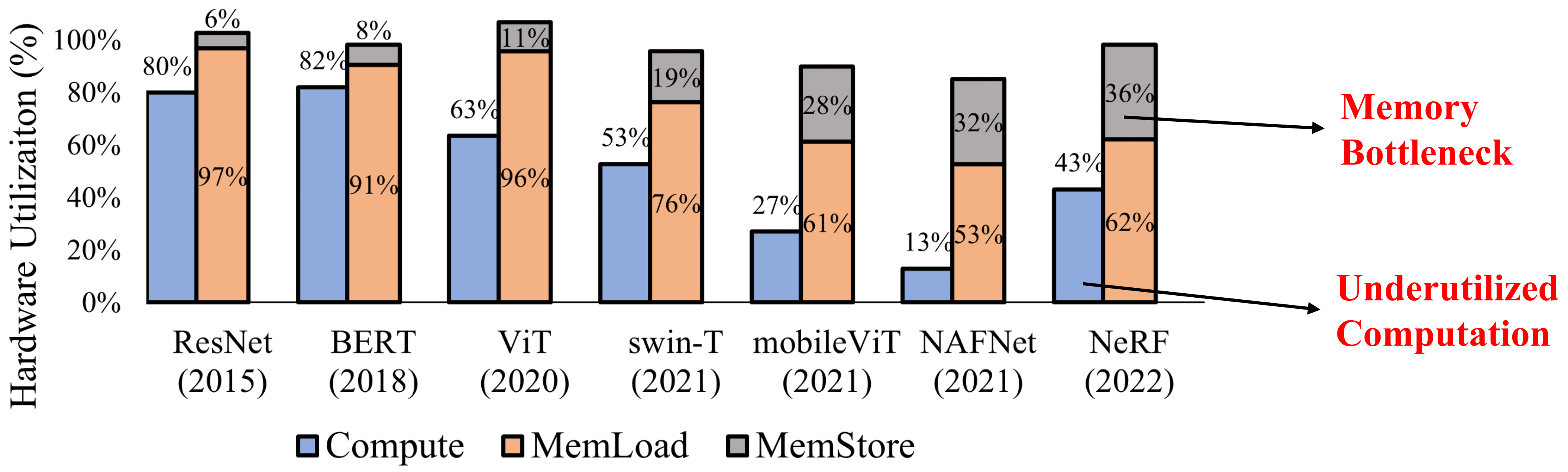
WELDER: Scheduling Deep Learning Memory Access via Tile-graph

Yining Shi[†‡], Zhi Yang[†], Jilong Xue[‡], Lingxiao Ma[‡], Yuqing Xia[‡], Ziming Miao[‡], Yuxiao Guo[‡], Fan Yang[‡], Lidong Zhou[‡]
†Peking University, ‡Microsoft Research

Modern DNNs Being Increasingly Memory Intensive

- Increasing needs to process higher fidelity data
  - E.g., larger images, longer sentences, high-definition graphics
- Memory throughput increased much slower than compute core
  - E.g., TensorCore could impose larger pressure on memory

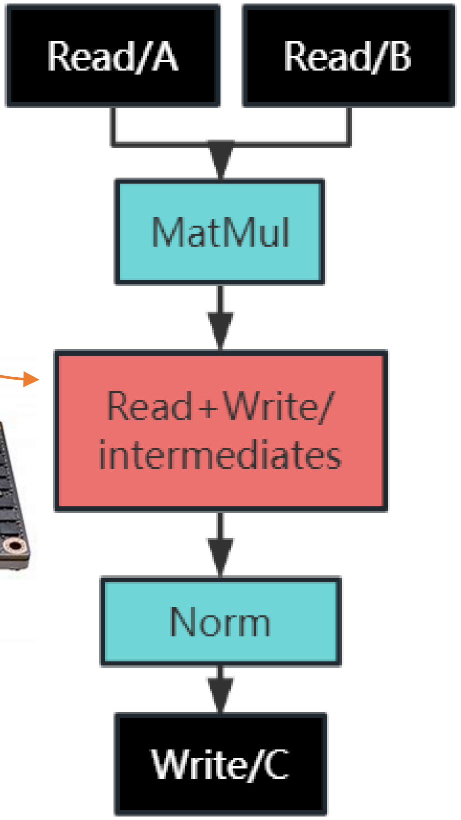


~**96%** memory bandwidth utilization, while only ~**51%** for computing core

Call For Extreme Data Reuse Optimization

- Memory-intensive operators: Element-wise, Normalization, Softmax, DW/PW-conv … A lot of memory access on intermediate results.
- Memory hierarchy in DNN accelerators (high-speed cache)

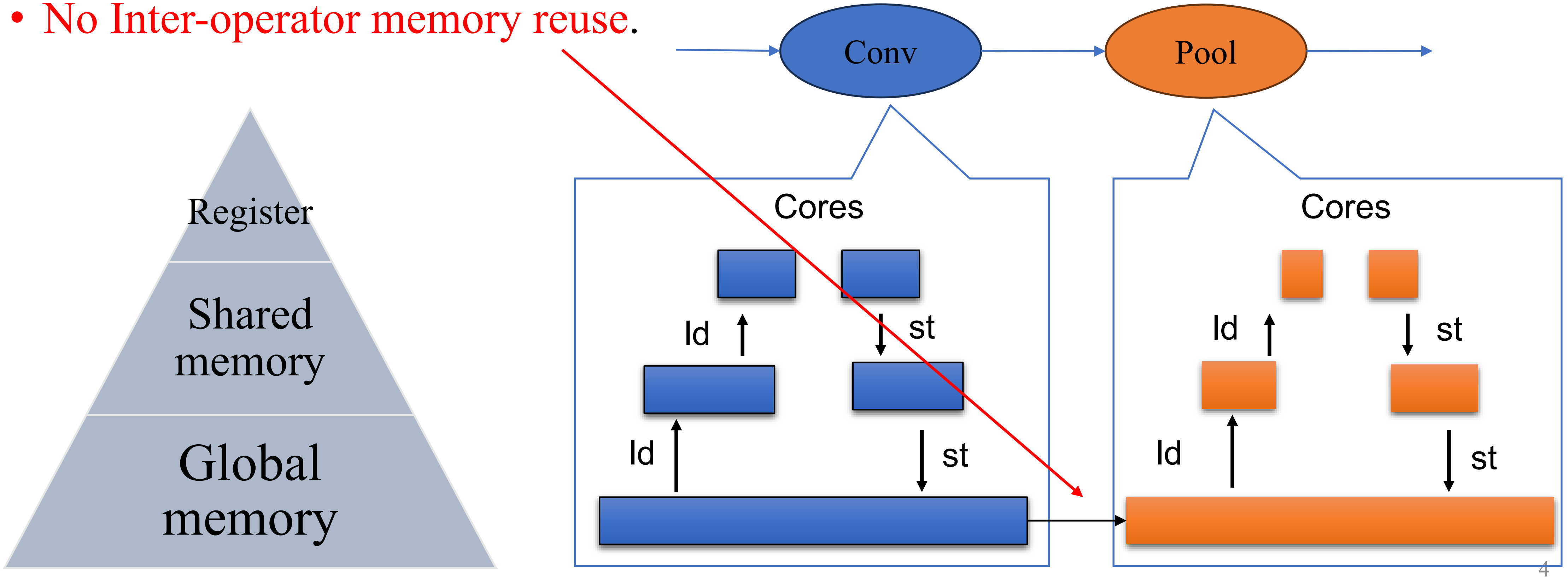
Device	Memory hierarchy
GPU (CUDA)	Global memory -> Shared memory -> Registers
GPU (ROCM)	Global memory -> LDS -> vGPR
GraphCore / SambaNova	Global memory -> Local buffer
Multi-GPUs	Host memory -> Device memory



