2025.03.04

To Be Continued

- CUTLASS: C++ CUDA Templates for Linear Algebra Subroutines
  - layout algebra: composing layouts (as projections?)
    - nested layouts aka modes: subset of axes, have specific semantics e.g. for threads, tensor cores
    - compile-time checks for static parts of layouts
    - also with swizzle functors
    - tiling as dividing shape by shape, or layout by shape
    - tiling compute resources
  - facilitate working with tensor cores
  - layout = (shape) : (stride), concatenable ("multiply")
    - tile: shape changes, but not stride
  - Python interface
    - brief code thanks to good defaults
    - brief but very informative error messages compared to C++ template compile errors
  - CuTe: C++ CUDA Template Library for Tensors
    - front-end to CUTLASS 3.0
    - conceptual GEMM Hierarchy
      - atom layer
        - centered around hardware-accelerated ops e.g. tensor core, vector op
      - tiled MMA/Copy
        - centered around optimal use of a single GPU
      - collective layer
        - centered around synchronization
      - kernel layer
        - grid planning, load balancing, thread marshalling
      - device layer
        - host-side

2025.03.05

- CUTLASS / CuTe layout algebra
  - mode of layout = axis = length-1 layout
  - *coordinate* = logical position in shape: natural (aka h-D), or colexicogr. ordinal (aka 1-D), or mixed (sub-shapes -> ord.) (aka R-D, R=rank); *natural coordinate* = same (nested) tuple as shape; *index* = offset in layout (inner product of natural coordinate and stride)
  - confusing: formalism and default stride is column-major (colexicographical)
  - tuple-like and NumPy-like operations: sublayouts (fully expressive), concatenation, grouping, flattening, slicing
  - To Be Continued
    - complement
    - left/right inverse
    - composition
    - logical product
      - zipped, tiled
    - logical divide
      - zipped, tiled

2025.03.06

coalesced layout: remove modes (axes) of size (dim) 1, remove modes that traverse contiguously with the preceding mode: (s0:d0, s1:s0*d0) --> (s0*s1, d0)

B by itself has its image disjoint from A except for position 0 -- because of disjoint coordinates when embedded in C=(A,B)

sorted layout: strides are non-decreasing

complement of layout A wrt. size M: layout B such that (A, B) (concatenated) is contiguous (dense) of size M; i.e. B fills the gaps of A due to strides

CuTe layout algebra

To Be Continued

(functional) composition of layouts: R(c) := (A o B)(c) := A(B(c))

composition is left-distributive with concatenation, so focus on B with 1 mode

R has coordinates compatible with B

coalesce A and let B=N:r, then A o B has the modes (dimensions and strides) from the middle of A with first and last mode adjusted, such that size(A o B)=N, and the first stride = the corresponding stride of A * (r / the cumulative size of modes to the left)

(s0:d0) o (s1:d1) = s1:(d0*d1) every d0th element, s1 total

2025.03.07

complement(A,M) has one more mode than A: leading mode (d0:1) where d0 is leading stride of A; except d0=1 same num of modes

Tilers add expressivity via hierarchical layouts:
layouts: A o (B1, B2) = (A o B1, A o B2)
tilers: (A1, A2) o <B1, B2> = (A1 o B1, A2 o B2)

A o B selects parts of A and reshapes them

zipped divide: puts the iterator modes coming from different pieces of the tiler into one toplevel sublayout

Logical divide:
A / B := A o (B, complement(B, size(A)))
split A into elems pointed to by B, and the rest

CuTe layout algebra

continued

logical dividing by a tiler keeps the hierarchical structure: (A1, A2) / <B1, B2> = (A1 / B1, A2 / B2)

in the hierarchical layout of A/B, the first toplevel mode is the tile, and the second is the iterator over tiles

tiled divide: puts the intra-tile modes of the tiler pieces into one toplevel sublayout

dividing by a multimode tiler: multidim tiles that look differently for different modes (axes) of A

layout<0>(zipped_divide(A, B)) = layout<0>(tiled_divide(A, B)) = A o B

Logical product:
A x B := (A, $\underbrace{\text{complement(A, size(A)*cosize(B))}}_{A^*}$ o B)

