Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions

Wolong: A Back-end Optimizer for Deep Learning Computation

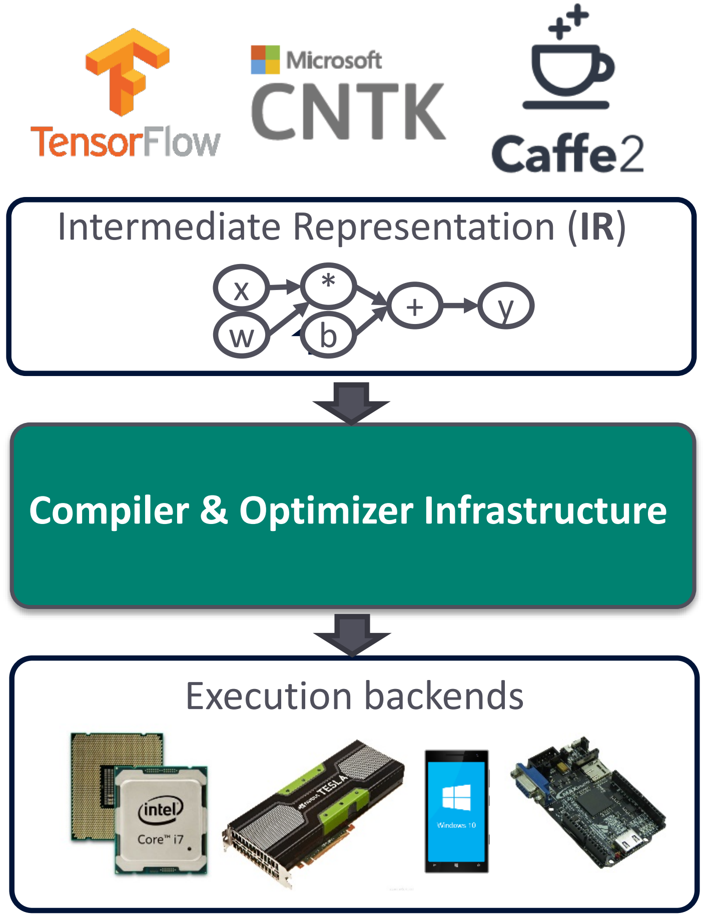
Jilong Xue

Researcher, Microsoft Research Asia

System Challenge in Deep Learning

- Innovations are emerging very fast in deep learning area
  - New DNN models and workload patterns
    - RNN, CNN, GAN, reinforcement learning, graph neural network, etc.
  - Diverse and emerging hardware accelerators,
    - GPU, FPGA, ASICs, edge devices, NV-Link, RDMA, etc.

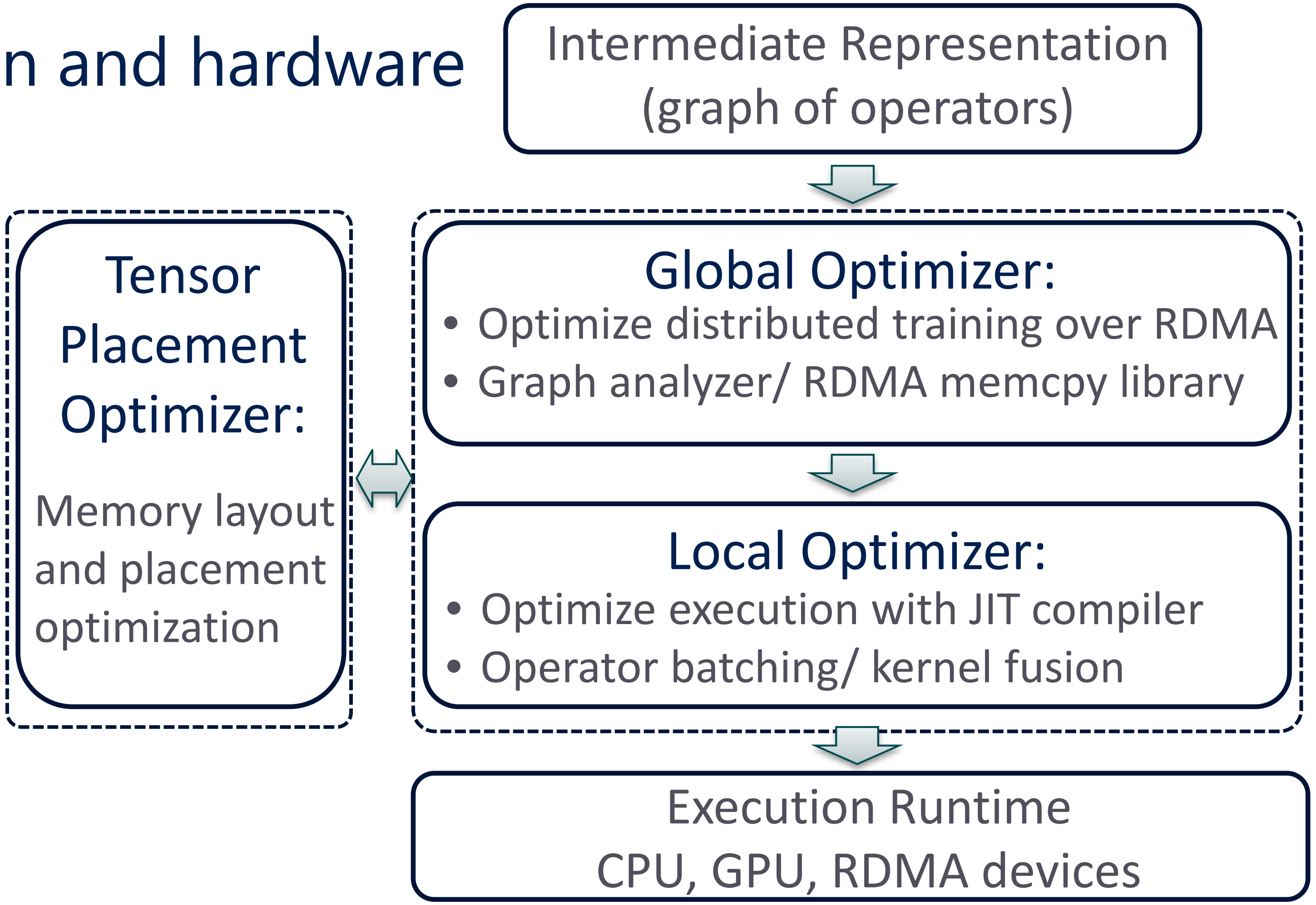
- Compiler stack is key to bridge framework and hardware
  - Combine information of computation graph and hardware
  - Optimize for both local execution and distributed scalability
  - Critical for both training and inference



Wolong: Optimizer Stack for Deep Learning

- System innovation to bridge application and hardware
  - General computation graph optimization
  - Software and hardware co-design
  - Just-in-time compiler

- Transparent optimization
  - Communication efficiency
  - Accelerator execution efficiency
  - Memory efficiency

