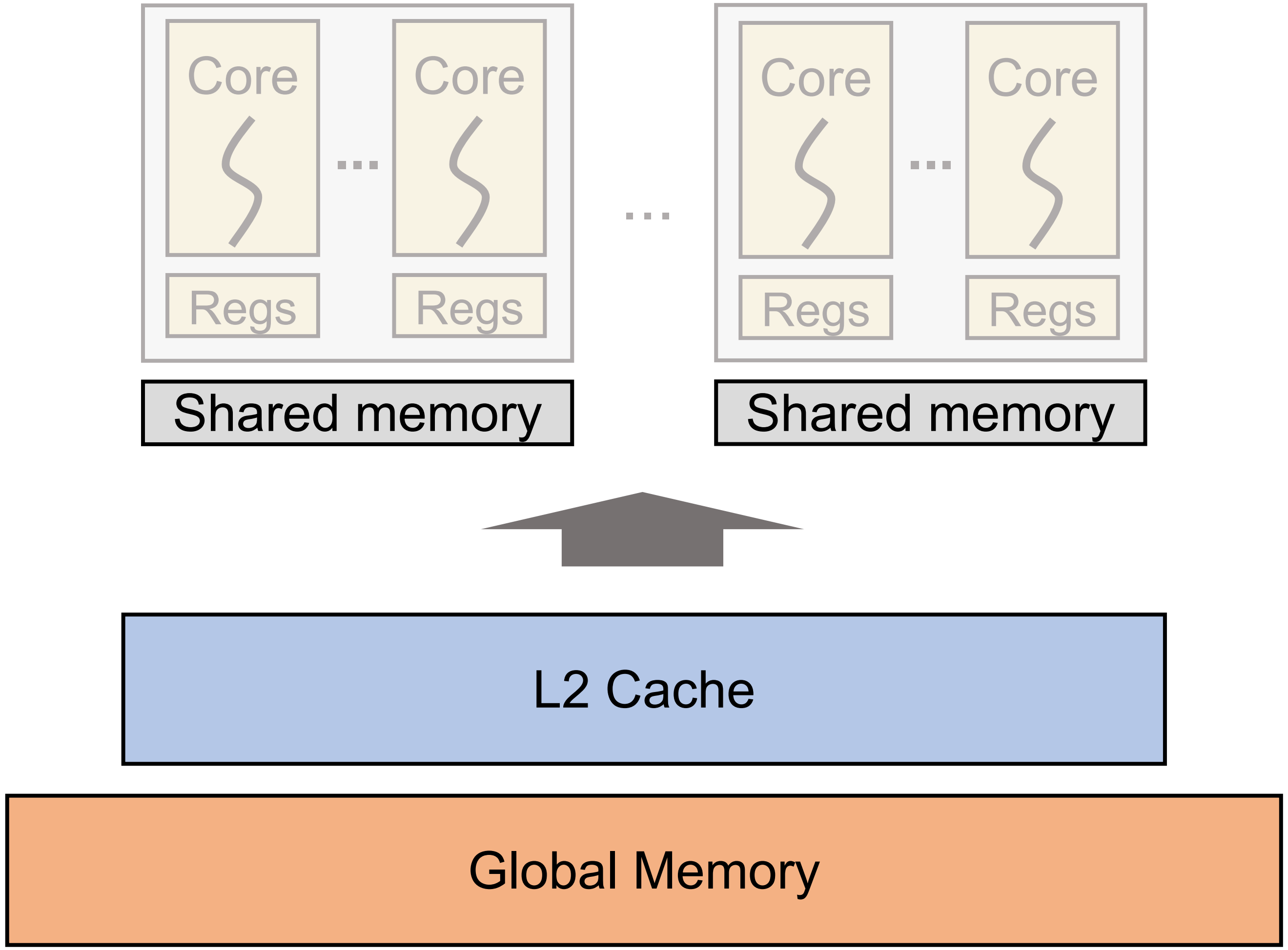
We propose **GAMUT,** a library that automatically **generates fast code for matrix multiplication-like tasks** for the GPU with **low compilation overhead.**

