Agenda

- 3:20 to 3:40: Coffee Break
- 3:40 to 4:00: Brief Introduction to CUDA Programming
- 4:00 to 5:00: Build Tensor Programs with Hidet in Python
  - Part 1: Introduction to Deep Learning Compiler Hidet
  - Part 2: Interactive Demos on how to use Hidet (with Jupyter Notebooks)

Tutorial Website

UNIVERSITY OF TORONTO

CentML

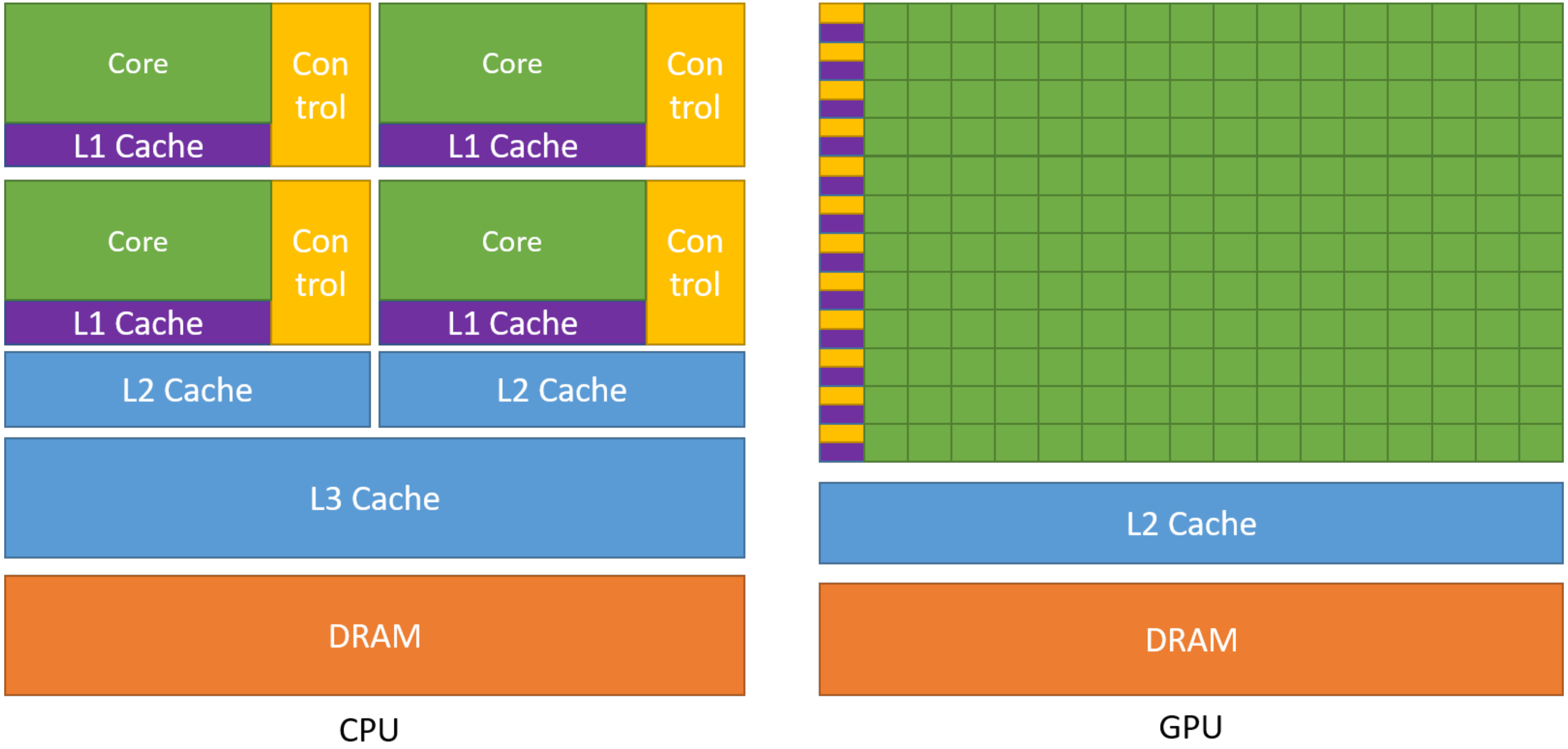
A Brief Introduction to CUDA Programming