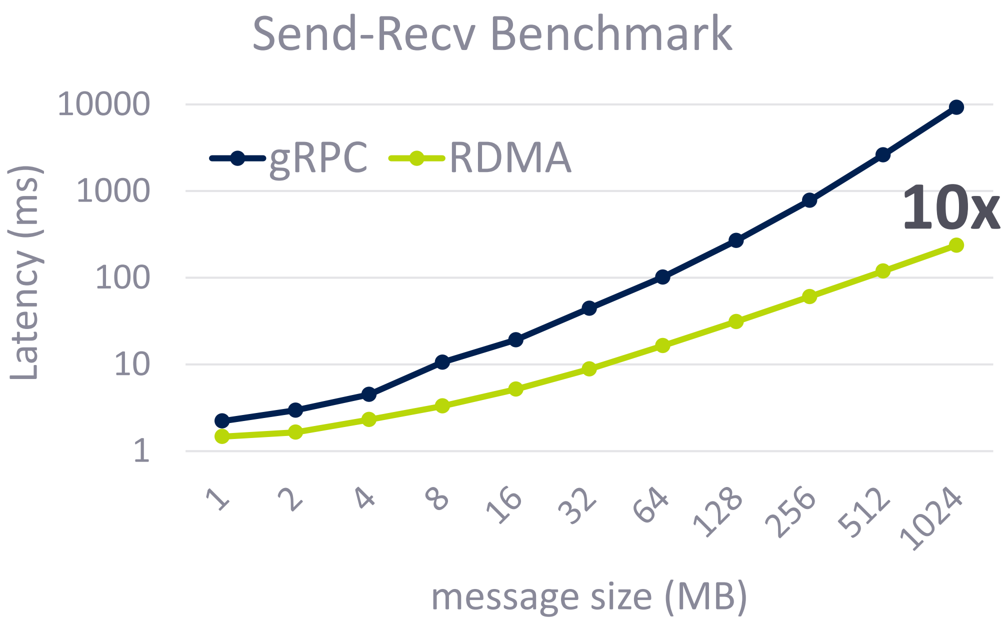


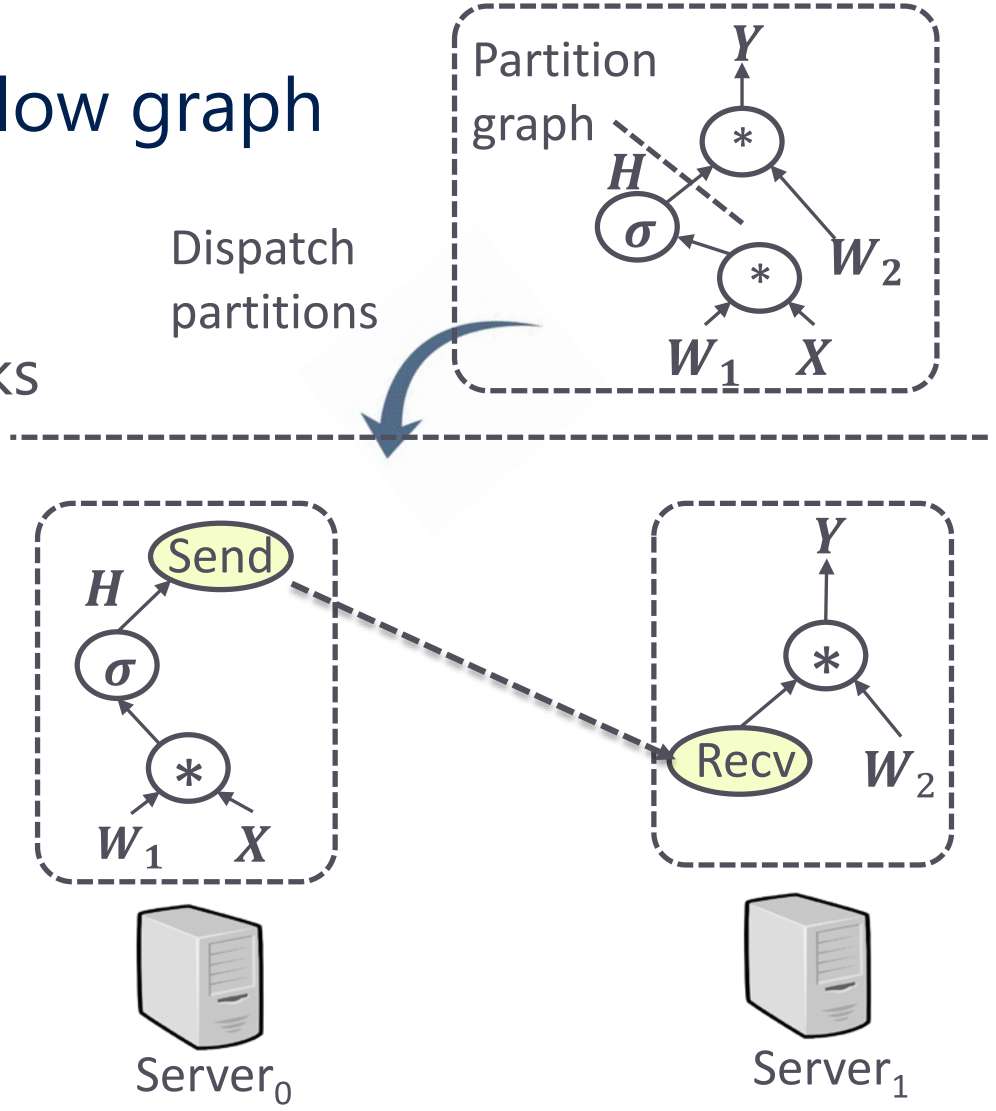
Global Optimizer

Fast Distributed Deep Learning Computation over RDMA

Distributed Dataflow Graph Execution

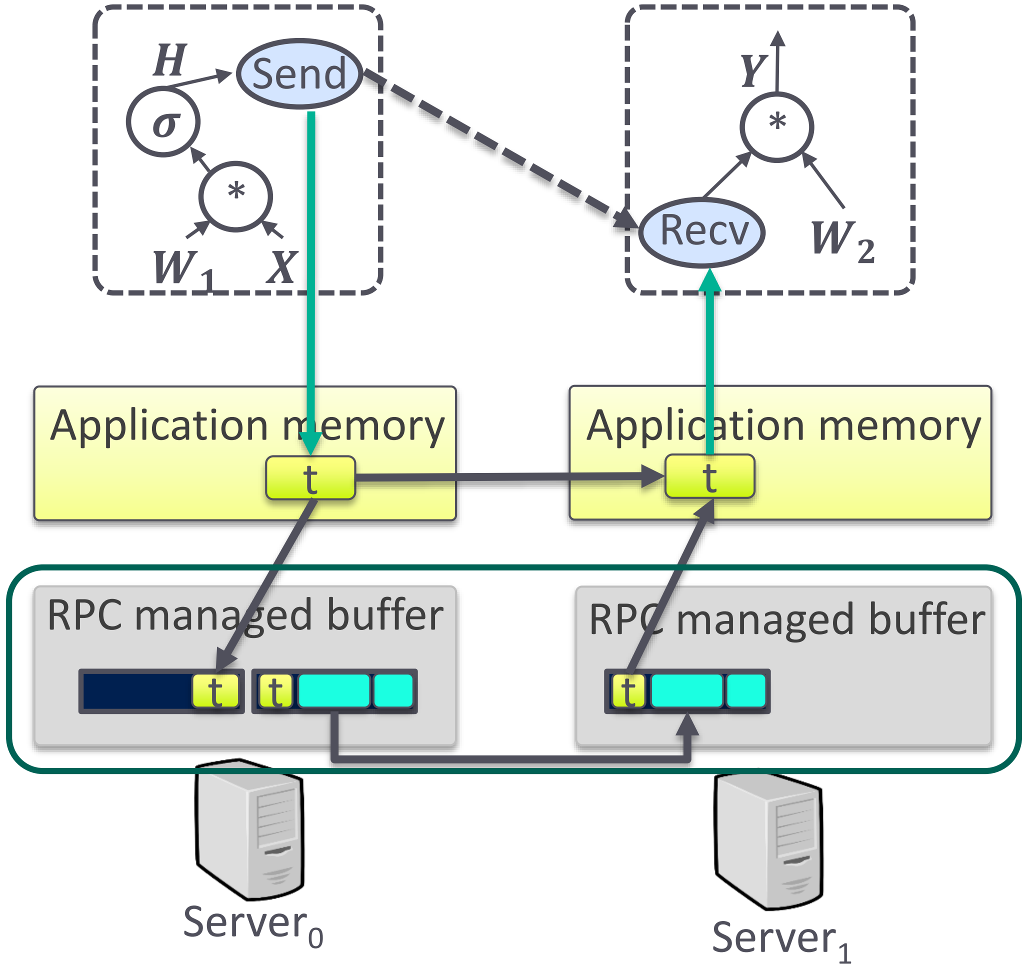
- Deep learning computation is modeled as dataflow graph
  - Achieve parallel manner through graph partitioning
    - Model parallelism vs. data parallelism
  - Tensor transmission across server becomes bottlenecks





