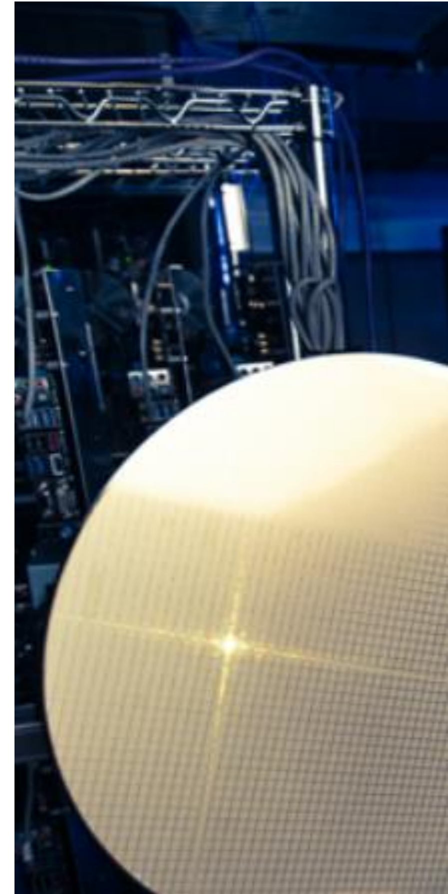
Software Optimization of High-Performance x86 with Integrated AI Coprocessor

Linley Fall Processor Conference
October 20 – 29, 2020

Mike Thomson
CPU, Neural and Performance Engineer
mthomson@centtech.com

Centaur Technology Background

- 25-year-old startup in Austin, owned by VIA Technologies
- We design, from scratch, low-cost x86 processors
  - Everything to produce a custom x86 SoC with ~100 people
    - Architecture, logic design, and microcode
    - Design, verification, and layout
    - Physical build, fab interface, and tape-out
- Designs shipped by Tier 1 OEM's