- Call for a systematic inter-operator co-schedule approach to reuse intermediate results.
  - Place element-wise on Registers & regional operators (e.g., Normalization) on Shared memory

Current Practice: Focus on Intra-operator Data Reuse

- High performance kernels use multi-level tiling to evenly partition the workload onto computation cores. (e.g., CUTLASS, Triton, TVM, Roller …)
- Improves intra-operator memory reuse by caching data tiles on each memory layer.
- No Inter-operator memory reuse.



Opportunity: Explore Inter-operator Data Reuse

- **Tile-graph**: a tile-level abstraction to enable graph-level data tiling.
- By default, all operators are connected on the lowest memory layer.
- Tile-graph allows two operators to be connected at a higher memory layer

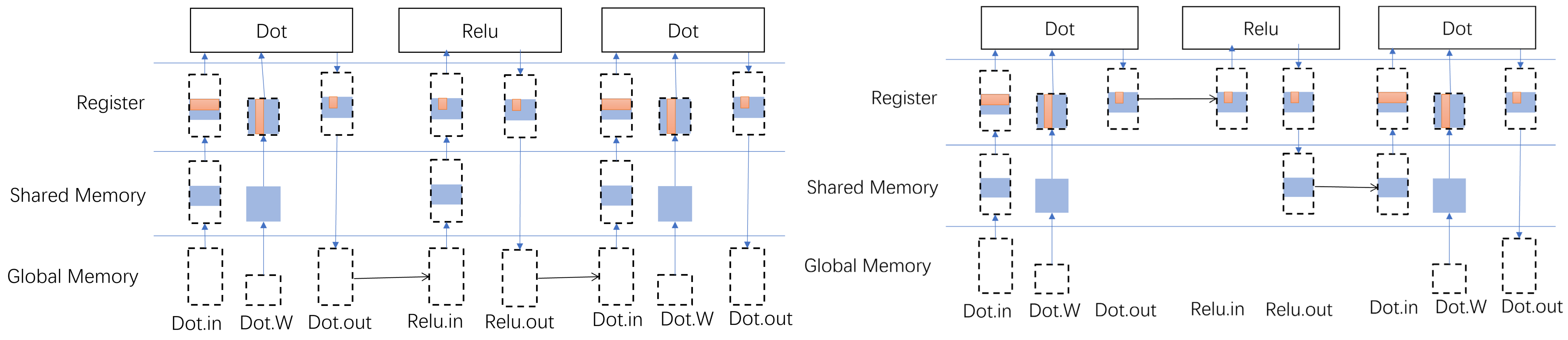


Fig. Left :Original, Right: Connect Relu on Register layer and connect second Dot on shared memory layer.

What’s the Challenges?

- Huge optimization space
  - Collectively schedule connection-layer and the tile-shape of multiple connected operators

- Conflict tile shape across operators
  - Operators require different tile shape and cannot be directly connected

Welder System Overview

- **Frontend**
  - DNN models -> Tile-graph
- **Tile-graph Scheduler**
  - Graph-connecting
  - Sub-graph scheduling
- **Code Generation**
  - Schedule -> Fused kernel code