Matrix Multiplication for GPUs

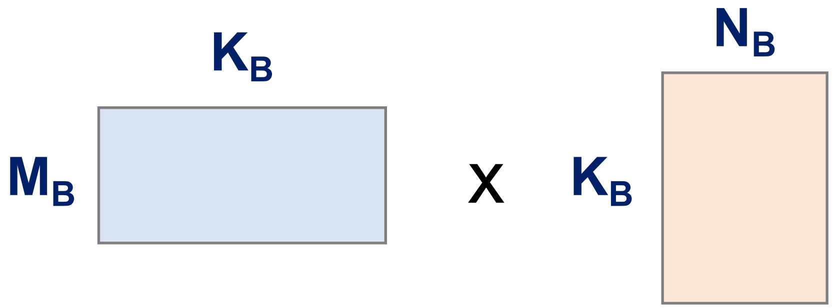


GPU Architecture

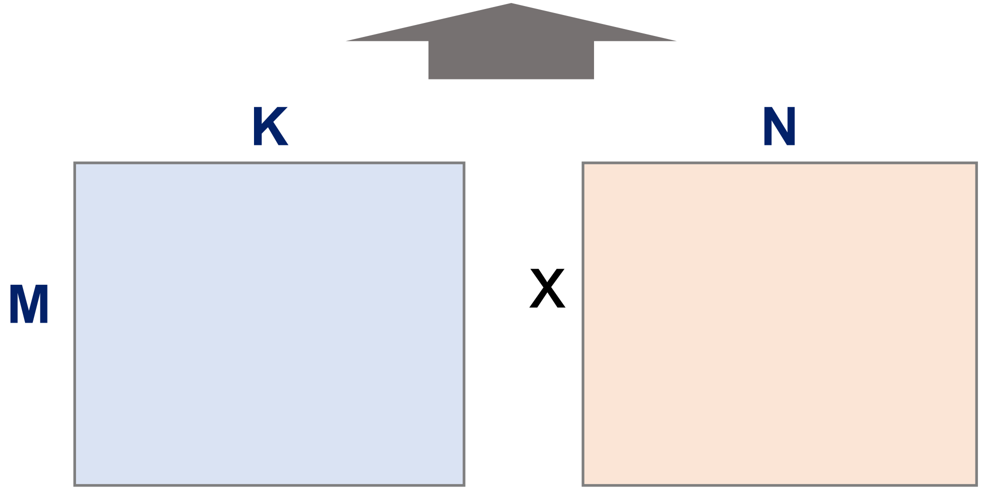
Matrix Multiplication for GPUs



GPU Architecture

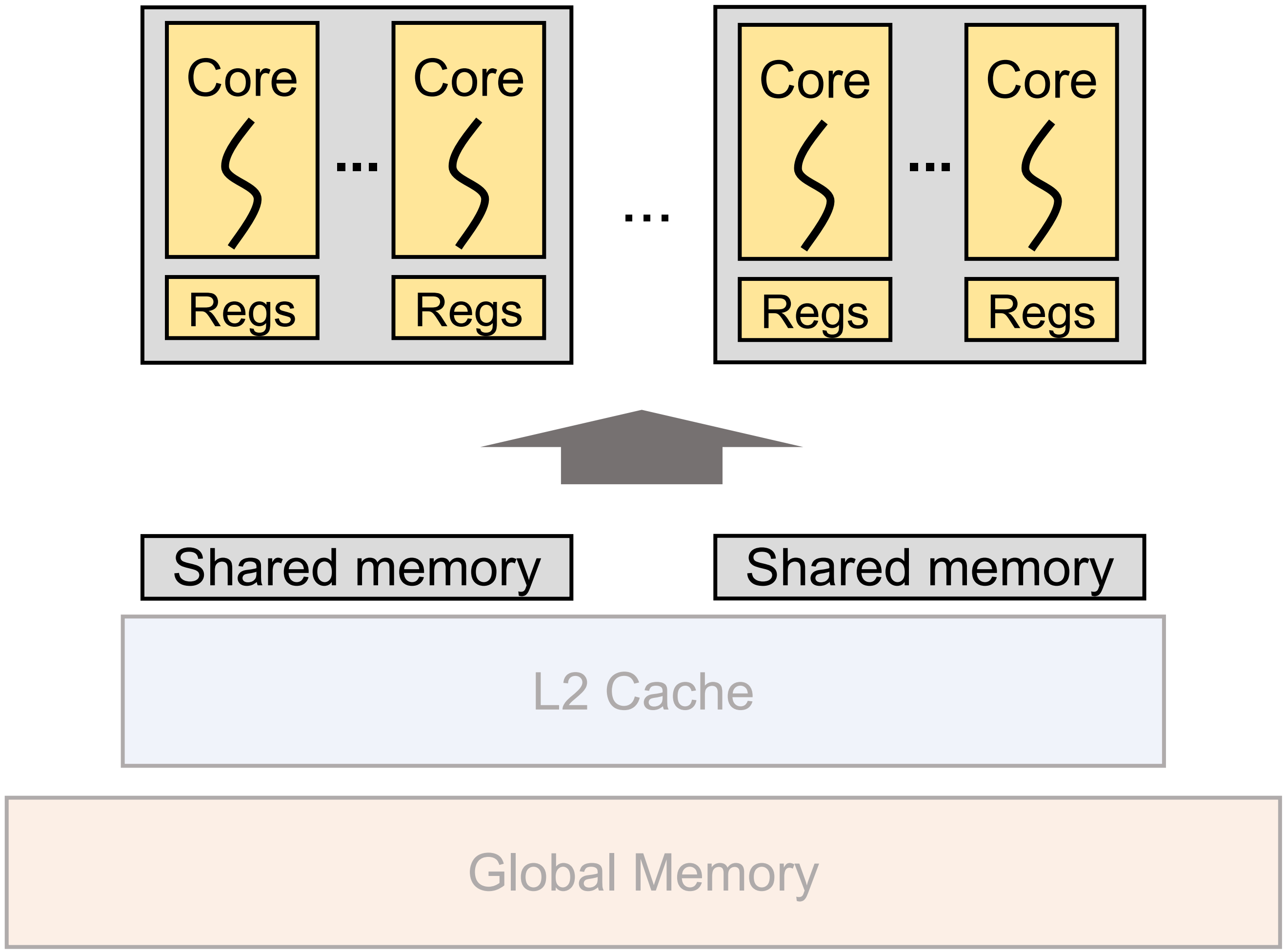


Matrix Multiplication (Shared memory)

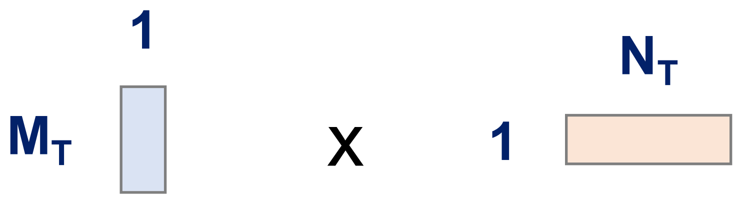


Matrix Multiplication (Global memory)

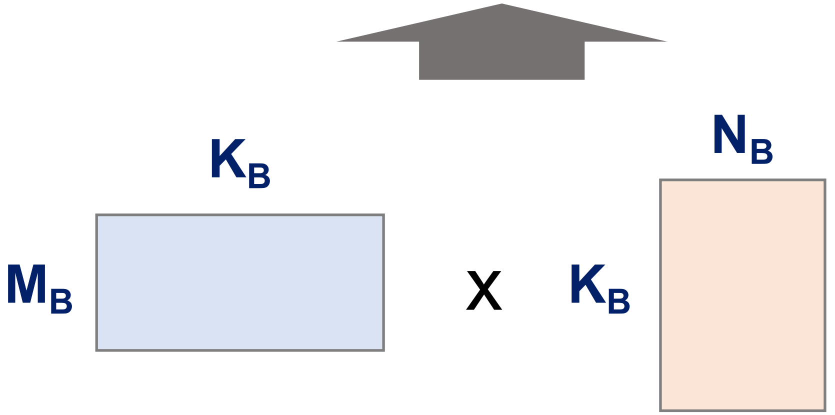
Matrix Multiplication for GPUs



GPU Architecture



Matrix Multiplication (Registers)



Matrix Multiplication (Shared memory)

Variations of Matrix Multiplication

Variations of matrix multiplication can be created in two ways.

1. Change the inner computation → Change loading process of MM

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[i])*(A[i][k]*B[k][j] - thres[i]);
```

2. Change how results are stored → Change storing process of MM