Yaoyao Ding

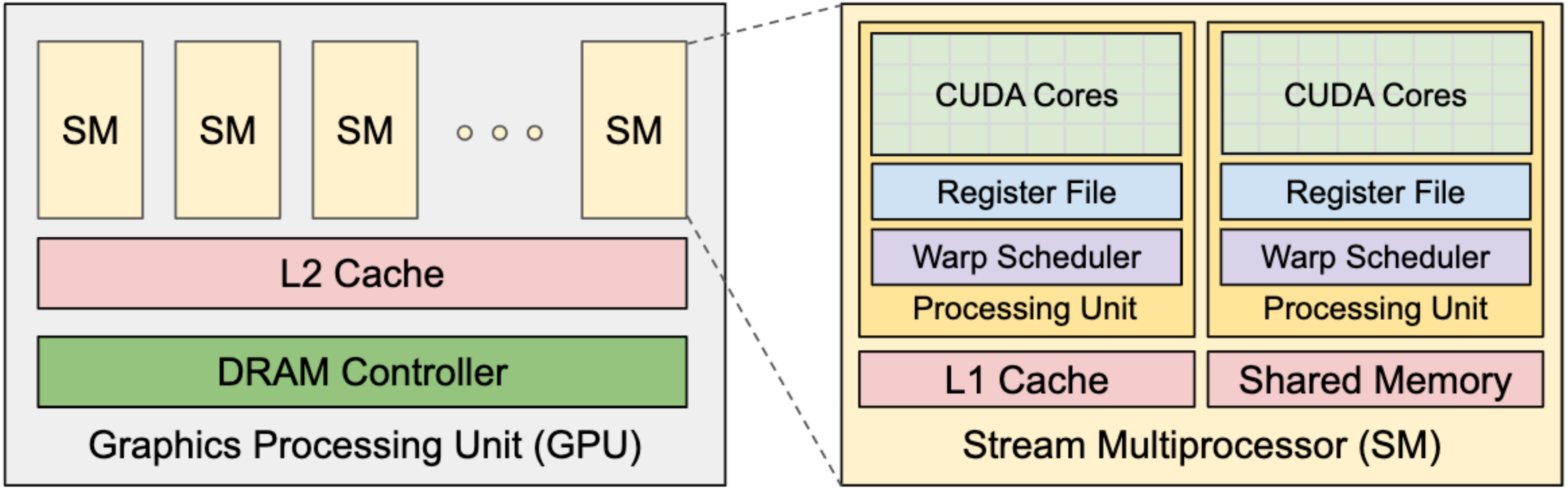
yaoyao@cs.toronto.edu

Contents are adopted from NVIDIA's CUDA C++ Programming Guide

CPU vs. GPU


Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
L2 Cache
L2 Cache
L3 Cache
DRAM
CPU
L2 Cache
DRAM
GPU

Inside a NVIDIA GPU



CUDA Kernel and Its Invocation

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Something New:

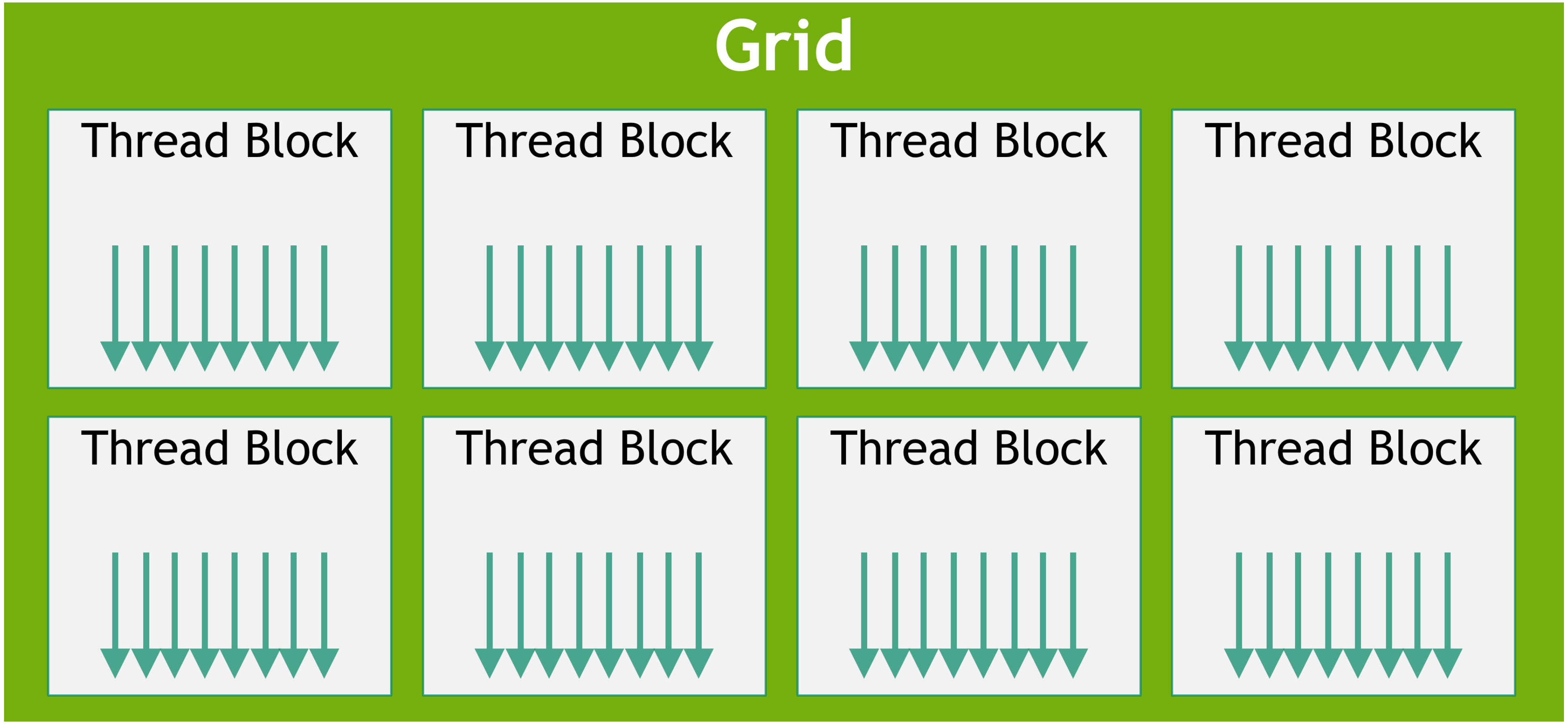
- `__global__`
- `threadIdx.x`
- `<<<1, N>>>`