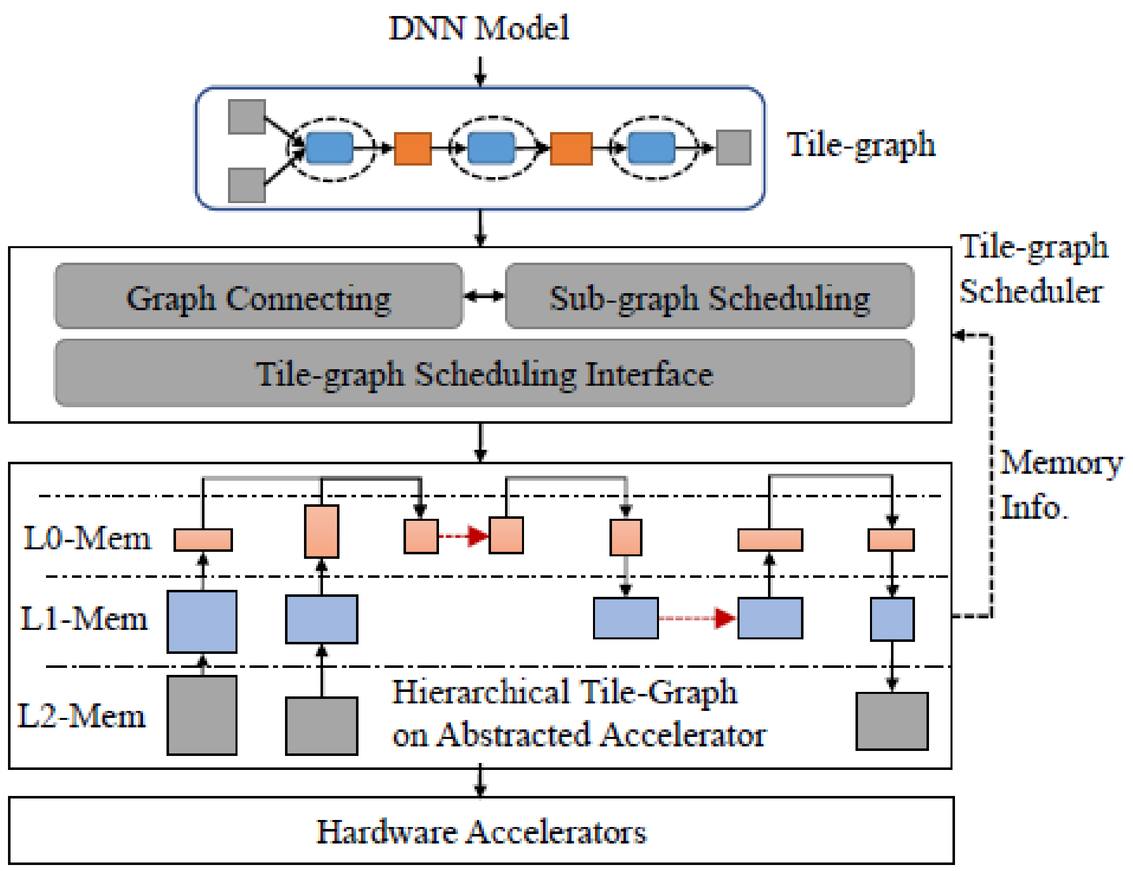
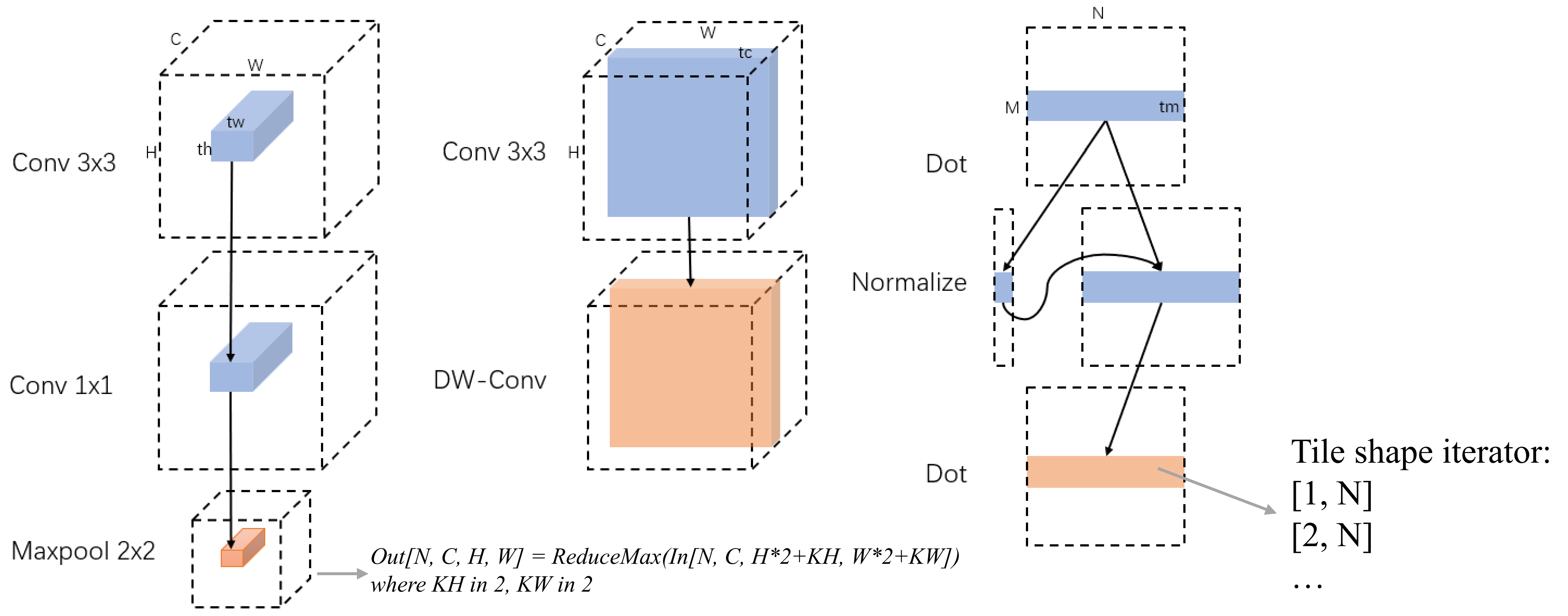


Fig. Welder Overview

Design I: Resolve Conflict by Tile Propagation

- Within a subgraph, tile can be propagated with regard to the output node's tile shape.



*Out[N, C, H, W] = ReduceMax(In[N, C, H*2+KH, W*2+KW])*
where KH in 2, KW in 2

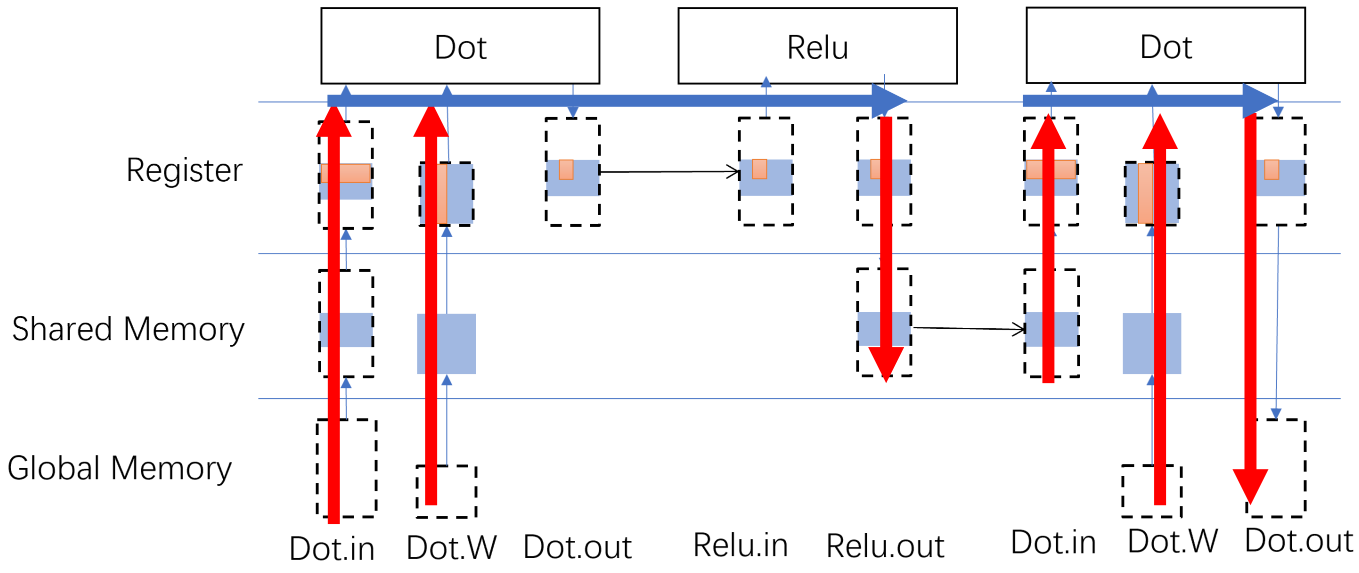
Tile shape iterator:
[1, N]
[2, N]
…

- Dependent Regions are inferred by analyzing the node's Tensor Expression.