General Message Passing Library (e.g., RPC)

- Unavoidable memory copy overhead in RPC
  - Generally designed for dynamic data structure
  - Lacks knowledge of actual data placement and size
  - Extra memory copy from data serialization
- Software/hardware co-design to completely remove memory copy overhead
  - Leverage runtime application information
  - RDMA network



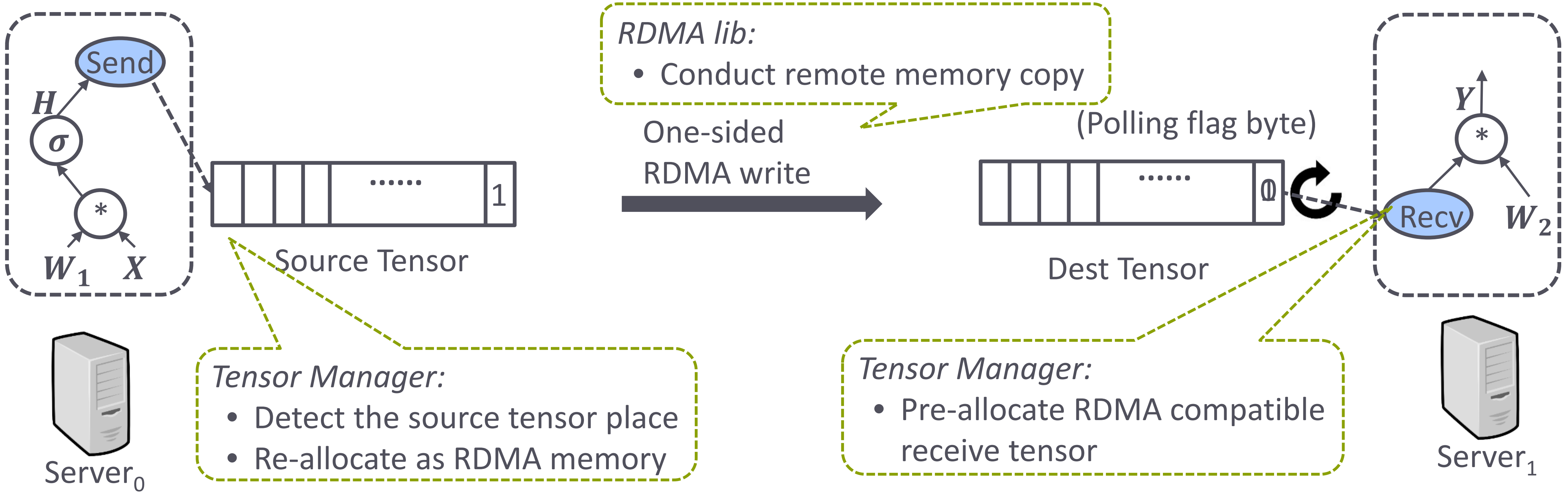
Combine Dataflow Graph Computation with RDMA

- Tensor abstraction in deep learning computation
  - Consists of a plain byte array with sufficiently large size (tens of KB to MB)
  - Do NOT require variant data serialization/deserialization
  - Do NOT require extra batching since access pattern is already sequential
- RDMA enables to manage local and distributed memory in a unified view
  - One-side RDMA R/W : efficient memory copy between host memory
  - GPU-Direct RDMA : efficient memory copy between host and device memory
- Global graph optimizer for distributed computation
  - Has the entire view and control of memory placement among devices and servers
  - Capable of making globally optimized strategy for tensor data placement in runtime

Optimized Communication Mechanism

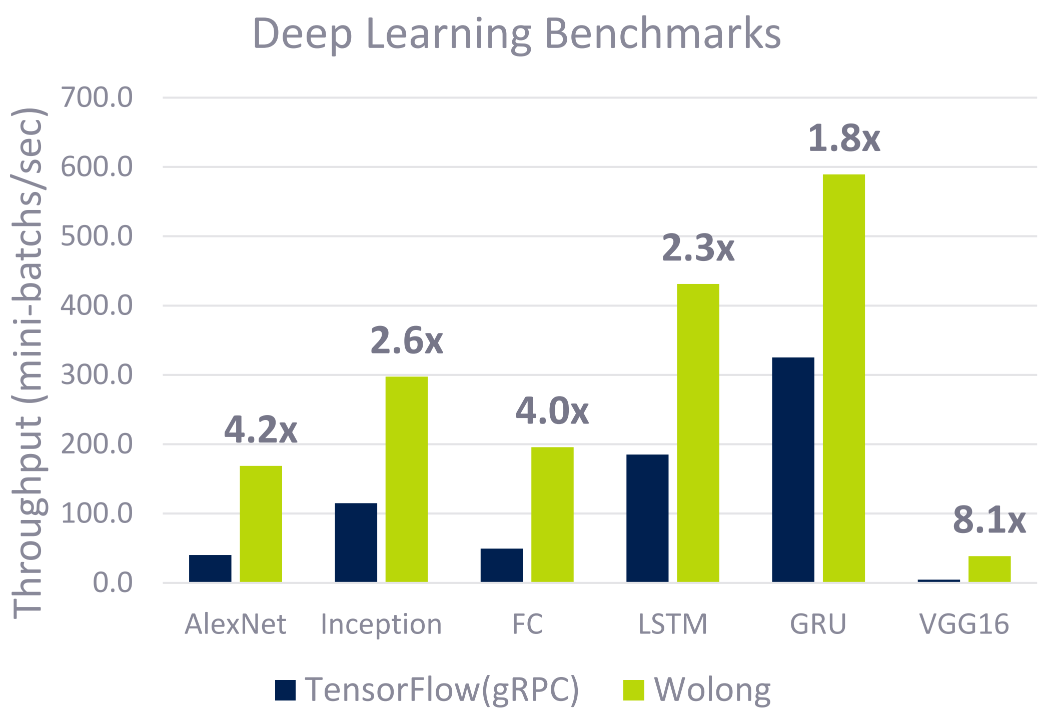
- Transfer statically placed tensor through one-side RDMA write
  - Phase I: graph analyzing
  - Phase II: graph execution