Thread Hierarchy: Up to 3 dimensions

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Thread Hierarchy: Thread, Thread Block, Grid



Thread Hierarchy

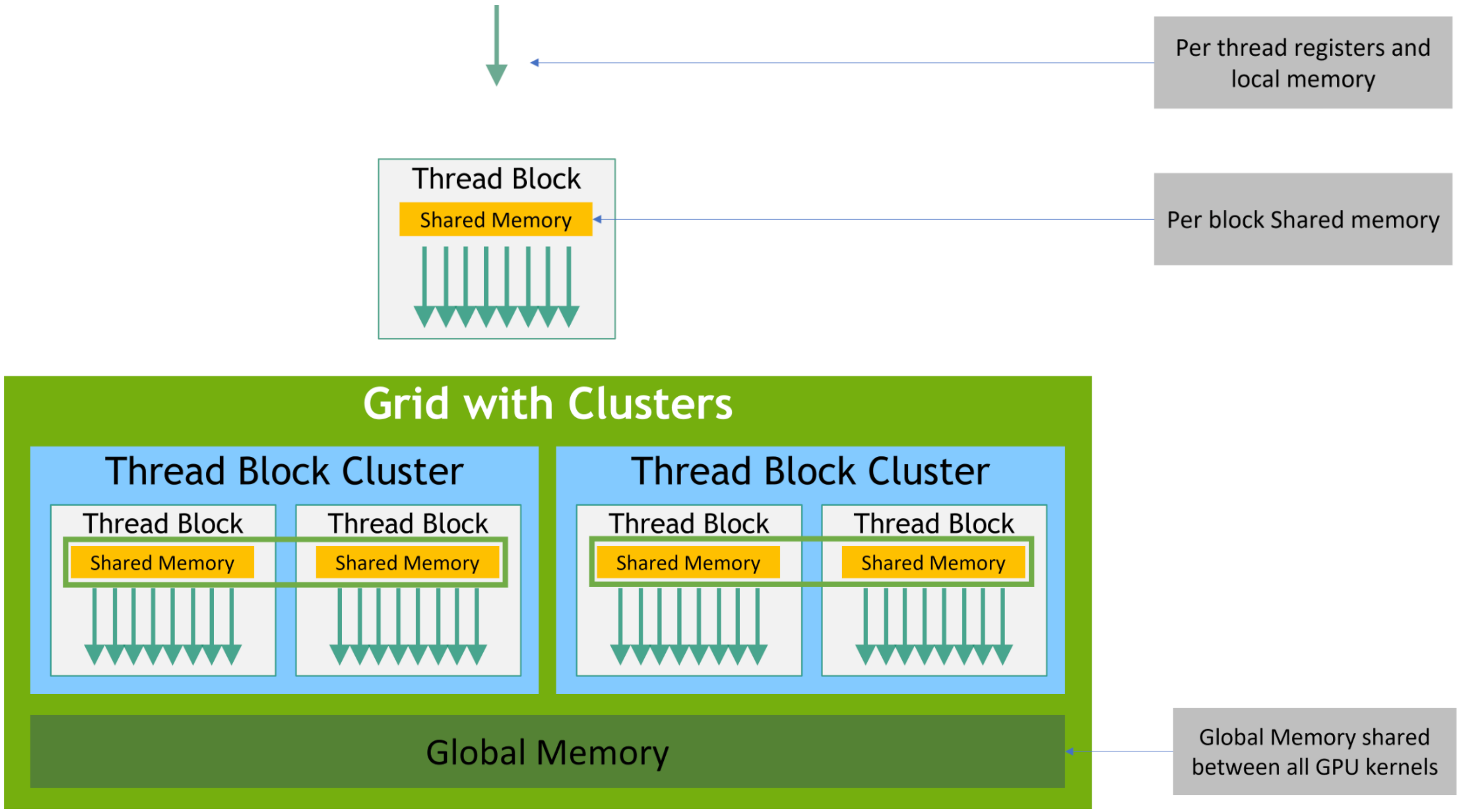
```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

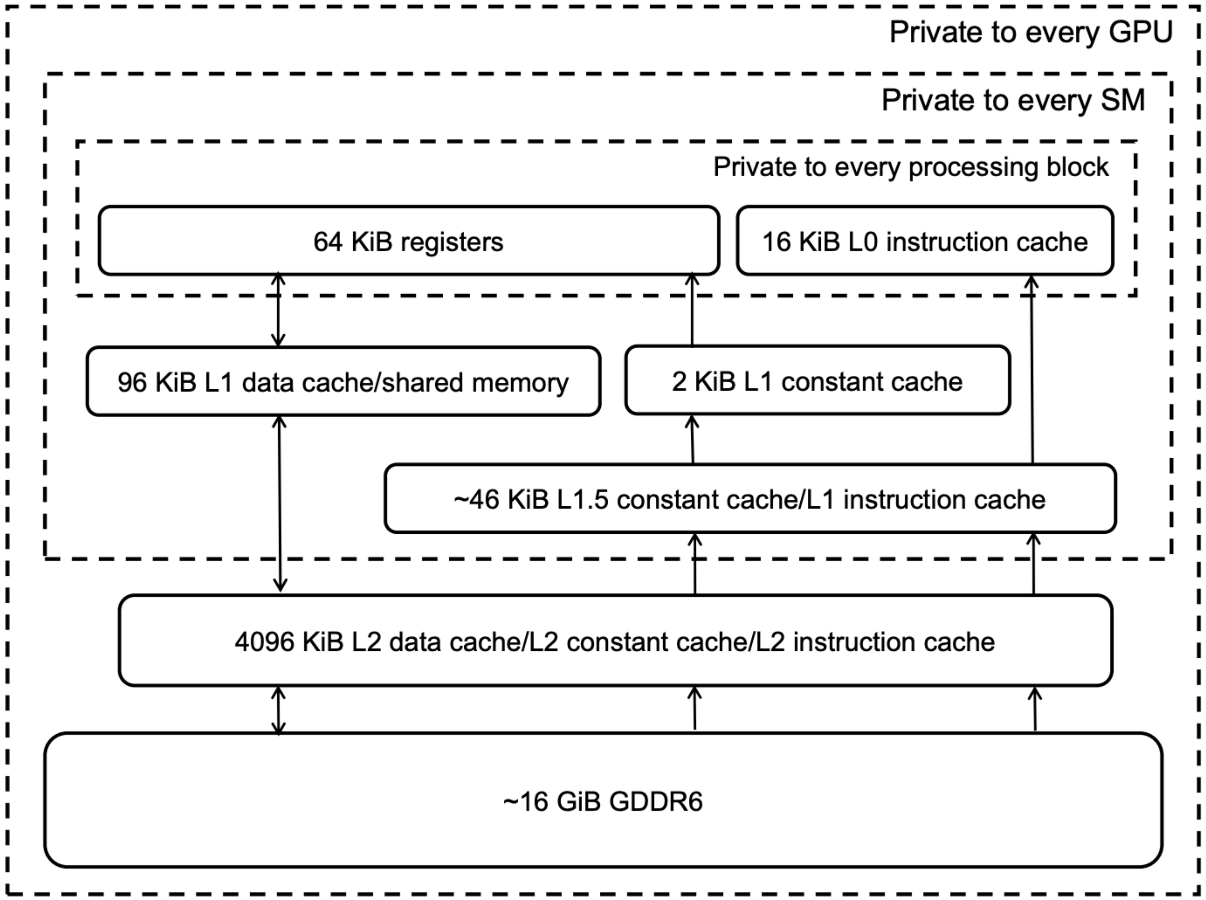
Threads in a block can cooperate by:

- Shared memory, and
- Synchronizing their execution via `__syncthreads()`

Memory Hierarchy



Memory Hierarchy