A is the tile, B is the number and order of repetitions, A* is the available repetition layout

logical_product is tricky because B needs to be designed for A; blocked_product simplifies this

raked_product disperses the tiles

2025.03.08

tensor = layout + engine + [non]owning
engine = RAM iterator + mem tag
mem tag = global | shared | registers

tensor slice: new non-owning tensor with offset iterator, and new (restricted) layout

Thread-Value partitioning: a layout mapping threads + per-thread values to coordinates

partitioning: zipped_divide then slice;
inner_partition, local_tile: give each e.g. threadgroup a tile;
outer_partiotion, local_partition: iterate at fixed position / slice of the tiles

CuTe: tensors and algorithms

since CuTe can represent datatype, mem tag, shape and stride at compile time, algorithms can specialize to hardware-specific instructions

copy: the compiler-selected impl. may need a specific explicit synchronization

the type-inferred default impl can be overriden

copy_if: copy with a mask tensor

gemm: accumulating, one of: element-wise product, outer product, batched outer product, matrix product; selected by a type param

axpby: y := a*x+b*y (generalizes FMA)

fill, clear memory

Generics:

transform: map in-place

fold an operation (C++17 style)

2025.03.10

hardware instruction levels:
- single thread (e.g. FMA)
- quadpair (Volta: V100)
- single warp (Ampere: A100, RTX 30x0)
- warpgroup (Hopper: H100)

quadpair: kinda 8 thread tensor core, i.e. 4 QPs per warp; QP 0 is threads 0-3 & 16-19 etc.

not introduced, but much optimized

128 1D threads

mnemonic: D := A*B + C

MMA and Copy atoms: expose specific PTX instructions with a unified, templated interface

for a specific PTX operation, provide A,B,C,D types and layouts

CuTe Matrix Multiply

simplest example: tiles over threadblocks aka CTAs

often represents inputs as (M,K),(N,K) rather than (M,K),(K,N)

M/N/K-major: has stride 1 in the M/N/K mode

```
auto cta_coord = make_coord(blockIdx.x, blockIdx.y, _);
Tensor gA = local_tile(mA, cta_tiler, cta_coord, Step<_1,X,_1>{});
```

example: independently spcifying shared mem layouts, partitioning patterns, PTX instruction via TiledCopy and TiledMMA

static templated args eg. bM=128 x bN=128 x bK=8

selects sublayouts except for pos. X for both cta_tiler and cta_coord

```
auto tA = make_layout(make_shape(Int<32>{},Int<8>{}));
Tensor tAgA/tAsA = local_patition(gA/sA, tA, threadIdx.x);
...   copy(tAgA(_,_,k_tile), tAsA); cp_async_fence();
      cp_async_wait<0>(); gemm(tCsA, tCsB, tCrC); __synctreads();
```

builders take Copy_Atom<...> / UniversalFMA<TC,TA,TB> and threads (and values for TiledCopy) layout(s), then TiledX.get_slice(threadIdx.x) to extract tAgA, tAsA, tCsA...

2025.03.24

The Ultra-Scale Playbook Table of Contents
one GPU training
recomputing activations
profiling compute and communication
data parallelism
overlap grad reduction with backward pass
bucket grads into concat tensors for reduction
ZeRO: 1 (optim. states), 2 (+ grads), 3 (+ params)
tensor parallelism
along hidden dim
seuence parallelism: along input dim
expert parallelism
context parallelism
ring attention
zig-zag ring attention
pipeline parallelism
Zero Bubble and DualPipe
finding best training config
step 1: fit step in mem
step 2: target batch size
step 3: throughput
fusing, threading, mixed precision
Flash Attention 1-3

I'll go through these

2025.03.26

- picotron
  - tensor parallel
    - matmul → nonlin → matmul
    - split is best: first by columns, then by rows
      - output proj.
    - vocab embedding
      - mask input on GPU to only compute what's in the GPU's dictionary
        - and shift
  - dataloader
    - tokenize_dataset
  - process group manager
    - PP*TP*DP grid of GPUs
    - selects subsets of GPUs for a given parallel schema
    - sets up one process per GPU
  - data parallel
    - distributed sampler
    - all_reduce to average gradients
    - dataparallel buckets
      - grad accumulation: reduce mem by not parallelizing over the whole batch
        - manual grad_acc in a temp var to reduce at higher prec FP32

parallelisms:

data → batch dim

tensor → hidden dim

pipeline → model layer dim

example:

DP=2

TP=2

PP=2

→ 8x parallel