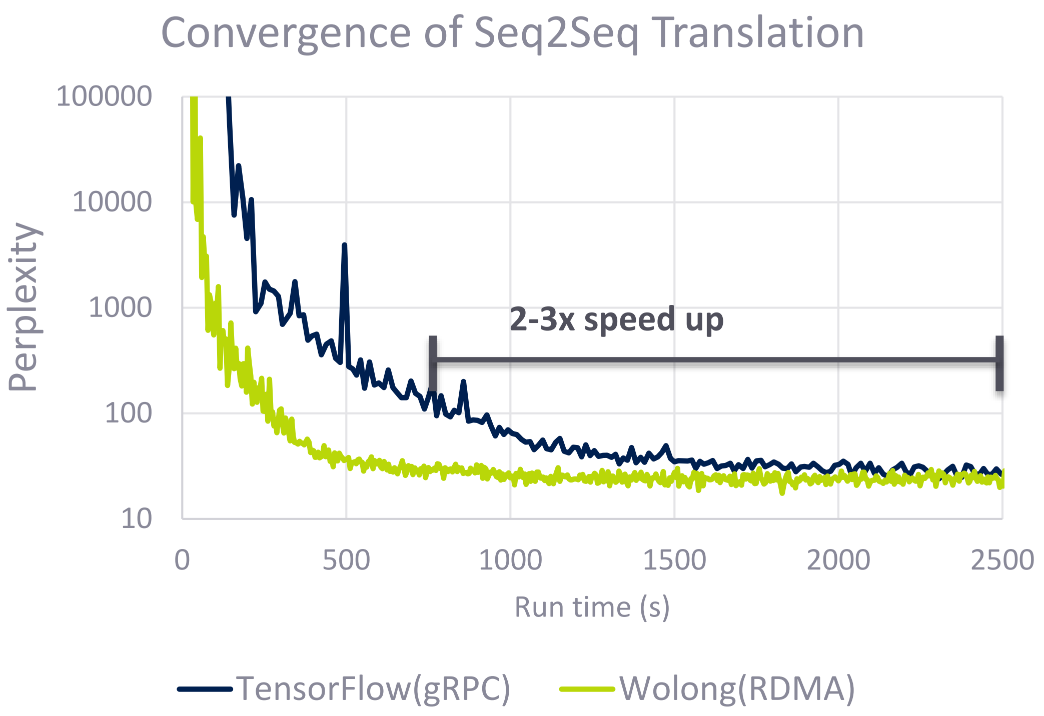
RDMA-based zero-copy communication



Global Optimizer: Performance Evaluation

- Improve training throughput, convergence speed and scalability





More details in our paper: *RPC Considered Harmful: Fast Distributed Deep Learning on RDMA*

* Experiments are conducted on 8 servers 8 Nvidia GTX 1080 GPUs; The translation model uses WTM'15 datasets;

Local Optimizer

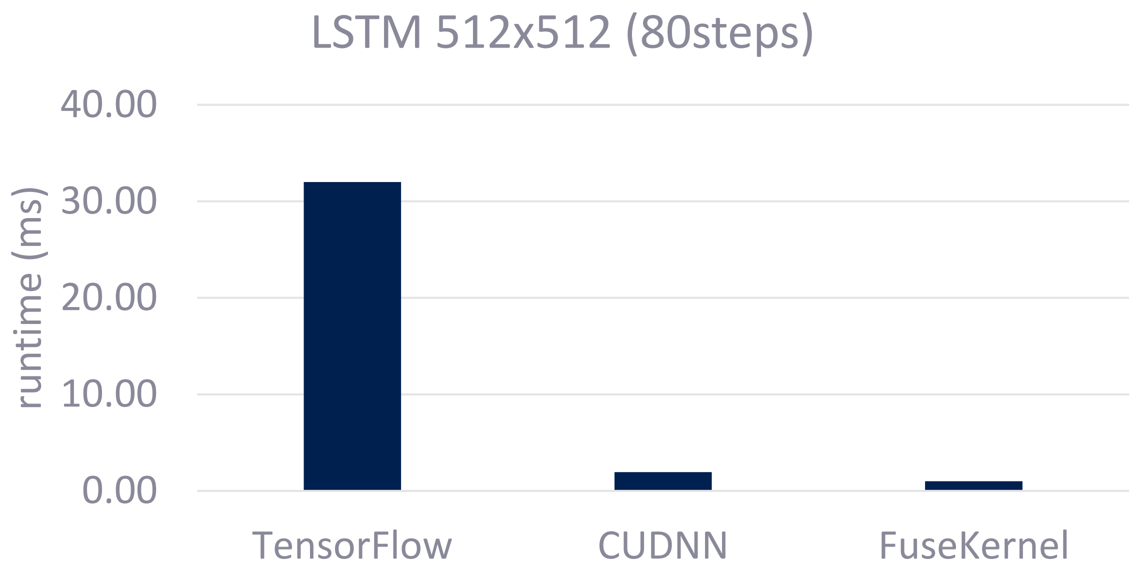
Kernel Fusion for Deep Learning on GPU

Motivation

- Deep learning frameworks model computation as graph of primitive **operators**
  - **Expressivity** to represent arbitrary neural network structure
  - **Flexibility** to run on multi-device and multi-server through graph partitioning

- Significant framework overhead to schedule thousands of operators
  - Kernel-launch overhead
  - Cross operator communication overhead
  - Too fine-grained to leverage vendor's library

- Example: 80-step LSTM model
  - Contains 1686 operators in TensorFlow



DL Frameworks vs. Vendor Provided Library

Flexibility

- Deep learning frameworks
  - E.g., TensorFlow, PyTorch, CNTK
  - Embrace flexibility and expressivity
  - Performance inefficiency

- **DL framework + Compiler**
  - Generate library-like code in runtime
  - Win both of the worlds

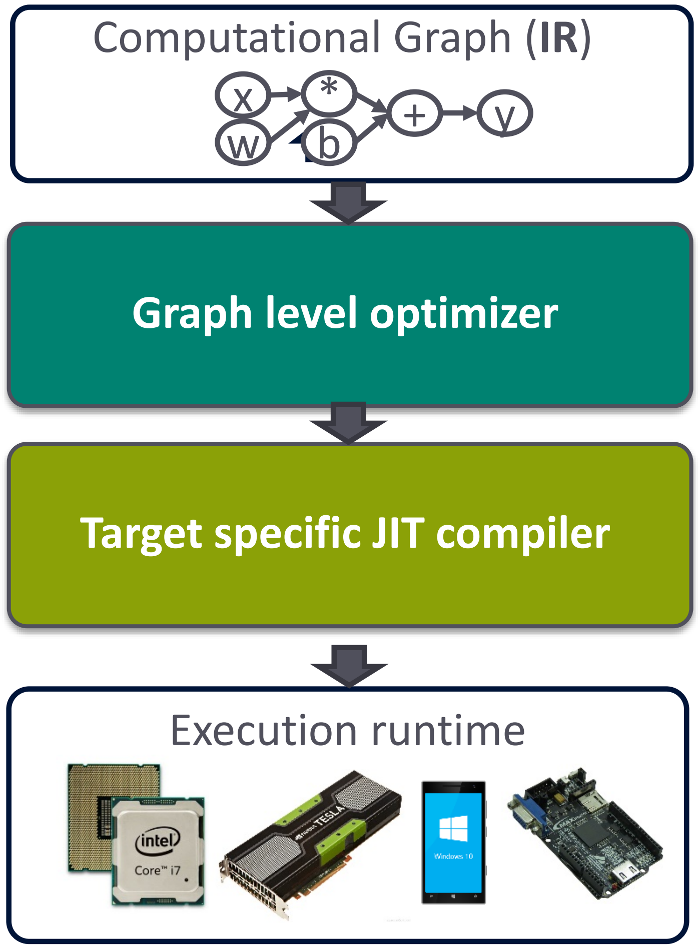
- Hardware specific library
  - E.g., cuDNN, cuBlas, MKL
  - Designed for extreme efficiency
  - Impossible to handle customized or new network structure