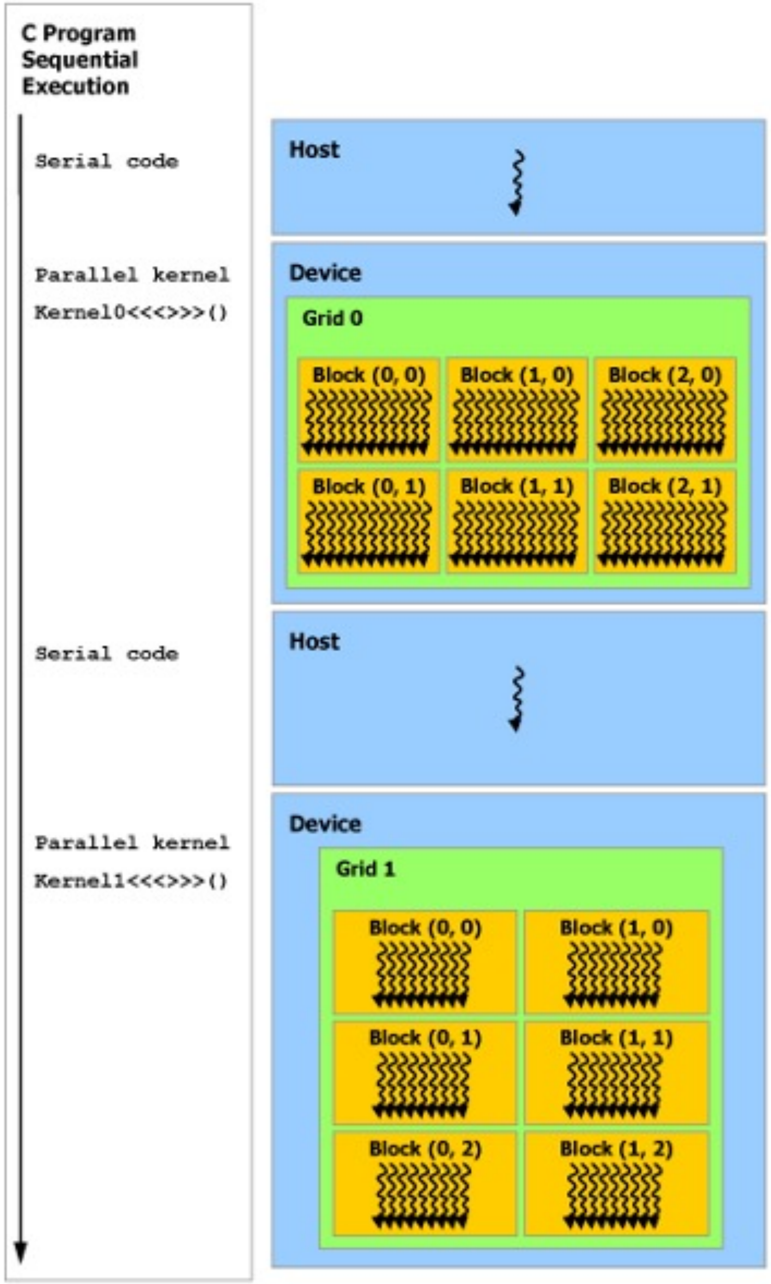
Memory hierarchy of the Turing T4 GPU (TU104)

NVIDIA T4 GPU Latency:

- Shared memory: 19 cycles
- Global memory:
  - L1 hit: 32 cycles
  - L2 hit: ~188 cycles
  - L2 miss & TLB hit: 296 cycles
  - L2 miss & TLB miss: 616 cycles

Reference: Dissecting the NVidia Turing T4 GPU via Microbenchmarking

Heterogeneous Programming



Device Memory

Host Memory

Device Memory

Example: Vector Addition