Design II: Lightweight Traffic Cost Model

- Tile graph's execution can be viewed as data tiles moving vertically (LOAD/STORE) and horizontally (COMPUTE).

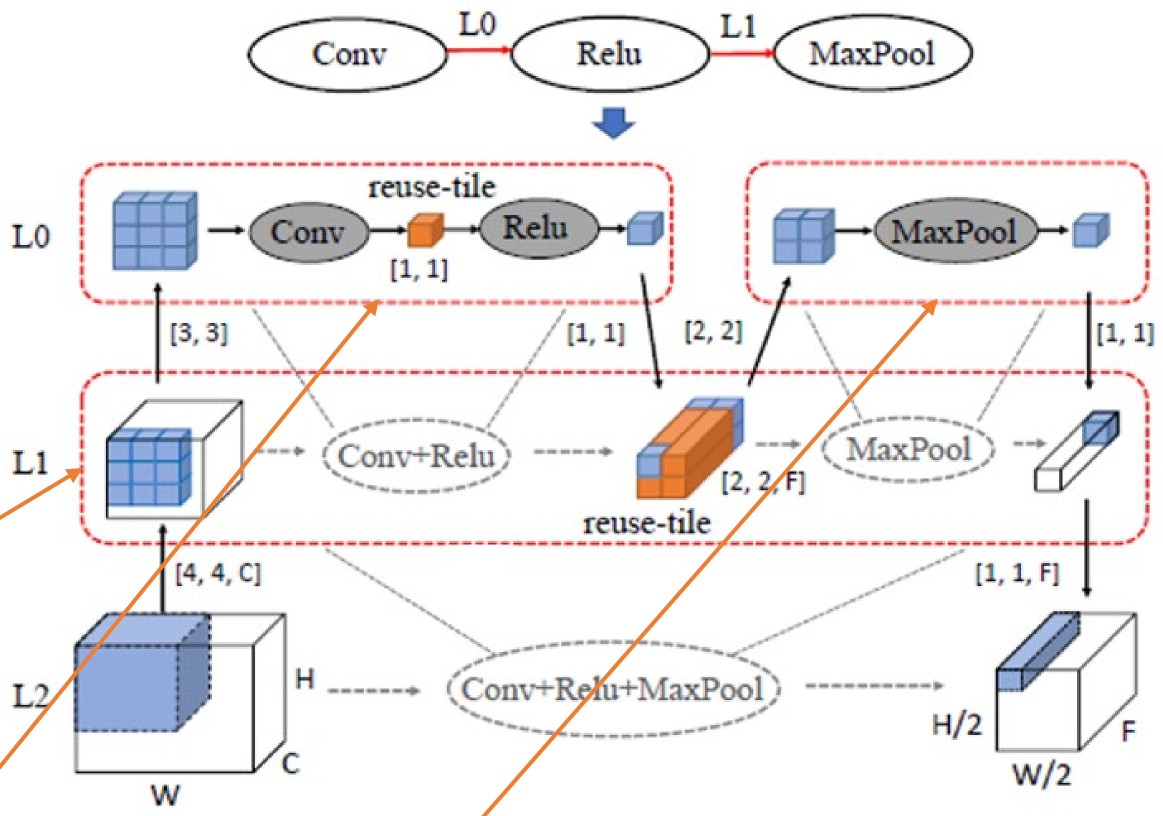
- Memory intensive workloads can be optimized by minimizing the data tile's vertical movement (Traffic).

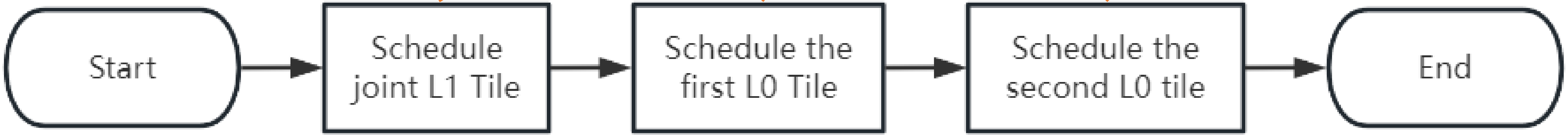


Design III: Decouple Optimization Space

- (different layers) Traffic on L2->L1 is only dependent on L1 tiles, and is not related to L0 tiles.

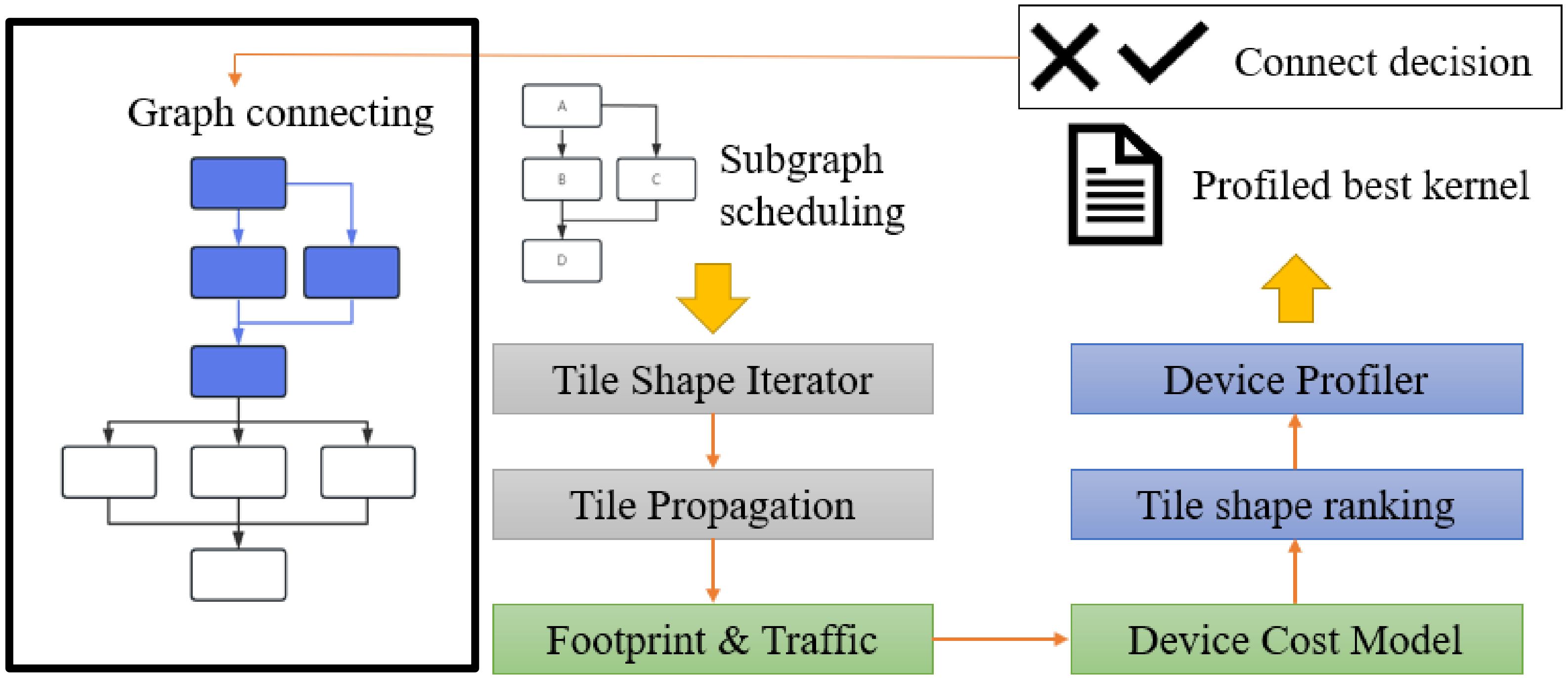
- (different operators) Traffic on L1->L0 can be optimized independently as they are not directly connected on that layer.