Efficiency

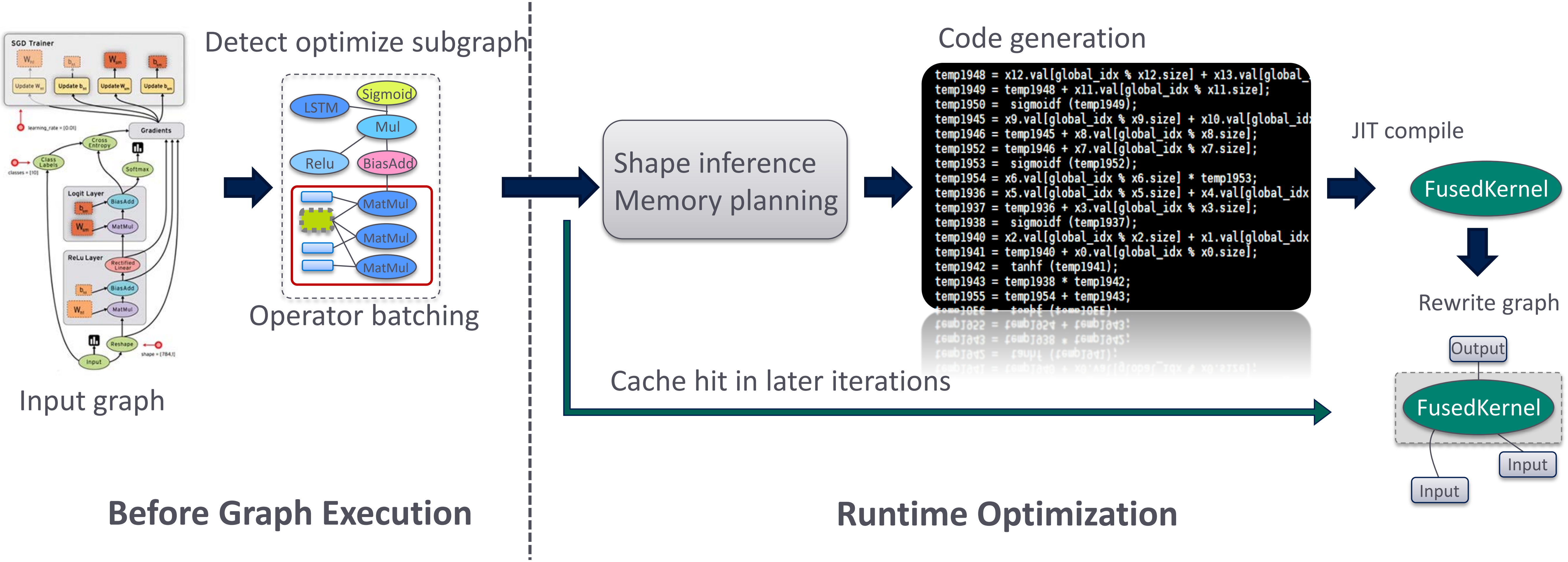
Wolong Compiler Design

- Computation graph level optimization
  - Graph rewriting based on computational equivalence
  - Common subexpression elimination, constant folding etc.
  - ***Operator batching***: automatically batch same type operators to better leverage batch efficiency

- Target and application specific runtime compilation
  - Static shape and type inference
  - Static memory planning
  - Aggressive kernel fusion

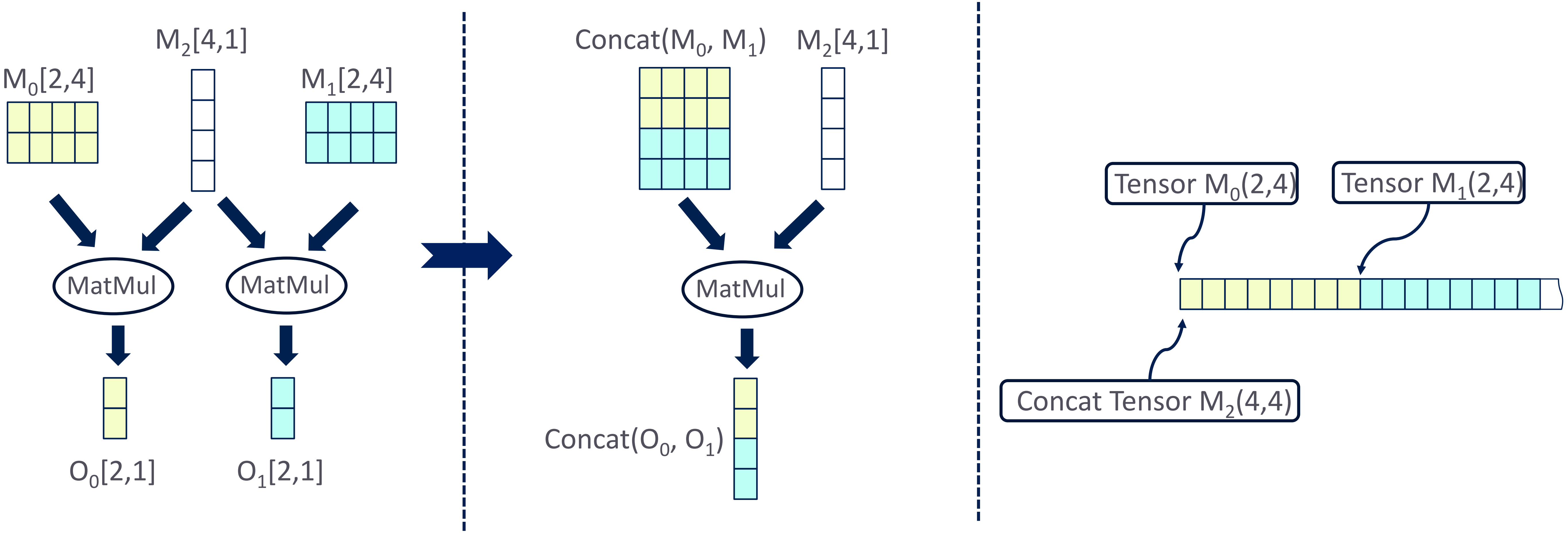


Wolong Compiler Execution Workflow



Graph Level Optimization: Operator Batching

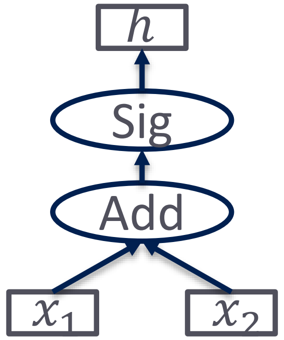
- Automatically conduct GEMM fusion and static memory placement optimization