```
__global__ void add_kernel(int *a, int *b, int *c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        c[i] = a[i] + b[i];
    }
}
```

Example: Matrix Multiplication (Matmul)

```
__global__ void matmul_kernel(float *a, float *b, float *c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < n && j < n) {
        float acc = 0.0;
        for (int k = 0; k < n; k++) {
            acc += a[i * n + k] * b[k * n + j];
        }
        c[i * n + j] = acc;
    }
}
```

GitHub: www.github.com/hidet-org/hidet
Installation: `pip install hidet`
Usage: `torch.compile(model, backend='hidet')`

Hidet:
An Open-Source Deep Learning Compiler

Yaoyao Ding

yaoyao@cs.toronto.edu

Tensor Programs are Everywhere

Paradise cosmic beach

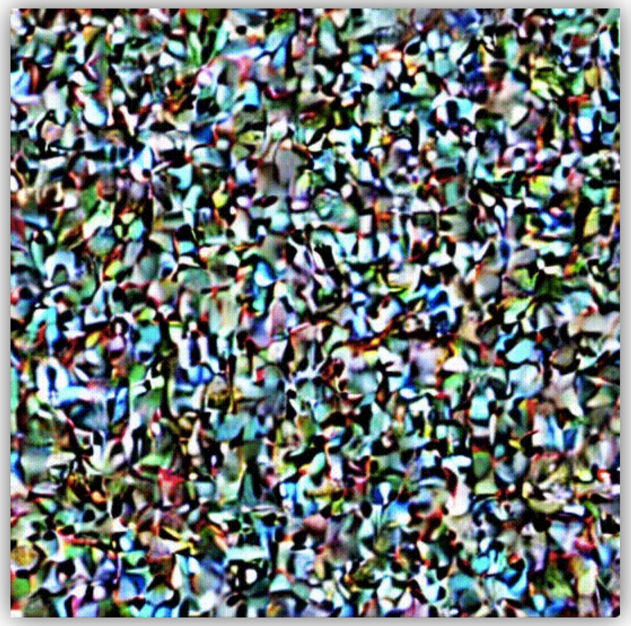


Image Generation [1]

Compose a funny rap for yourself in 5 lines

Yo, it's ChatGPT, the AI sensation
Here to answer your every question
I may be digital, but don't you worry
My rhymes are fire, they'll make you hurry
So come on and ask, let's make history!

Chat Bot

$694K per day [2]