©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

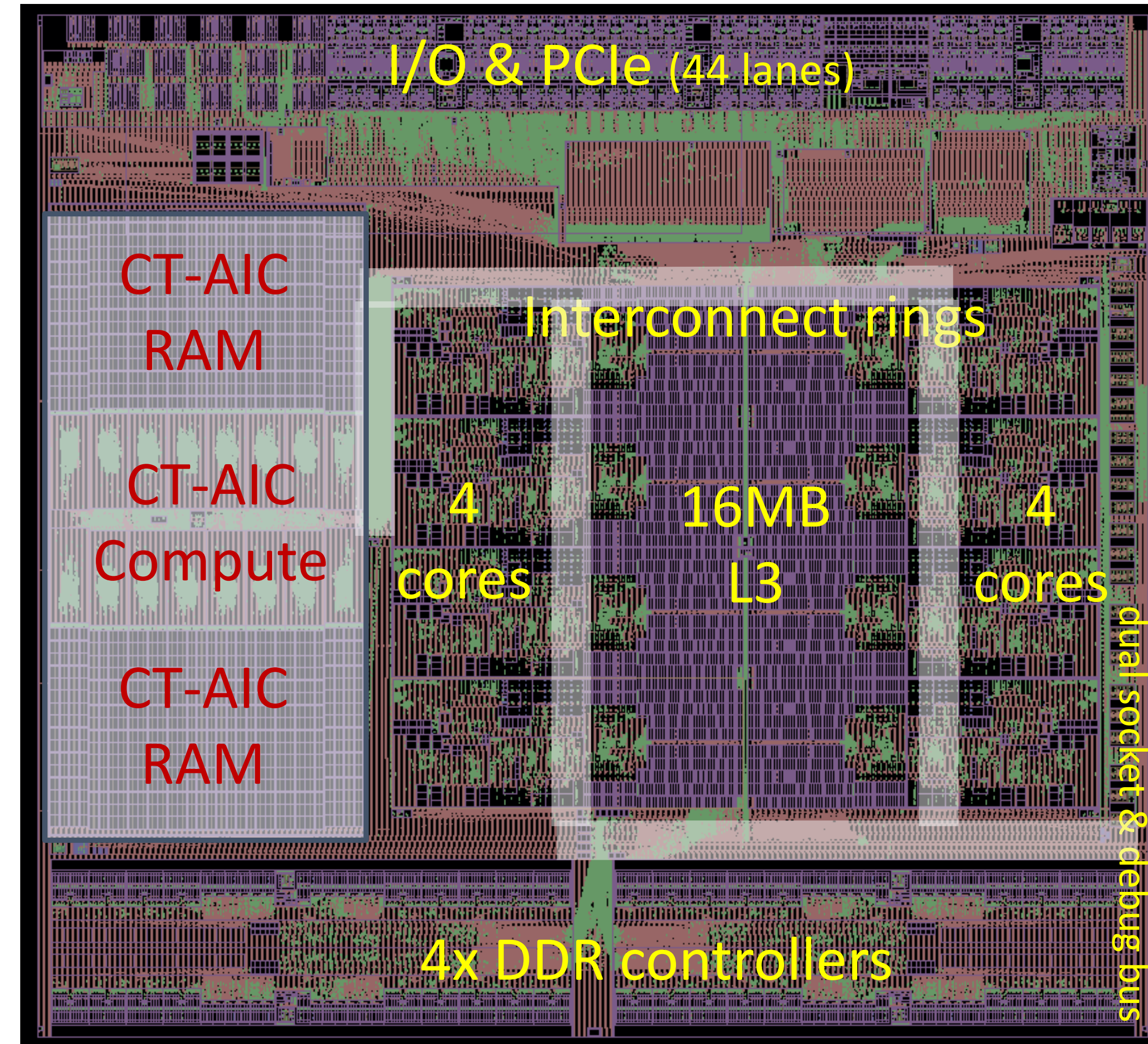
CHA Platform Overview

CHA: Server-class x86-64 SoC

- 8 cores (x86-64)
- 16MB L3 cache
- 4 DDR4-3200 controllers
- 44 PCIe lanes
- AVX512
- Multi-socket support
- Haswell-level IPC
- CT-AIC: Centaur AI Coprocessor

World's first high-performance x86 processor with integrated AI coprocessor!