```
for(i = 0; i < M; i++) for(...) for(...)
  C[Pzip[i]][Rzip[j]] += P[i][k]*R[k][j];
```

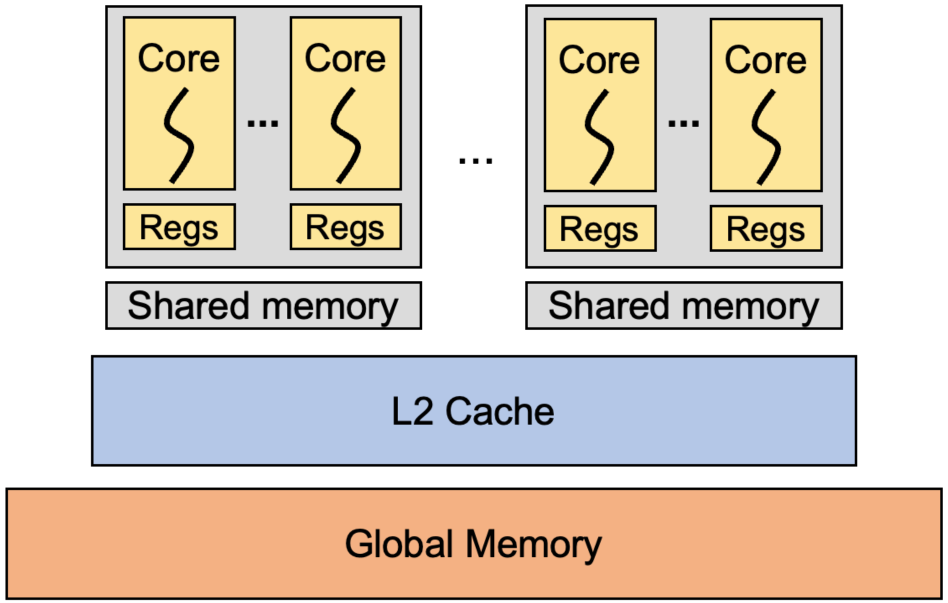
Changing Inner Computation

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[i])*(A[i][k]*B[k][j] - thres[i]);
```

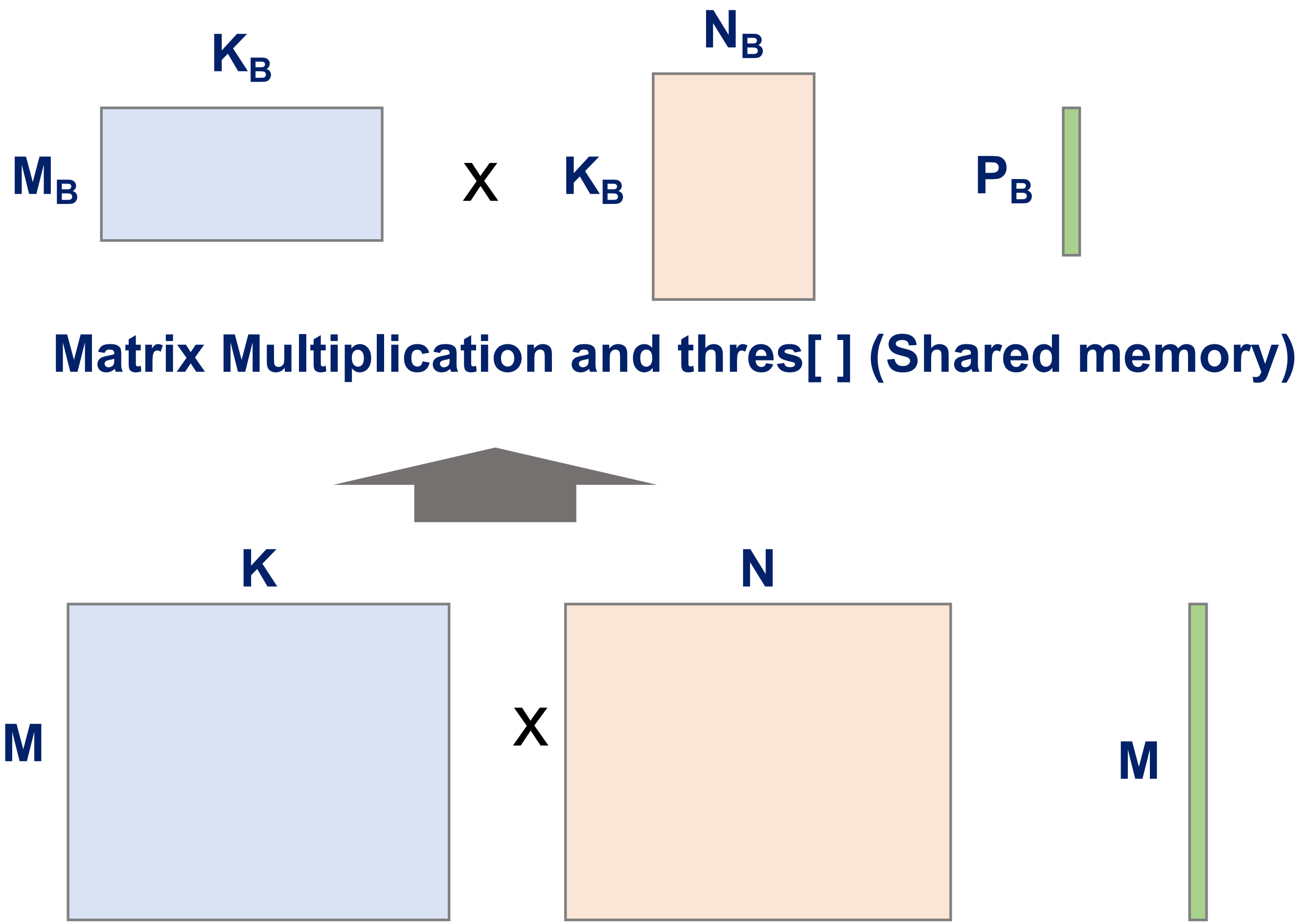
1. Parse inner computation and generate instructions