```
void conv2d(
    float32 image[N, Ci, H, W],
    float32 filter[Co, Ci, Kx, Ky],
    float32 output[N, Co, H, W]
) {...}
```

Tensor Programs

Run fast

Reduce cost

[1] The Illustrated Stable Diffusion, by Jay Alammar
[2] https://www.semianalysis.com/p/the-inference-cost-of-search-disruption

Tensor Program Generation & Optimization

- Vendor Library (Manually optimization)
- State-of-the-art Tensor Compiler: Apache TVM

Limited Support for
Non-Loop-Oriented Optimizations

Sub-Optimal Performance

Long Optimization Time (e.g., hours)

- **Task-Mapping Programming Paradigm (ours)**

Good Support for
Non-Loop-Oriented Optimizations

Up to 1.4x Better Performance

11x Less Optimization Time

Deep Learning Compiler: TVM

Computation

```
C = compute([M, N], lambda i, j: reduce([K], A[i, k] * B[k, j]))
```

❶ Generate Default Tensor Program

Default Program

```
for i in range(M):
  for j in range(N):
    for k in range(K):
      C[i, j] += A[i, k] * B[k, j]
```

❷ Apply **Declarative** Scheduling Primitives

Schedule Primitives	Original Program	Scheduled Program

fuse(i, j)	for i in range(128):
	  for j in range(4):
	    body(i, j)
		for ij in range(512):
		  body(ij / 4, ij % 4)

split(i, 128)	for i in range(512):
	  body(i)
		for oi in range(4):
		  for ii in range(128):
		    body(oi * 128 + ii)

reorder(i, j)	for i in range(128):
	  for j in range(4):
	    body(i, j)
		for j in range(4):
		  for i in range(128):
		    body(i, j)

bind(i, threadIdx.x)	for i in range(128): body(i)	body(threadIdx.x)

Apply Declarative Scheduling Primitives

```
for i in range(M):
  for j in range(N):
    for k in range(K):
      C[i, j] += A[i, k] * B[k, j]
```

Default Program

1

```
oi, ii = split(i, 64)
oj, ij = split(j, 64)
```

```
for io in range(M/64):
  for ii in range(64):
    for oj in range(N/64):
      for ij in range(64):
        for k in range(K):
          i, j = io * 64 + ii, jo * 64 + ij
          C[i, j] += A[i, k] * B[k, j]
```

2

```
reorder(oi, oj, ii, ij)
```

```
for io in range(M/64):
  for oj in range(N/64):
    for ii in range(64):
      for ij in range(64):
        for k in range(K):
          i, j = io * 64 + ii, jo * 64 + ij
          C[i, j] += A[i, k] * B[k, j]
```

3

```
bind(oi, blockIdx.x)
bind(oj, blockIdx.y)
```

```
for ii in range(64):
  for ij in range(64):
    for k in range(K):
      i = blockIdx.x * 64 + ii
      j = blockIdx.y * 64 + ij
      C[i, j] += A[i, k] * B[k, j]
```

Scheduled Program

github.com/hidet-org/hidet

For simplicity, we assume M and N are a multiple of 64.

Deep Learning Compiler: TVM

Computation

```
C = compute([M, N], lambda i, j: reduce([K], A[i, k] * B[k, j]))
```

1 Generate Default Tensor Program

Default Program

```
for i in range(M): for j in range(N): for k in range(K):
  C[i, j] += A[i, k] * B[k, j]
```

2 Apply **Declarative** Scheduling Primitives (called Scheduling)

```
1. oi, ii = split(i, 64)      4. bind(oi, blockIdx.x)
2. oj, ij = split(j, 64)      5. bind(oj, blockIdx.y)
3. reorder(oi, oj, ii, ij)    6. ...
```

Two Scheduling Methods:

- Template-based: AutoTVM [1]
- Rule-based: Ansor [2]

Kernel Tuning:
Search best schedule hyper-parameter

Scheduled Program