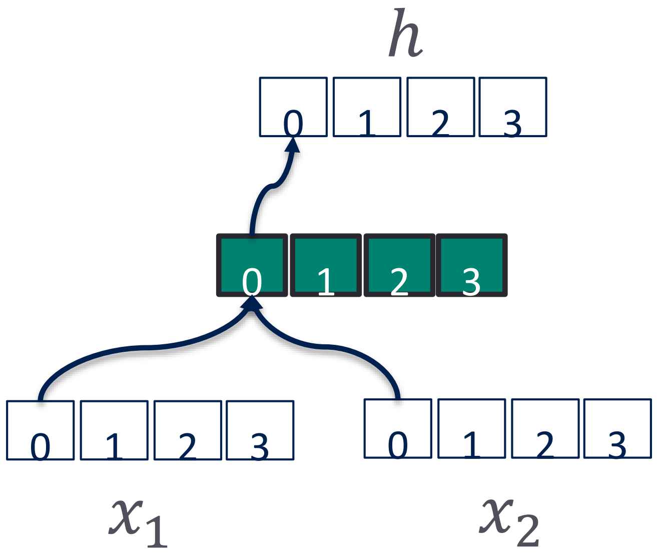


JIT Compilation: Kernel Fusion

- Leverage aggressive kernel fusion to completely remove scheduling overhead
- Element-wise (i.e., point-wise) operators
  - No cross-element dependency between operators
  - Better leverage cache, register locality

$$h = sigmoid(x_1 + x_2)$$



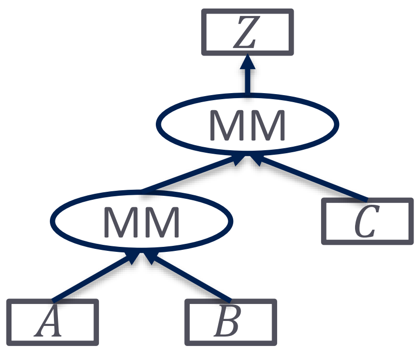



```
__global__
void kernel_0(float *x1, float *x2, float *h)
{
  int idx = blockIdx.x * blockDim.x +
            threadIdx.x;
  if (idx < 1024) {
    float temp0 = x1[idx] + x2[idx];
    float temp1 = sigmoidf(temp0);
    h[idx] = temp1;
  }
}
```

JIT Compilation: Kernel Fusion

- Fuse arbitrary (non element-wise) operators in to single kernel
  - Operator data dependency may introduce cross threads data dependency in kernel
  - Need global synchronization to guarantee correctness
  - Cross operator communication uses device memory

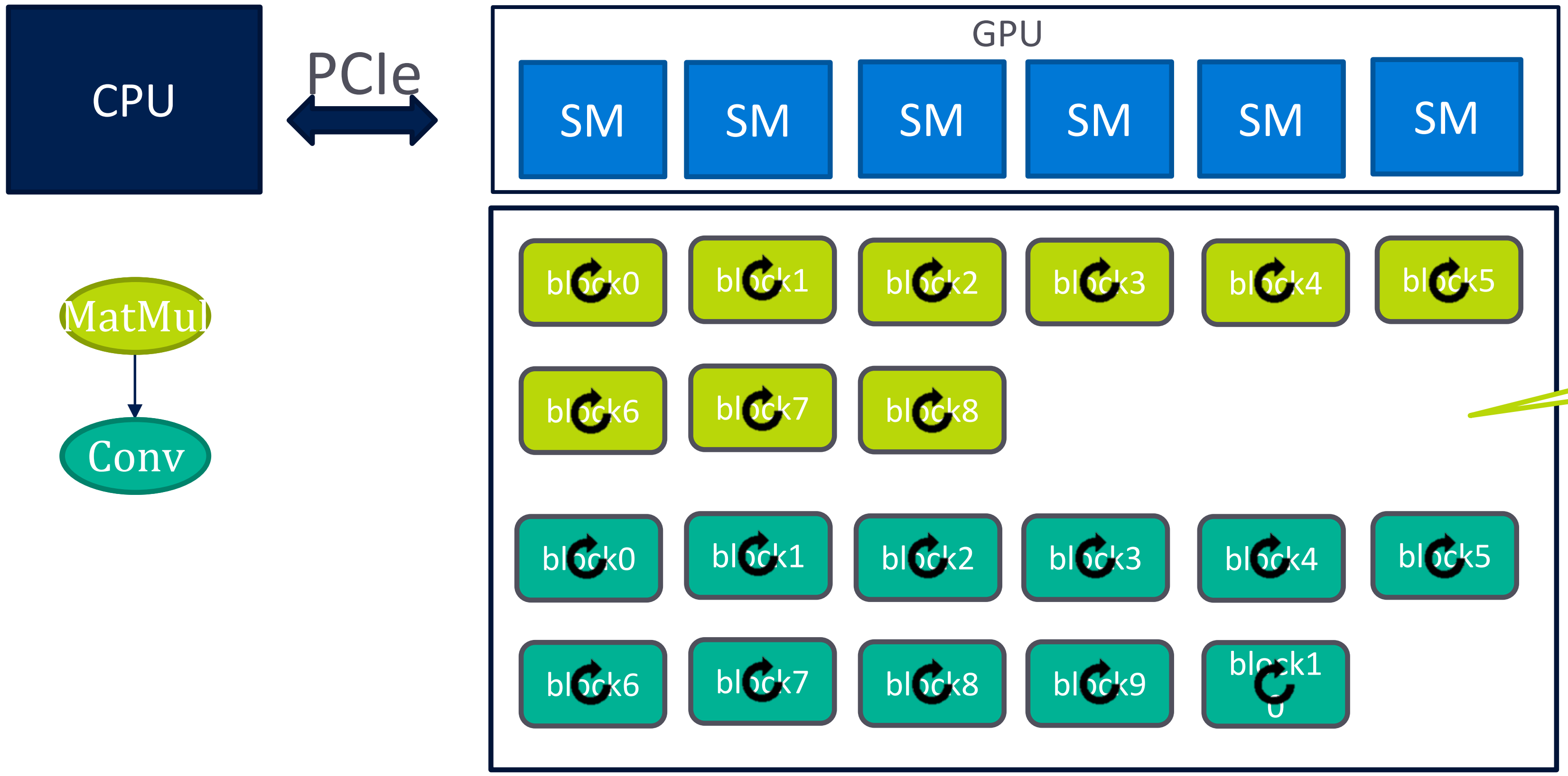
- E.g., fuse two matrix multiplications: $Z = A \times B \times C$




```
void kernel_0(float *A, float *B, float *C, float *Z) {
  if (idx < 1024) {
    buffer[idx] = MatMul_f(A, B);
    Global_Sync();
    Z[idx] = MatMul_f(buffer, C);
    h[idx] = temp1;
  }
}
```

Graph Computation in DL Frameworks

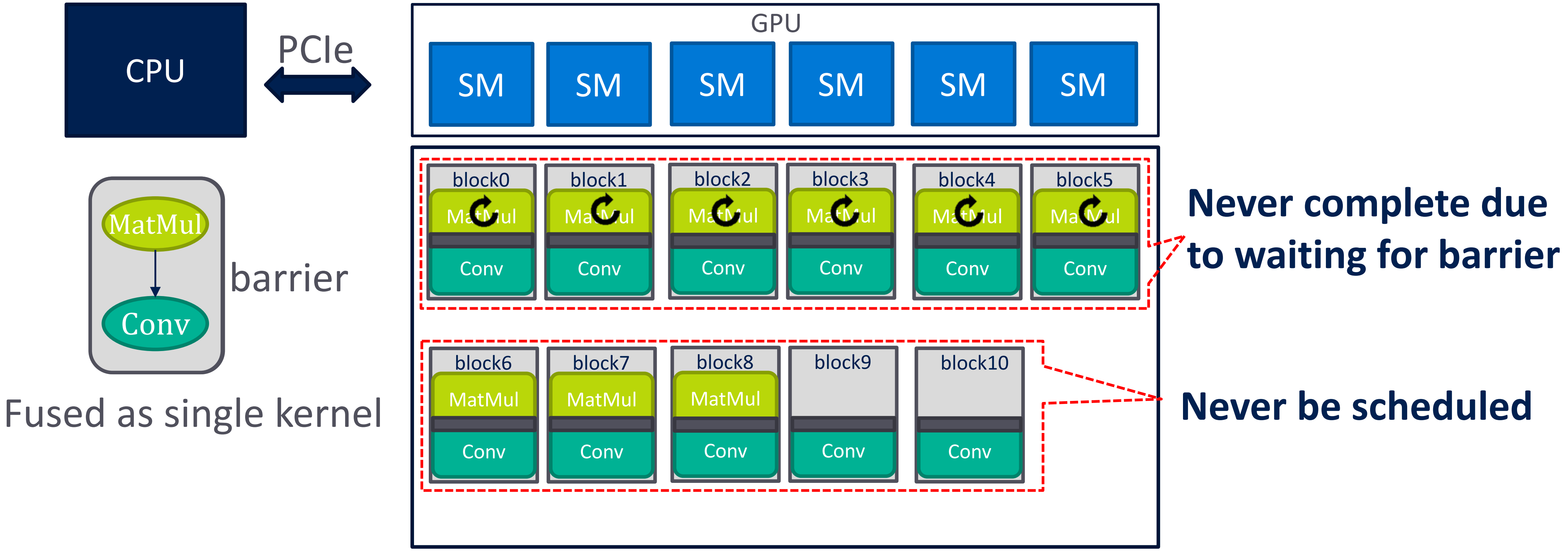
- Operators (kernels) are scheduled (launched) one by one