2. Load additional data used in computation (e.g. thres[i])
   → Use different loading strategy depending on how data is indexed (e.g. thres[j], thres[i][j])

Matrix Multiplication for GPUs

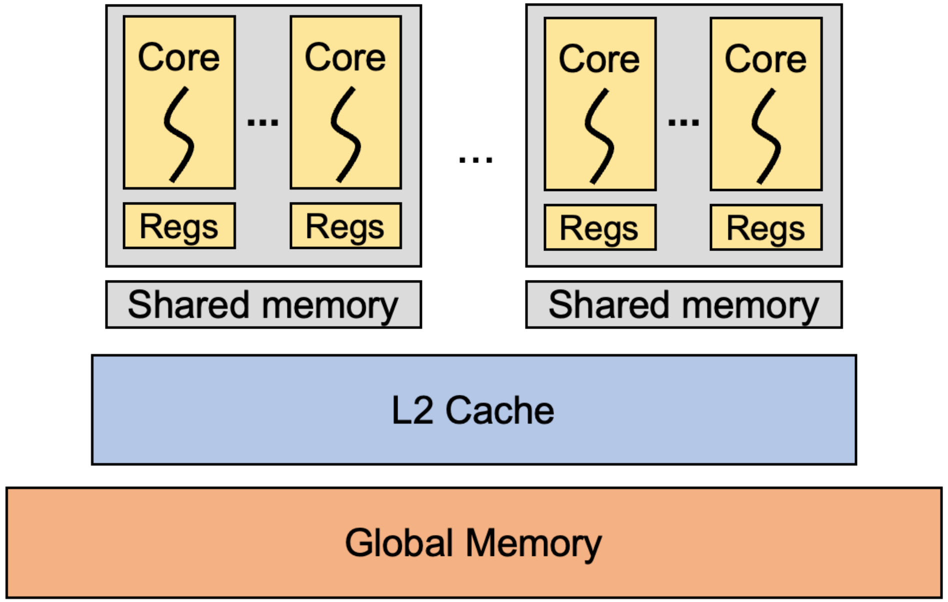


GPU Architecture

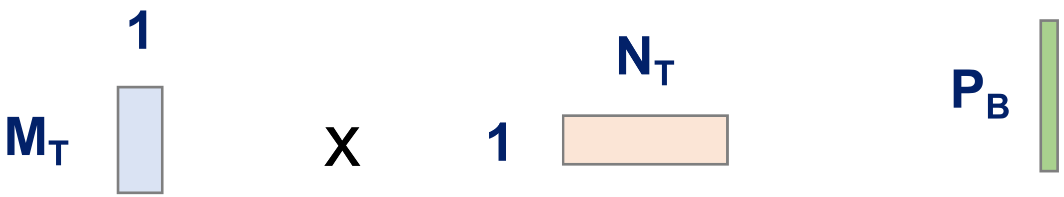


Matrix Multiplication and thres[ ] (Global memory)