```
for ii in range(64):
  for ij in range(64):
    for k in range(K):
      C[i, j] += A[ii + blockIdx.x * 64, k] * B[k, ij + blockIdx.y * 64]
```

Declarative Loop-Oriented Scheduling

github.com/hidet-org/hidet

[1] Chen, Tianqi, et al. "Learning to optimize tensor programs." NeurIPS 2018.
[2] Zheng, Lianmin, et al. "Ansor: Generating high-performance tensor programs for deep learning." *OSDI* 2020.

Limitation of Loop-Oriented Scheduling

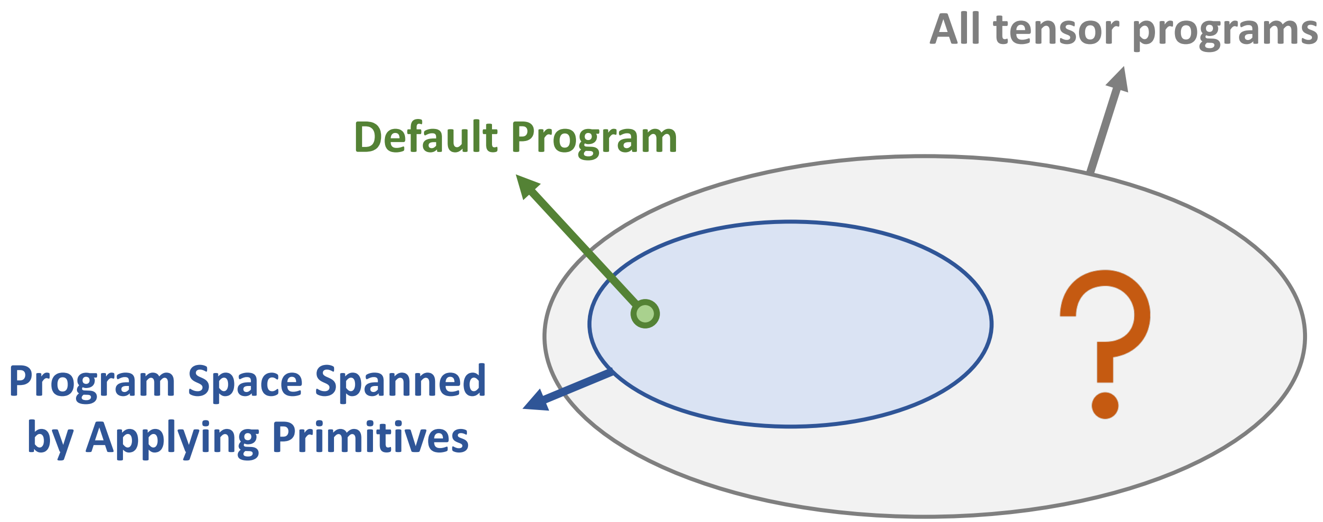
Computation

❶ Generate Default Tensor Program

Default Program

❷ Apply **Declarative** Scheduling Primitives

Scheduled Program



Limited Expressiveness of Loop-Oriented Scheduling ➡

Sub-Optimal Performance

(e.g., can not express double buffering opt.)

Long Optimization Time

(4-6 hours on average)

An Example of Non-Loop-Oriented Optimization

```
def matmul(A: fp32[M, K], B: fp32[K, N],
           C: fp32[M, N]):
  SmemA, SmemB = shared fp32[64, 8], fp32[8, 64]
  RegsC = local fp32[...]

  for k0 in range(128):
    # Load A and B fragments
    # from global memory to shared memory
    SmemA, SmemB = cooperative_load(A, B, k0)
    sync_threads()
    # RegsC = SmemA * SmemB + RegsC
    RegsC = block_mma(SmemA, SmemB, RegsC)
    sync_threads()
  ...
```

Matrix Multiplication without Double Buffering

```
RegsA, RegsB = register fp32[...], fp32[...]
SmemA, SmemB = shared fp32[2, 64, 8], fp32[2, 8, 64]
SmemA[0], SmemB[0] = cooperative_load(A, B, 0)
sync_threads()

for k0 in range(127):
  p, q = k0 % 2, (k0 + 1) % 2
  RegsA, RegsB = cooperative_load(A, B, k0 + 1)
  RegsC = block_mma(SmemA[p], SmemB[p], RegsC)
  SmemA[q], SmemB[q] = RegsA, RegsB
  sync_threads()
...
```

Two buffers for A/B

Preloading **Next** Tile of A/B into Registers

Computation of **Current** Tile

Store Next Tile of A/B into Shared Memory

Matrix Multiplication with Double Buffering

Key Idea: Task Mapping Programming Paradigm

```
Smem = compute([64, 8], lambda i, j => Gmem[i, j])
```

1 Default Program

```
for i in range(64):
  for j in range(8):
    Smem[i, j] = Gmem[i, j]
```

2 Schedule