Overhead of kernel launching, DAG scheduling, memory copy, etc.

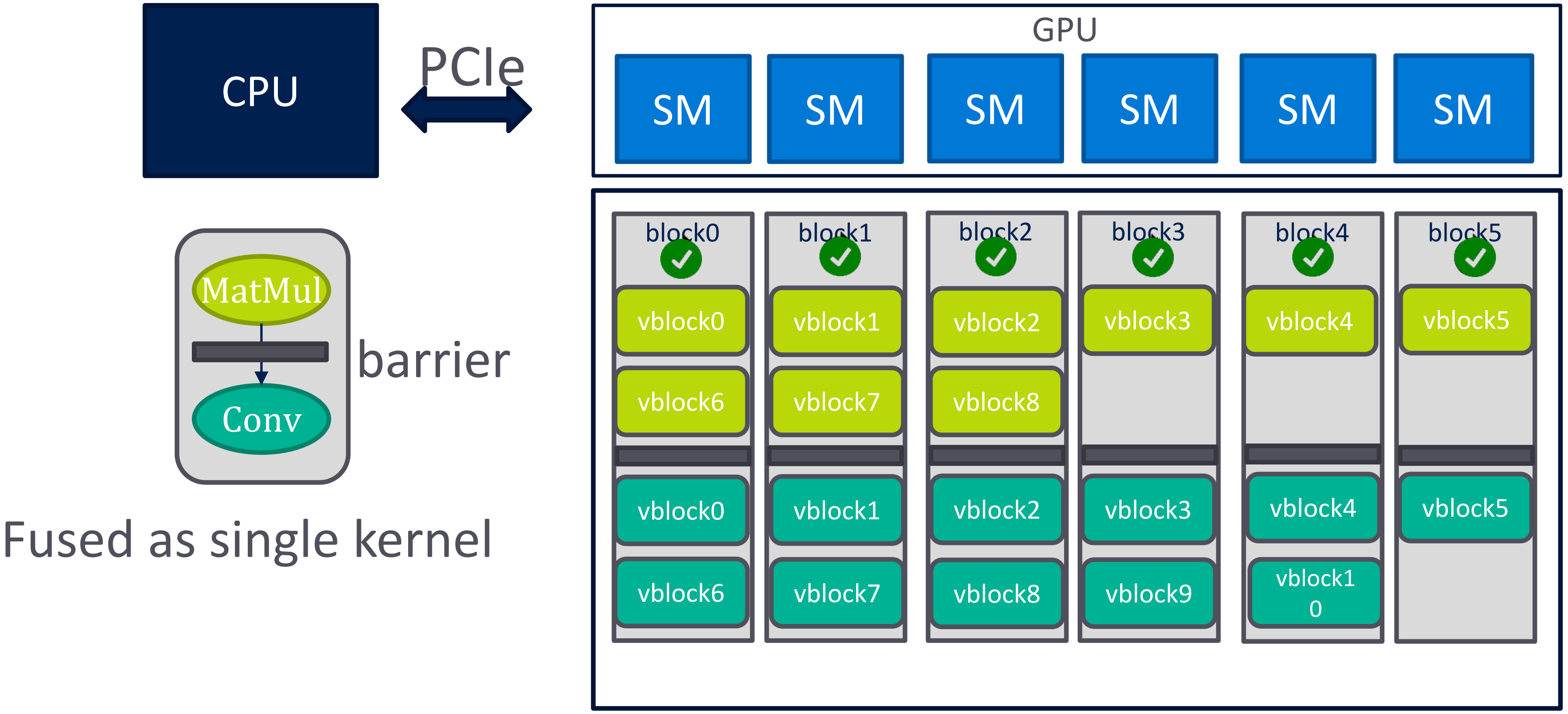
Arbitrary Kernel Fusion Is Limited by GPU Architechture

- Hard to conduct global synchronization across all threads



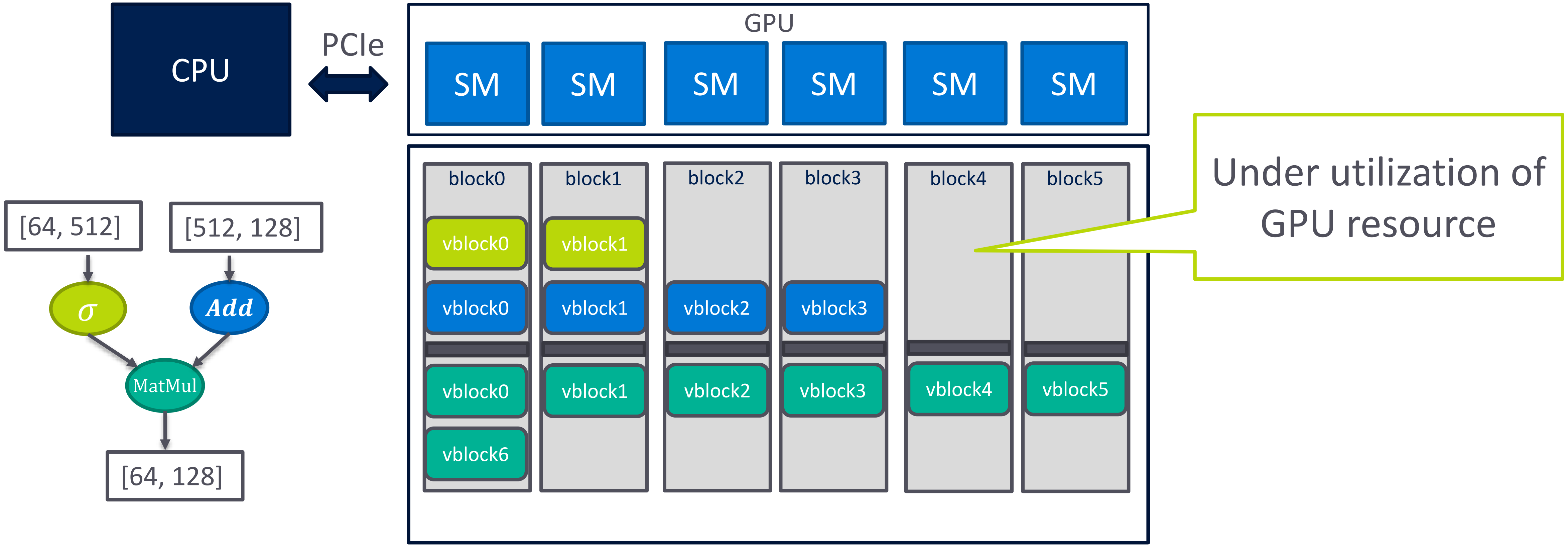
Our Solution: Persistent Threads and Virtual Blocks