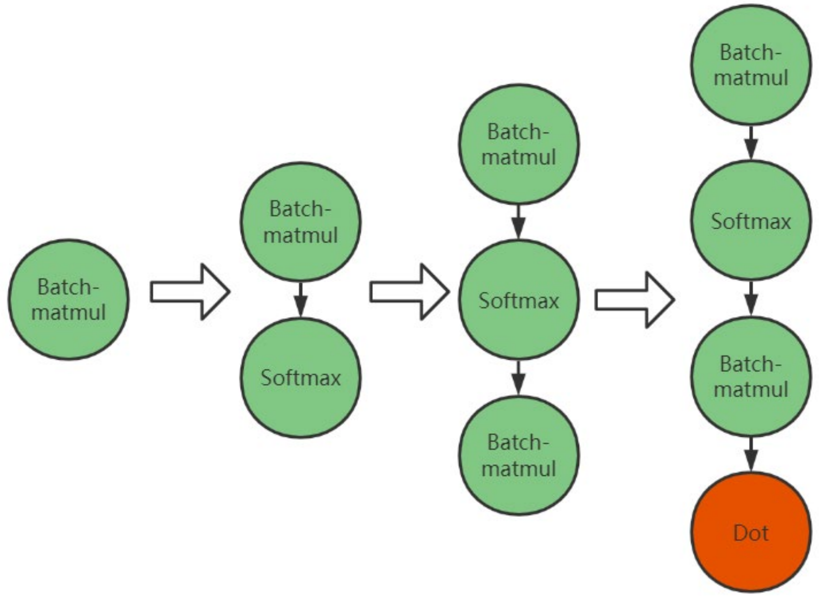




Welder's Workflow (I)

Follows the "**first connect then schedule**" method.

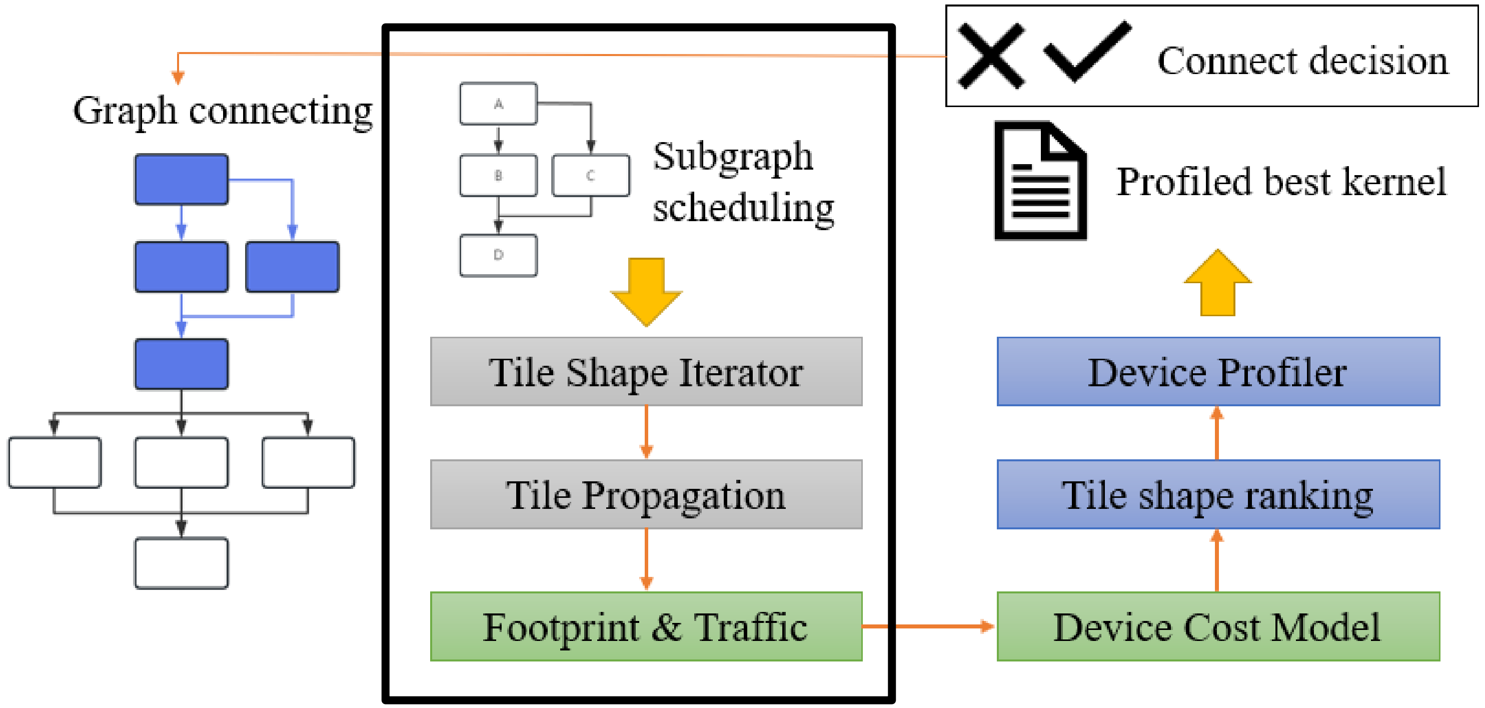




- **Graph Connecting**:
  - Try connecting on a higher memory level
  - Try lower memory layer if scheduling fails
  - Continue if potential fusion gain observed