```
io, ii = split(i, 16)
iik = fuse(ii, k)
bind(iik, threadIdx.x)
```

3 Lower Primitives

Declarative Loop-Oriented Scheduling

1 Write Program with Task Mapping

```
mapping = repeat(4, 1).spatial(16, 8)
for i, j in mapping(threadIdx.x):
  Smem[i, j] = Gmem[i, j]
```

2 Lower Task Mappings

Task Mapping replaces **Schedule Primitives**

```
for io in range(4):
  t = threadIdx.x
  i, j = io * 16 + t / 8, t % 8
  Smem[i, j] = Gmem[i, j]
```

Task-Mapping Programming Paradigm

github.com/hidet-org/hidet

Definition, Usage and Pros of Task Mapping

Iterate the Assigned Tasks

Define Task Mapping

```
...
mapping = repeat(4, 1).spatial(16, 8)

for  i, k  in mapping( threadIdx.x ):
  SmemA[i, k] = A[i, k]
```

Task Index

Worker Index

Implementation of a task

```
mapping = repeat(4, 1).spatial(2, 64)

for  i, j  in mapping( threadIdx.x ):

  SmemB[i, j] = B[i, j]

...
```

- **High Flexibility:**
  Allow developer to manipulate tensor programs in fine granularity
  => More Optimizations and Better Performance

- **Efficient Partial Tile:**
  Add the predicate inside the loop body
  => Reduce Tuning Space & Optimization Time

- **Post Scheduling Fusion:**
  Automatically fuse surrounding operators
  => Less Memory Transfer and Better Performance

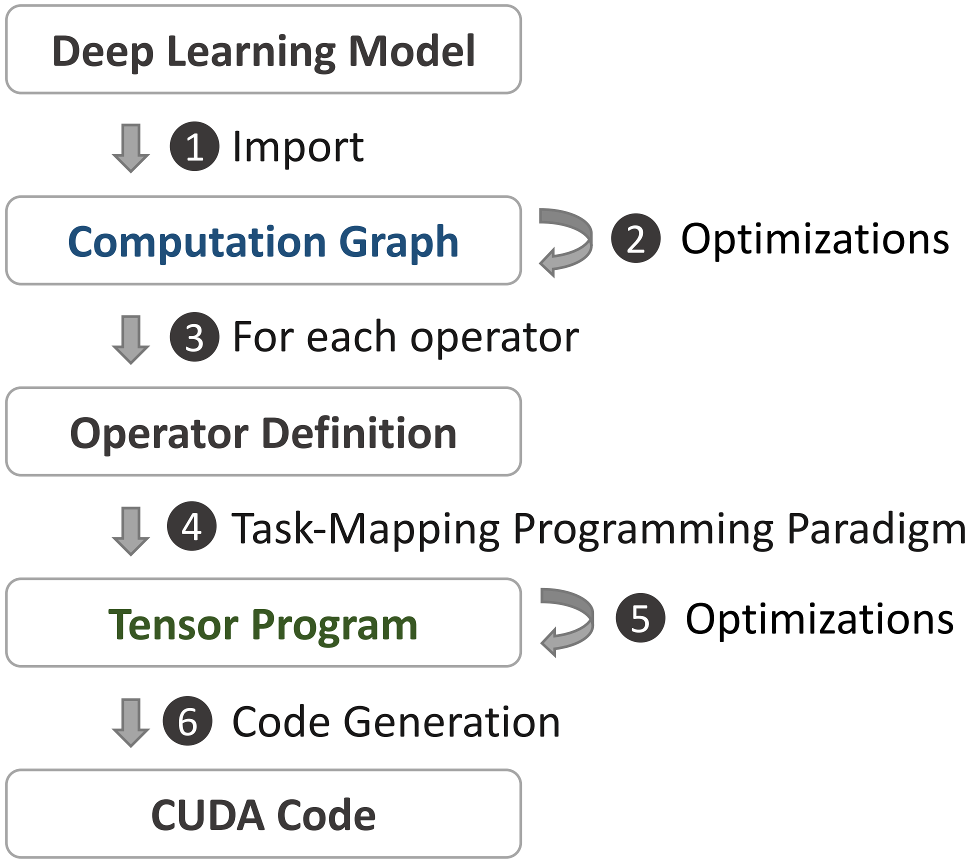
Implementation

GitHub: www.github.com/hidet-org/hidet
Installation: `pip install hidet`
Usage: `torch.compile(model, backend='hidet')`

- **Hidet**
  A Deep Learning Compiler from Scratch

- **Intermediate Representations (IRs)**
  - High Level: Computation Graph IR
  - Low Level: Tensor Program IR

- **Two Scheduling Machanisms at ④**
  - Template-based Scheduling
  - Rule-based Scheduling



Hidet Compilation Flow

Practice Session

Tutorial Website