- Assign virtual block task to persistent threads

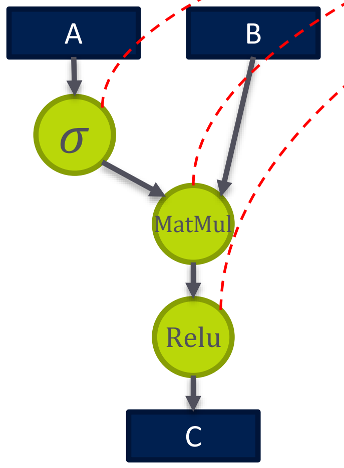


Kernel Packing

- Explore graph level parallelism in static code generation



Code Generation



Code Generator

```
//kernel code generated by Wolong compiler
__global__
void kernel_0(float *A, float *B, float *C) {
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < 1024) {
    float temp0 = sigmoidf(A[idx]);
    buffer0[idx] = temp0;
  }
  GlobalSync();
  for (int tix = bx; tix < ey; tix += offx) {
    for (int tiy = by; tiy < ey; tiy += offy) {
      MatMul(buffer0, B, buffer1, 1024, 1024, 128);
    }
  }
  if (idx < 1024) {
    float temp2 = reluf(temp1);
    C[idx] = temp2;
  }
}
```

Operator kernels: device functions

```
//device function for sigmoid operator
__device__ float sigmoidf(float in) {
  return 1.f / (1.f + expf(-in));
}
```

```
//device function for relu operator
__device__ float reluf(float in) {
  return fmaxf(0.f, in);
}
```

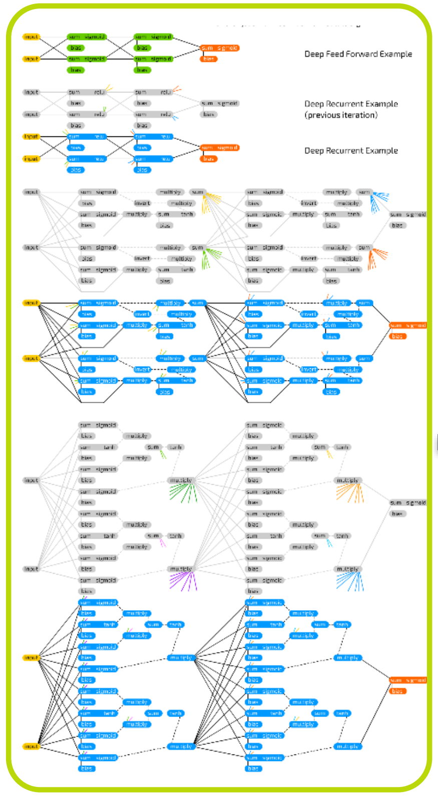
```
//device function for MatMul operator
__device__ float MatMul(float *a, float *b,
float *c, int m, int n, int k) {
  if (thread_ix < m && thread_iy < k) {
    float temp = 0.f;
    for (int i = 0; i < n; ++i) {
      temp += a[tix*n+i] * b[k*i+tiy];
    }
    c[k * tix + tiy] = temp;
  }
}
```

...

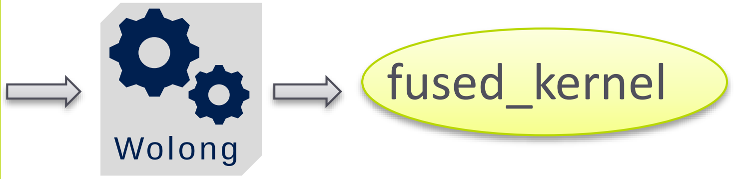
MatMul, Add, Mul, Sub, Div, Relu, Sigmoid, Tanh, Split, Max, Min, Convolution, etc.

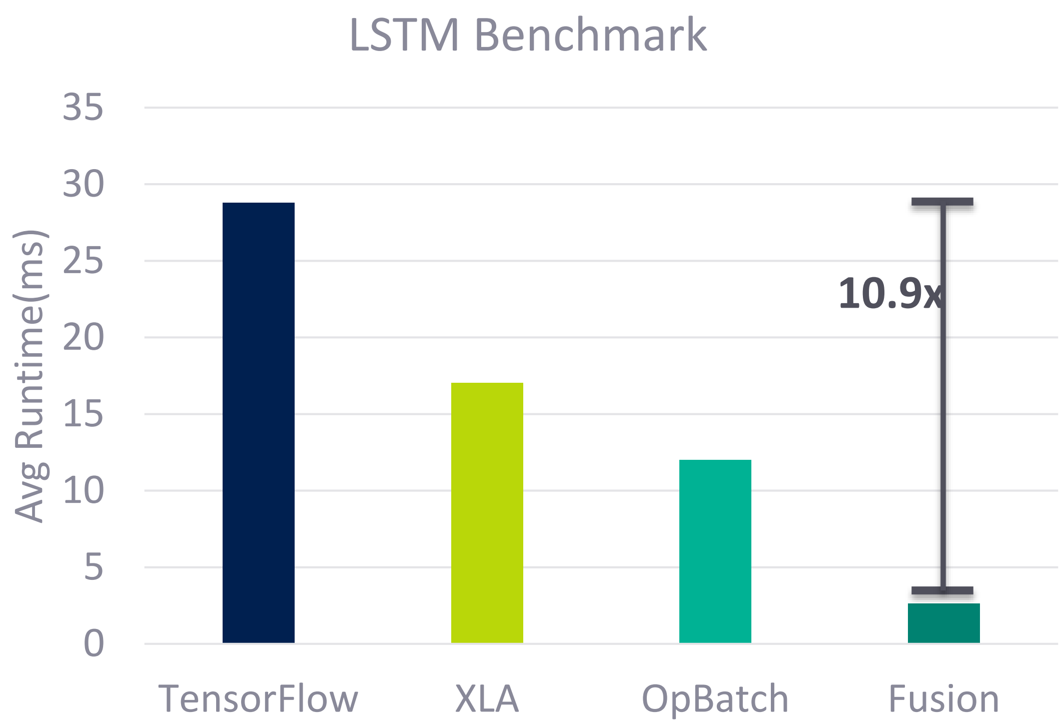
Performance of End-to-end Kernel Fusion

- RNN inference benchmark (LSTM-128uints-80steps)



Fused 1686 operators into 1 kernel





- Experiments are conducted on Nvidia GTX 1080 Ti GPUs

Conclusion

- A compiler infrastructure is critical for both cloud and edge AI
  - Optimize for fast distributed training in cloud
  - Optimize for efficient inference on accelerator devices
- System innovations to bridge applications and diverse hardware
  - Common intermediate representation (IR)
  - Co-design software and hardware for extreme efficiency
- Wolong prototype has demonstrated the initial improvements
  - Up to 8x speedup on training workloads
  - Up to 10x speedup on inference benchmark

Microsoft

Thank You!

Systems | Fueling future disruptions

Microsoft

Distributed Graph Optimizer of Wolong

- Transfer dynamically allocated tensor through RDMA write/read
  - Phase I: graph analyzing
  - Phase II: graph execution

Supports GPUDirect RDMA as well