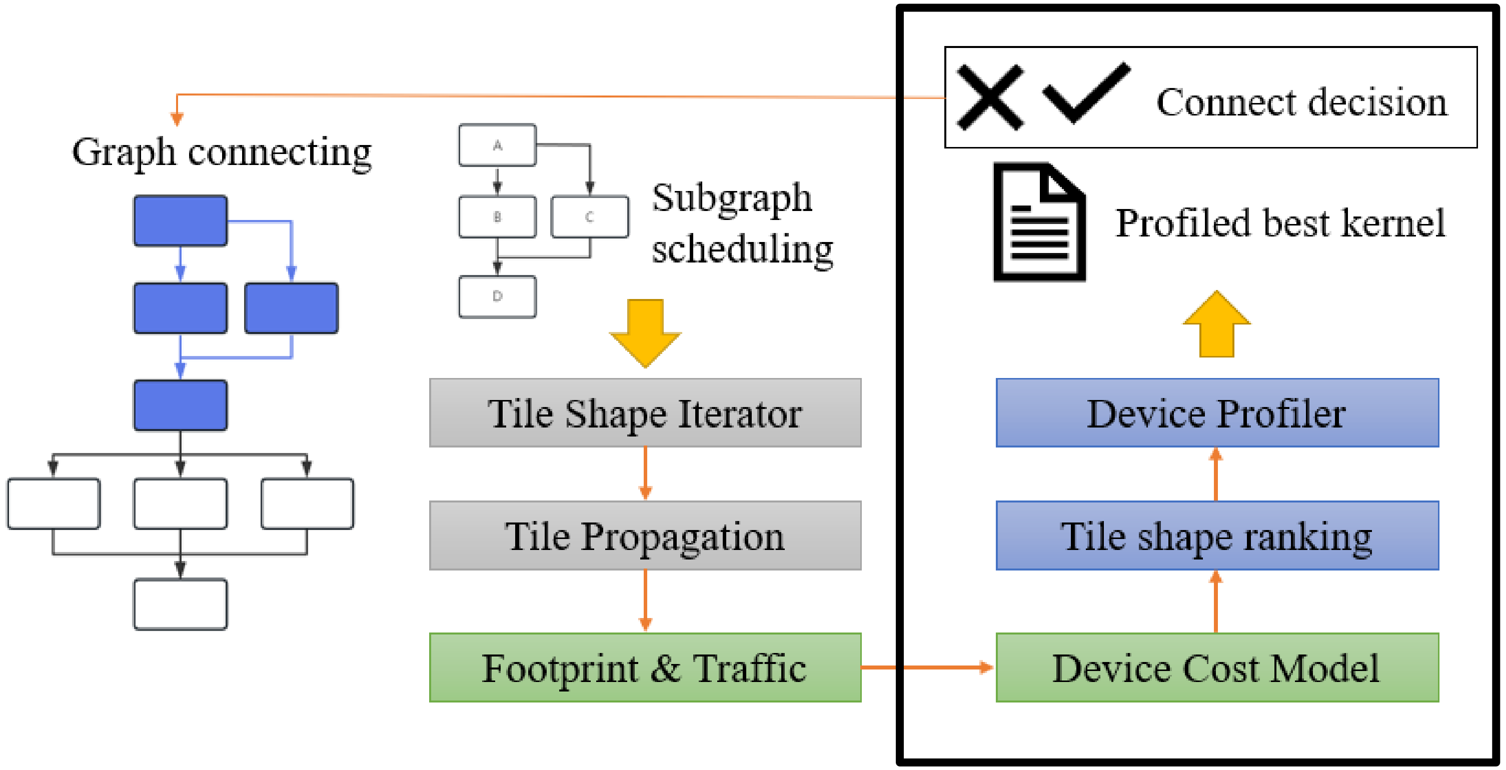
Welder's Workflow (II)

- **Tile shape Iterator:** try out different tile shape
- **Tile Propagation**: By analyzing the expression, Welder deduces read/write dependent region regarding to the given tile.
- **Footprint & Traffic, Device Cost Model**: static information used to roughly score and rank the tile.



Welder’s Workflow (III)

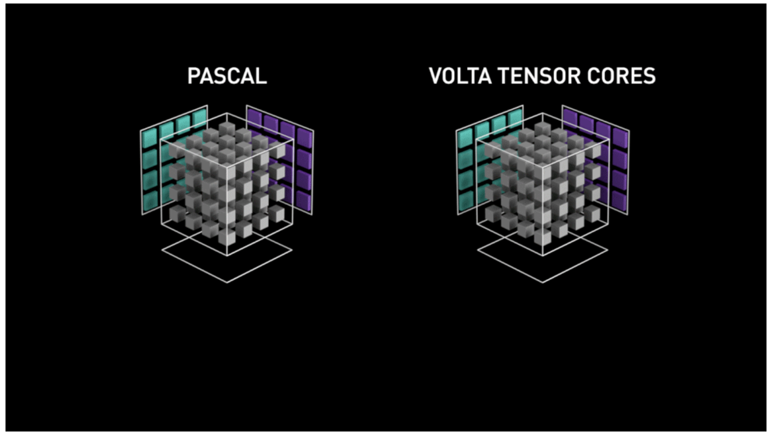
- **Device profiler**: Only a few tile config (top k from the device cost model) will be profiled.
- **Connect decision**: Connect if latency gain can be observed against unconnected case.



What About Compute-intensive Operators?

- Generating high-performance MMA kernels (while preserving the Tile-graph connection features)
  - Tensorized with high-performance block/warp level micro-kernel from expert libraries (e.g., CUTLASS)
  - Other adopted optimizations: Multi-stage software pipeline, layout swizzle …

*IR: C[N, M] += A[N, K] * B[K, M]*



Speedup Compilation

Tile-graph Initialization:

- Initialize some element-wise connection to register-level, saving additional cost to search for them.

Subgraph Cache:

- Generate a hash string for each subgraph
- Best schedule plan will be cached after tuning.
- For models like BERT, only one layer will be tuned.

Multi-process Support:

- Support parallel build and compile for each generated config.