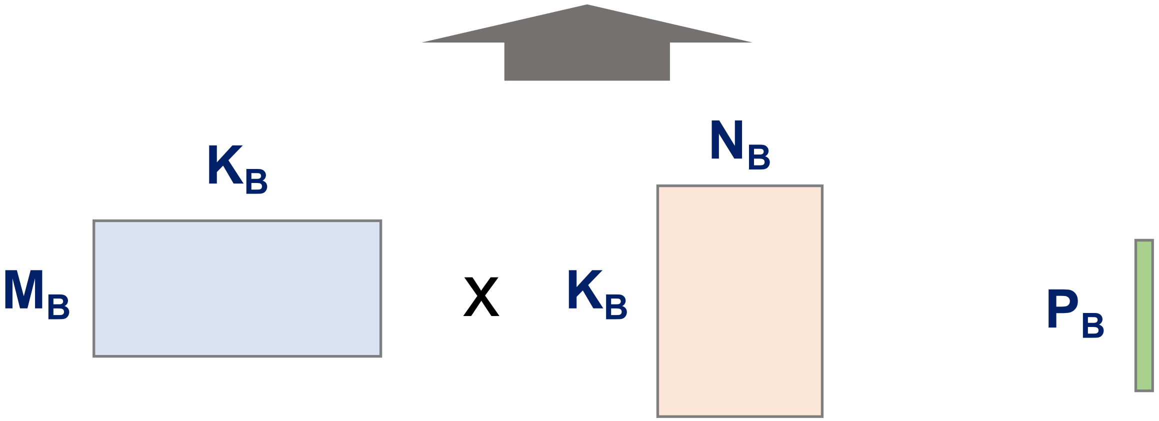
Matrix Multiplication for GPUs



GPU Architecture



Matrix Multiplication and thres[ ] (Registers)



Matrix Multiplication and thres[ ] (Shared memory)

Changing Inner Computation

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[i])*(A[i][k]*B[k][j] - thres[i]);
```

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[j])*(A[i][k]*B[k][j] - thres[j]);
```

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[i][j])*(A[i][k]*B[k][j] - thres[i][j]);
```

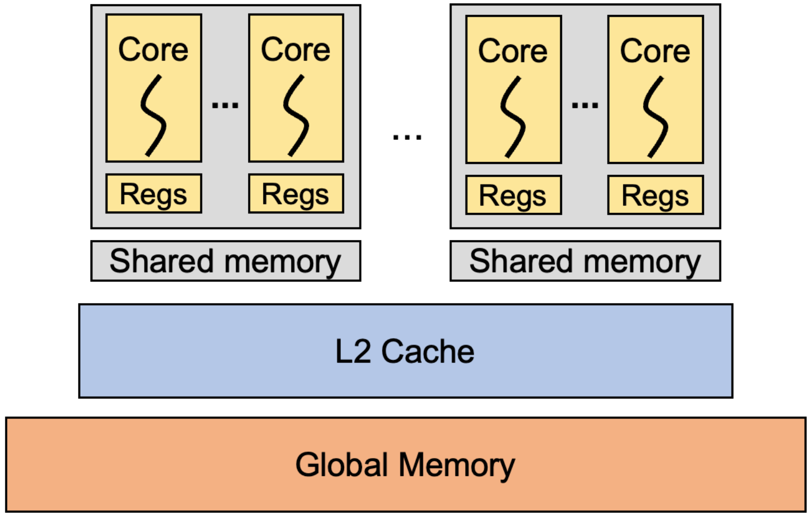
Changing Result Storage

```
for(i = 0; i < M; i++) for(...) for(...)
  C[Pzip[i]][Rzip[j]] += P[i][k]*R[k][j];
```

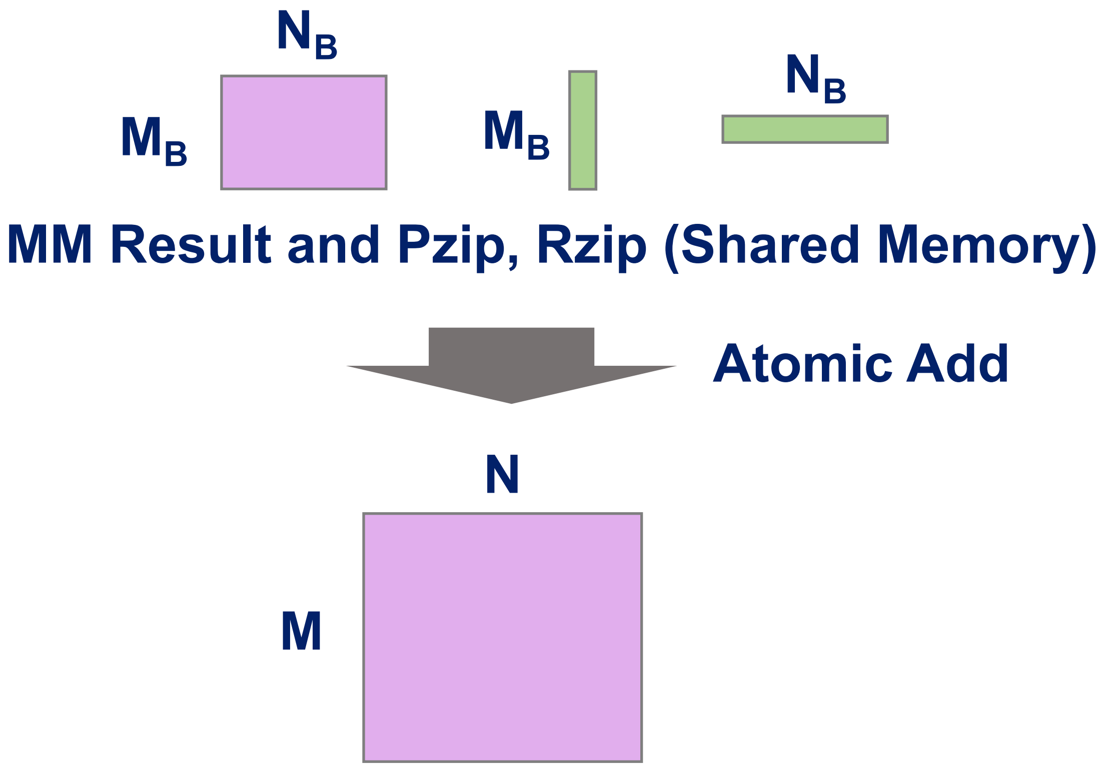
GAMUT recognizes how the results are written (e.g. using predetermined locations, to sparse array) and generates code accordingly.

Changing Result Storage