194 mm² in TSMC 16 nm FFC technology



©2020 Centaur Technology. All Rights Reserved

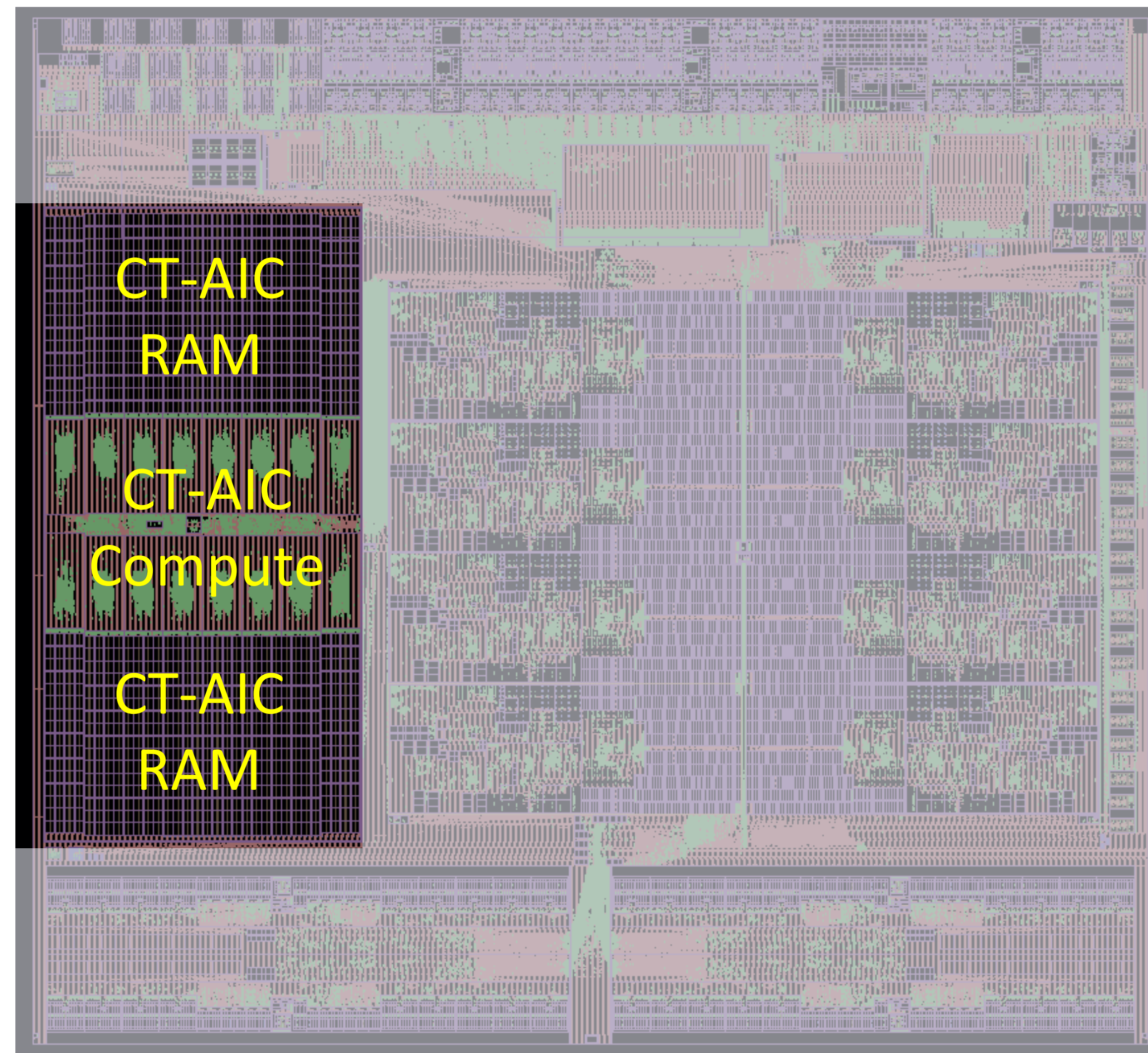
CenTaur Technology

Centaur AIC Overview

CT-AIC: Centaur AI Coprocessor

- 20 TOPS
- 4096-byte wide SIMD
- 16MB SRAM (20 TB/s)
- 2 DMA ports
- INT8, UINT8, INT16, BFloat16
- 2.5 GHz

Real, working silicon with MLPerf[1] benchmark suite results!