Evaluation Setup

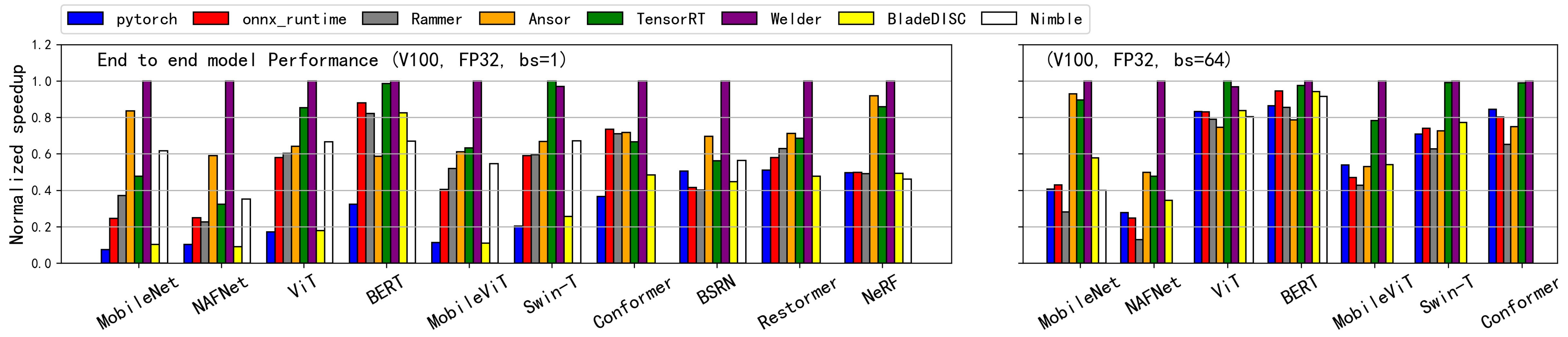
- Benchmarks:

Model	Type	Task	Year
MobileNet	CNN	Image Classification	2018
BERT	Transformer	NLP	2018
ViT	Transformer	Image Classification	2020
Conformer	CNN+Transformer	Speech Recognition	2020
MobileViT	CNN+Transformer	Image Classification	2021
Swin-Transformer	Transformer	Image Classification	2021
NeRF	MLP	3D-scene Generation	2021
NAFNet	CNN	Image Restoration	2022
Restormer	CNN+Transformer	Image Restoration	2022
BSRN	CNN	Image Super-resolution	2022

- Evaluated Baselines:
  - Pytorch v1.12.0
  - onnx_runtime v1.12
  - TensorRT v8.4
  - Ansor v0.9 (tuned 800 steps each task)
  - Astitch (Implemented in BladeDISC v0.3)
  - Rammer
  - Nimble
  - Welder (ours)

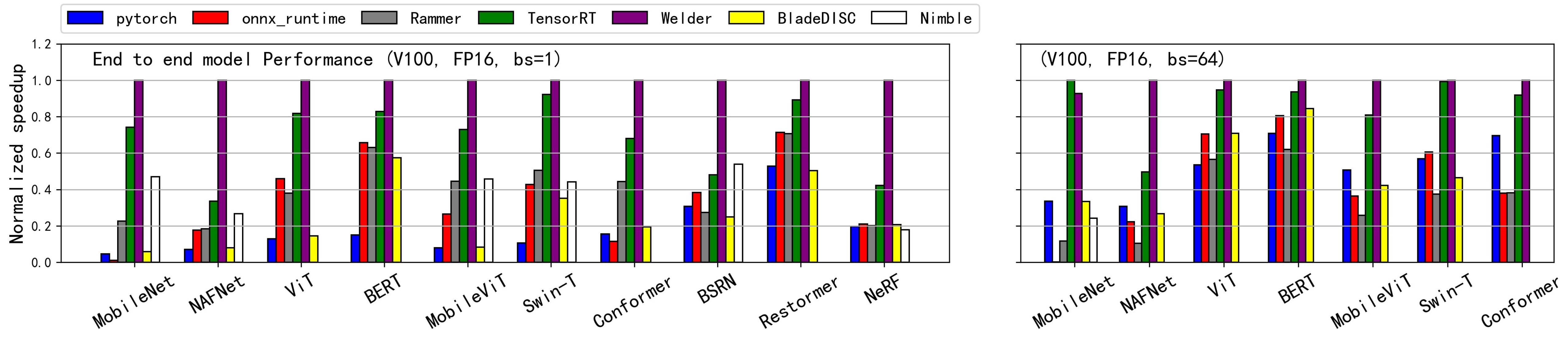
- Hardware:
  - NVIDIA V100
  - NVIDIA RTX-3090
  - AMD MI50

Performance on V100 (float+SIMT)



Performance on V100 (half+TensorCore)

On average: **~1.4x** to TensorRT
~ **2x** to onnxruntime **~1.5x** to Ansor



Other devices (AMD-MI50 & RTX-3090)