```
for(i = 0; i < M; i++) for(...) for(...)
  C[Pzip[i]][Rzip[j]] += P[i][k]*R[k][j];
```



GPU Architecture



Result Location (Global memory)

Changing Result Storage

```
for(i = 0; i < M; i++) for(...) for(...)
  C[Pzip[i]][Rzip[j]] += P[i][k]*R[k][j];
```

```
for(i = 0; i < M; i++) for(j = 0; j < N; j++)
  accum = 0;
  for(k = 0; k < K; k++)
    accum += P[i][k]*R[k][j];
  accum > thres ? C_sparse.add(accum)
```

```
for(i = 0; i < M; i++) for(j = 0; j < N; j++)
  accum = 0;
  for(k = 0; k < K; k++)
    accum += P[i][k]*R[k][j];
  min_heap_100.add(accum)
```

Parameter finding

Upon installation, GAMUT finds the optimal block sizes (M_b, N_b, K_b, M_t, ...) for **matrix multiplication** (done once).

When a new query is encountered, GAMUT **incrementally scales the tile sizes up or down** to fit the memory of the streaming processors.

The hash of the parse tree of the query, along with the block sizes, is saved so that **the same query can be executed immediately in the future.**

Baselines

cuBLAS, CUTLASS : Commonly used **matrix multiplication libraries** for the GPU

- Fast performance for matrix multiplication
- Unable to support matrix multiplication-like tasks in general

Apache TVM : Popular deep learning compiler, able to optimize DL workloads for a variety of hardware.

- Able to support tasks with different inner computations
- Unable to support tasks that change result storage without generating intermediate results

Experiment Results (Compilation)

```
for(i = 0; i < M; i++) for(...) for(...)
  C[i][j] += P[i][k]*R[k][j];
```

Standard Matrix Multiplication

Method	GAMUT	cuBLAS	CUTLASS	TVM
Compile Time	3.3s	1.7s	4.9s	2m 21s

Compilation time for matrix multiplication

Matrix order	1k	32k
TVM Compile Time	2m 21s	51m 33s

TVM Compilation time for matrix multiplication

Experiment Results (Compilation)