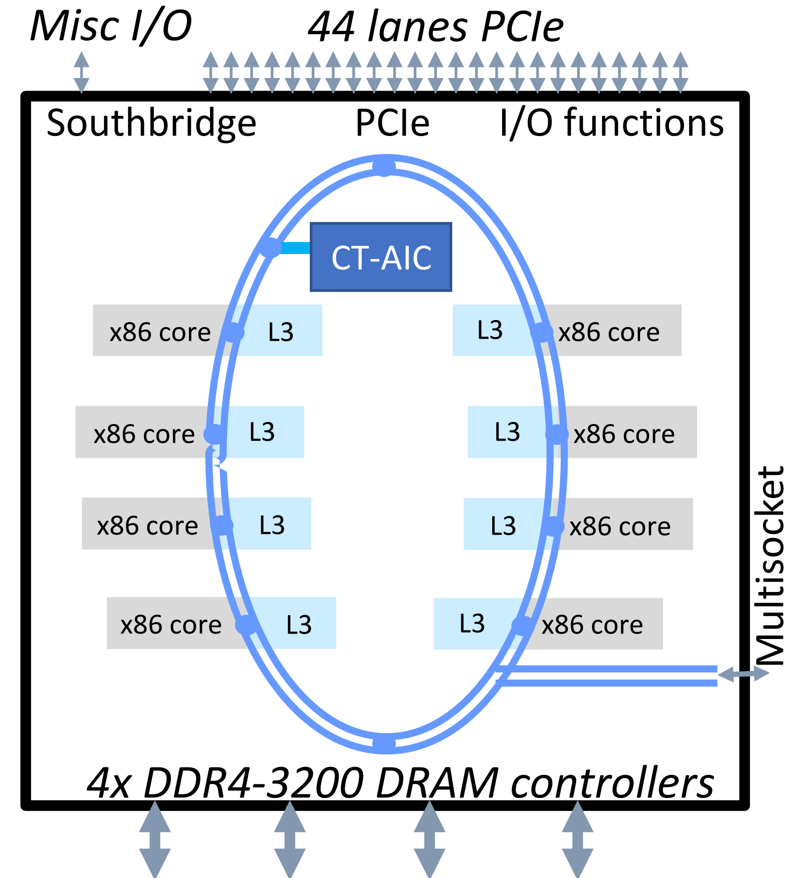
[1] MLPerf name and logo are trademarks. See www.mlperf.org for more information.

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

x86 + AIC = Performance, Flexibility

- Low latency
- High throughput
- Integrated solution
- Ubiquitous x86-64 platform
- Expandable ML performance
  - Multi-socket
  - Multi-system
  - PCIe: support for 3rd-party expansion cards



©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

MLPerf Inference Results – CT-AIC Speedup

Centaur AIC submission - v0.5 (Fall 2019)[1]

Benchmark	Latency (ms)	Throughput (IPS)
ResNet-50	1.05	1,218
SSD-MobileNet-V1	1.54	652
MobileNet-V1	0.33	6,042

Centaur AIC submission - v0.7 (Fall 2020)[2]

Benchmark	Latency (ms)	Throughput (IPS)
ResNet-50	1.05	1,254 (+3%)
SSD-MobileNet-V1	1.33 (-14%)	1,938 (+300%)

MobileNet-V1 is not part of MLPerf Inference v0.7, but Centaur's AIC is faster now[3]:

- Latency: 0.27 ms (18% improvement)
- Throughput: 7,220 IPS (19% improvement)

[1] MLPerf v0.5 Inference Closed SingleStream and Offline, Preview category. Retrieved from www.mlperf.org 27 January 2020, entry 0.5-32.
[2] MLPerf v0.7 Inference Closed SingleStream and Offline, Available category. Retrieved from www.mlperf.org 22 October 2020, entry 0.7-131.
[3] New MobileNet-V1 results not verified by MLPerf.

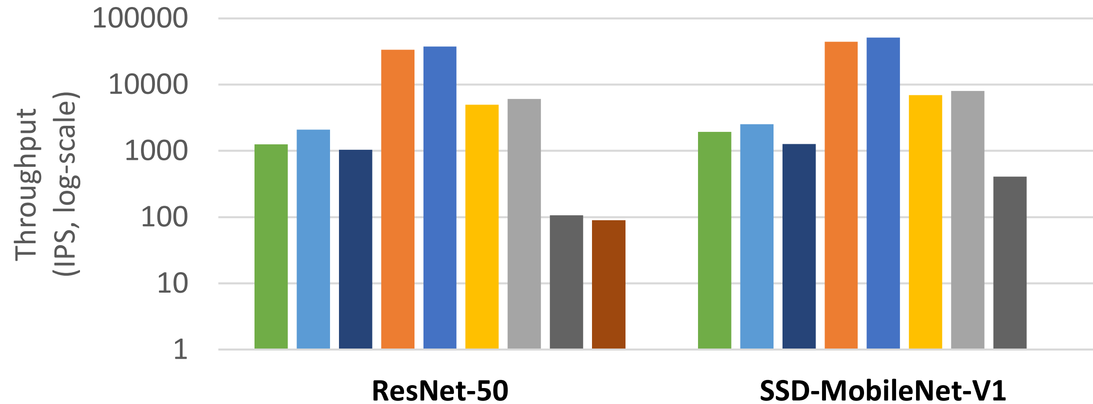
©2020 Centaur Technology. All Rights Reserved