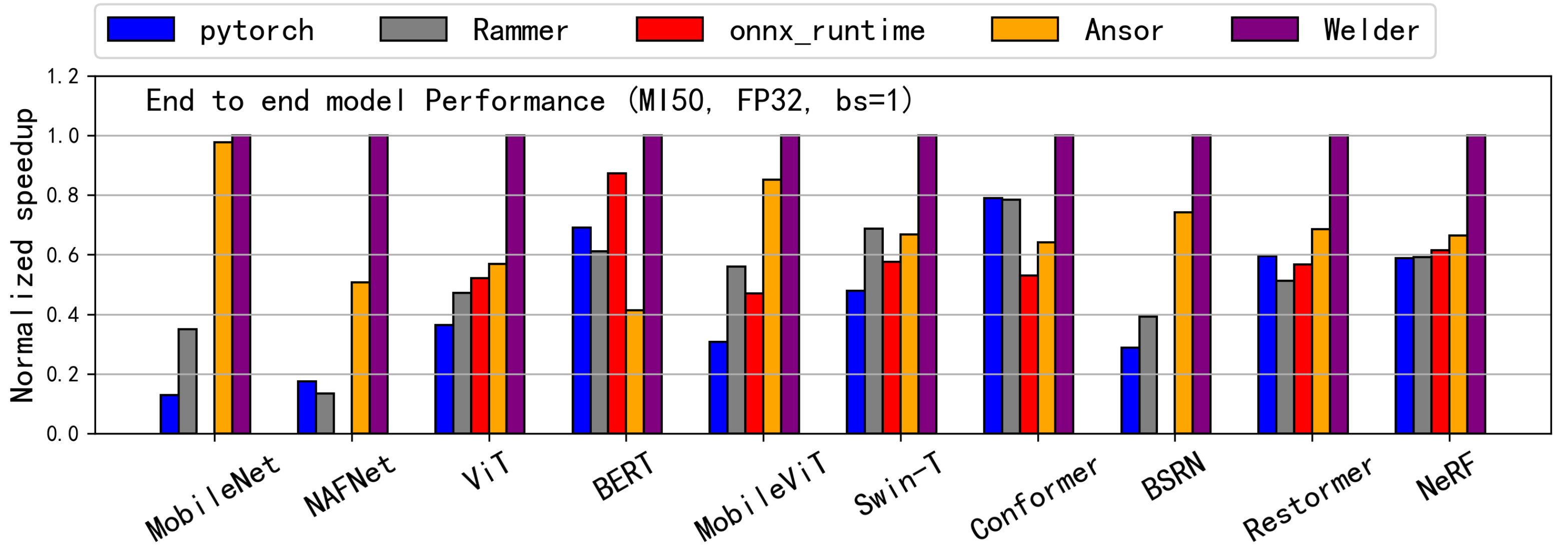


Fig : AMD ROCm MI50 GPU performance

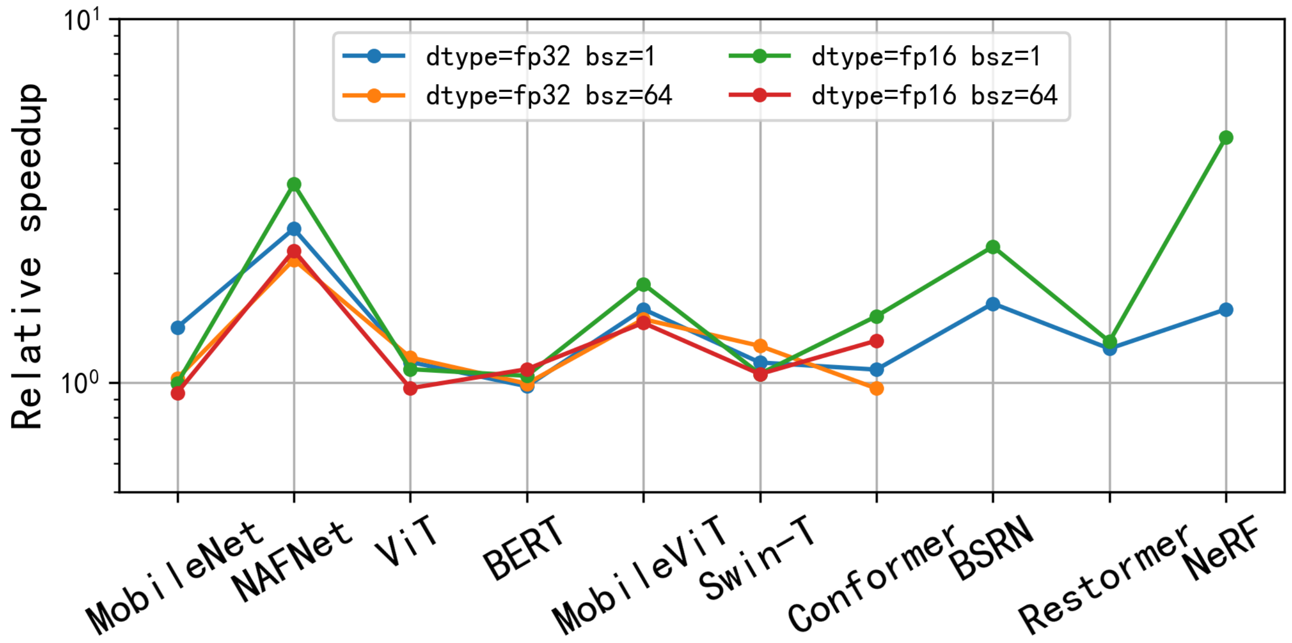
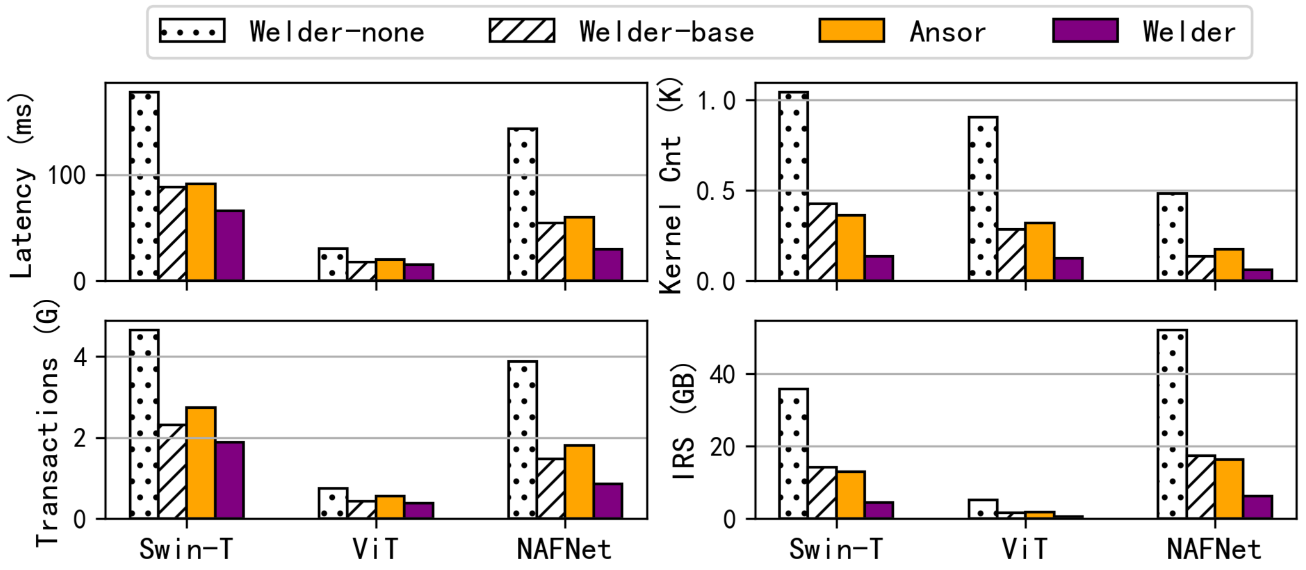


Fig : RTX-3090 performance (Compared with TensorRT) (**1.402x** speedup averaged)

Other Metrics

Memory Access Analysis:

- Welder can effectively save kernel counts, memory transactions, intermediate results as well as latency.



Welder-none : disable all tile connection
Welder-base : only enable element-wise kernel fusion

Compilation time:

Model	Ansor Time(s)	Ansor Trials	Welder Time(s)	Welder Trials
BERT	15285	8000	244	651
Mobilenet	45527	25600	561	927

~5 min for a model
~100x faster than Ansor

Conclusion

- **Increasing memory challenge** is observed in modern DNN inference workloads
- Welder proposes a **Tile-graph abstraction** that
  - Optimizes both inter- and intra-operator data reuses in a holistic space
  - Provides a general operator fusion mechanism
- Welder is exploring a systematic approach to take advantage of emerging trends in future model and accelerators

Thank you