```
for(i = 0; i < M; i++) for(...) for(...)
  R[i][j] += A[i][k]*B[k][j] +
   (A[i][k]*B[k][j]>thres[i])*(A[i][k]*B[k][j] - thres[i]);
```

Matrix multiplication-like task

Method	GAMUT	cuBLAS	CUTLASS	TVM
Compile Time	3.6s	N/A	N/A	2m 29s

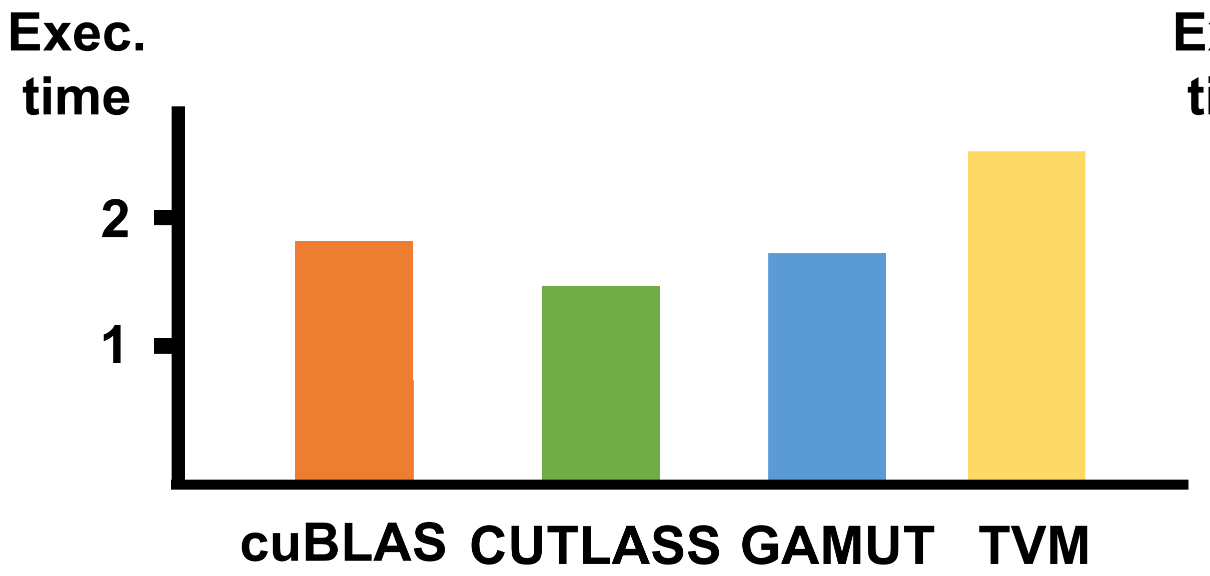
Compilation time for matrix multiplication-like task

Matrix order	1k	32k
TVM Compile Time	2m 29s	51m 17s

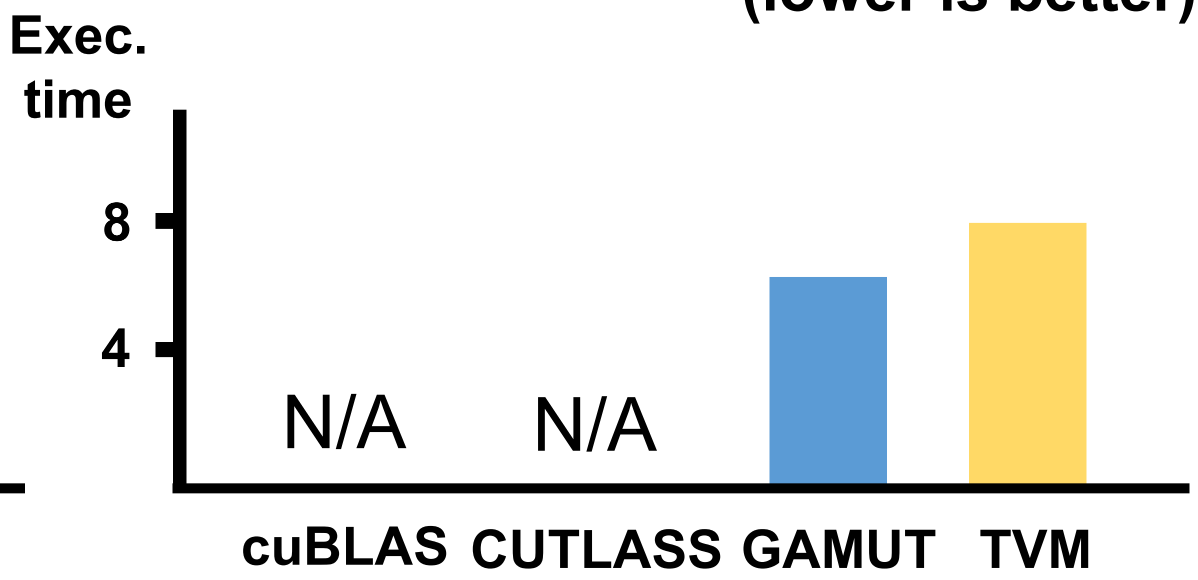
TVM Compilation time for matrix multiplication-like task

Experiment Results (Execution Time)

Matrix order : 16k
Unit : Seconds
(lower is better)



Standard Matrix Multiplication



Matrix Multiplication-like Task

Experiment Results Summary

	Performance	Compilation Time	Flexibility
Libraries (cuBLAS, CUTLASS)	Most performant	Low	Inflexible
DL compilers (TVM)	Less performant	High	Less flexible
GAMUT	Performant	Low	Flexible

Conclusion

GAMUT is a library that can **optimize matrix multiplication-like tasks for the GPU**. GAMUT has similar performance to state-of-the-art matrix multiplication libraries, while having faster compilation time, better performance, and more flexibility than deep learning compilers.

We expect GAMUT will **improve productivity for common data analysis tasks and facilitate research in the ML community** by allowing scientists to write simple code that is also very efficient.

https://github.com/xxcisxxc/GAMUT-release