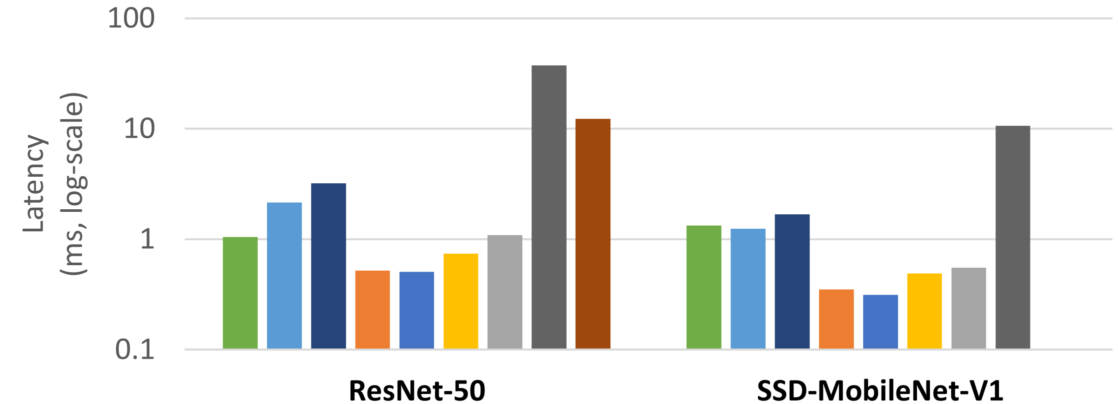
CenTaur Technology

MLPerf Inference v0.7 Results

Closed division, accelerator vendors self-identified as edge servers

(In case of vendor with multiple submissions for same accelerator, only including the fastest score)



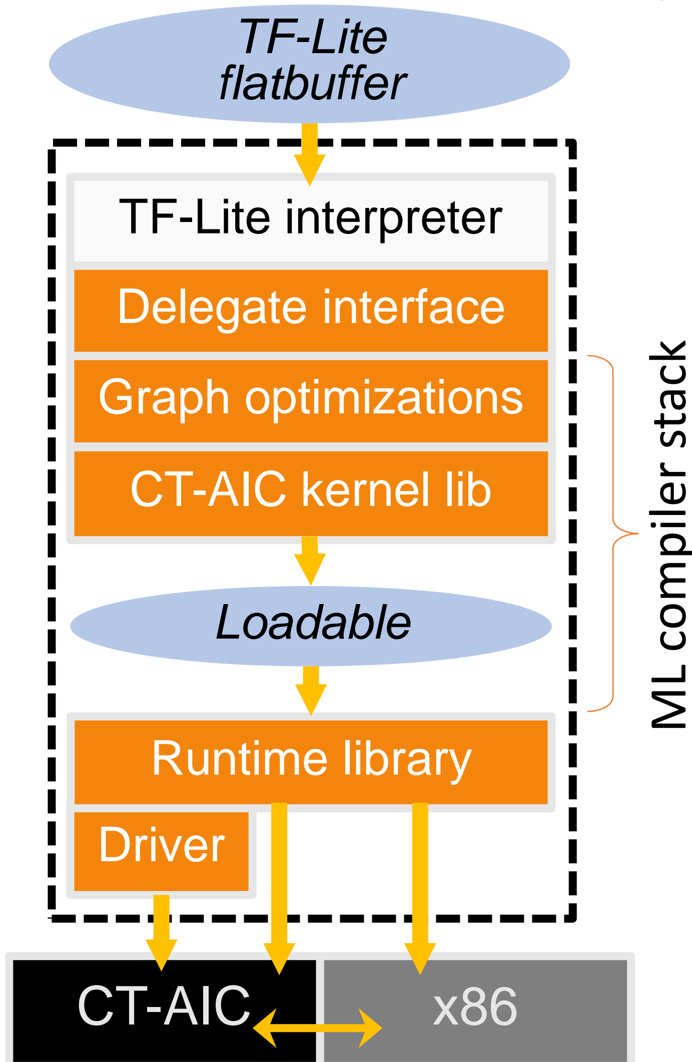


[1] MLPerf v0.7 Inference Closed SingleStream and Offline. IVA and Mobilint submissions are RDI category, all others are Available category. Retrieved from www.mlperf.org 22 October 2020, entries 0.7-131, 0.7-146, 0.7-147, 0.7-148, 0.7-149, 0.7-150, 0.7-151, 0.7-152, 0.7-153, 0.7-154, 0.7-155, 0.7-156, 0.7-157.

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Software Optimizations – Existing Stack



- Centaur's AIC: great MLPerf v0.5 → v0.7 speedup
  - Optimizations improved parallelism at high level:
    - TF-Lite interpreter
    - Delegate interface
    - Runtime library
- But what about our ML compiler stack?
  - Extremely optimized for existing workloads
  - Monolithic → crosses large semantic gap all at once
  - Has benefits, but also significant challenges
  - Centaur working on new MLIR-based ML compiler stack

©2020 Centaur Technology. All Rights Reserved

Centaur Technology

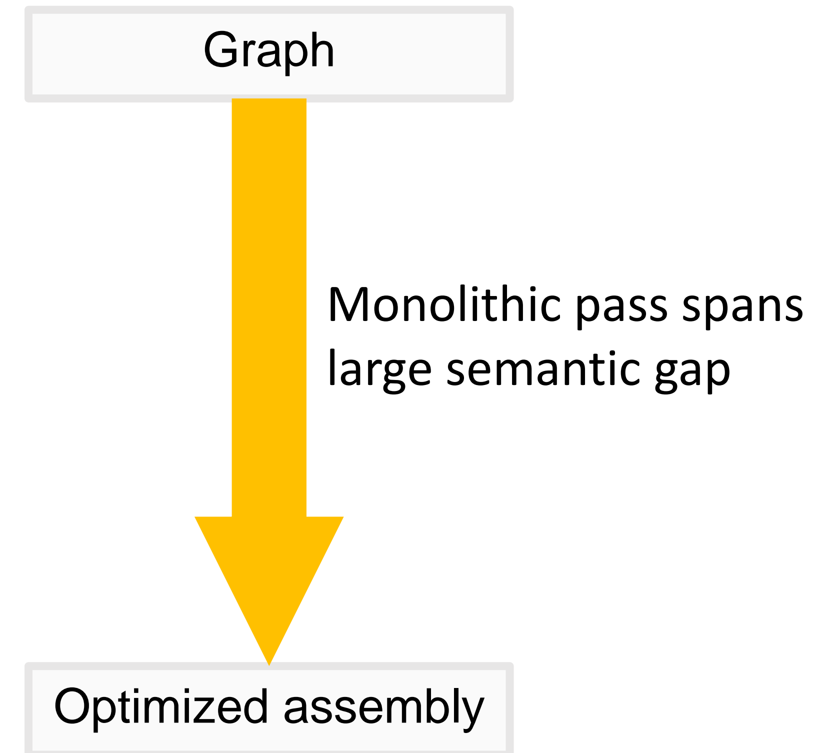
"Conventional" ML Compiler Stack

Companies moving away from "conventional" ML compiler stacks for good reason

Benefits: Hand tuned, extremely optimized for existing workloads

Challenges:

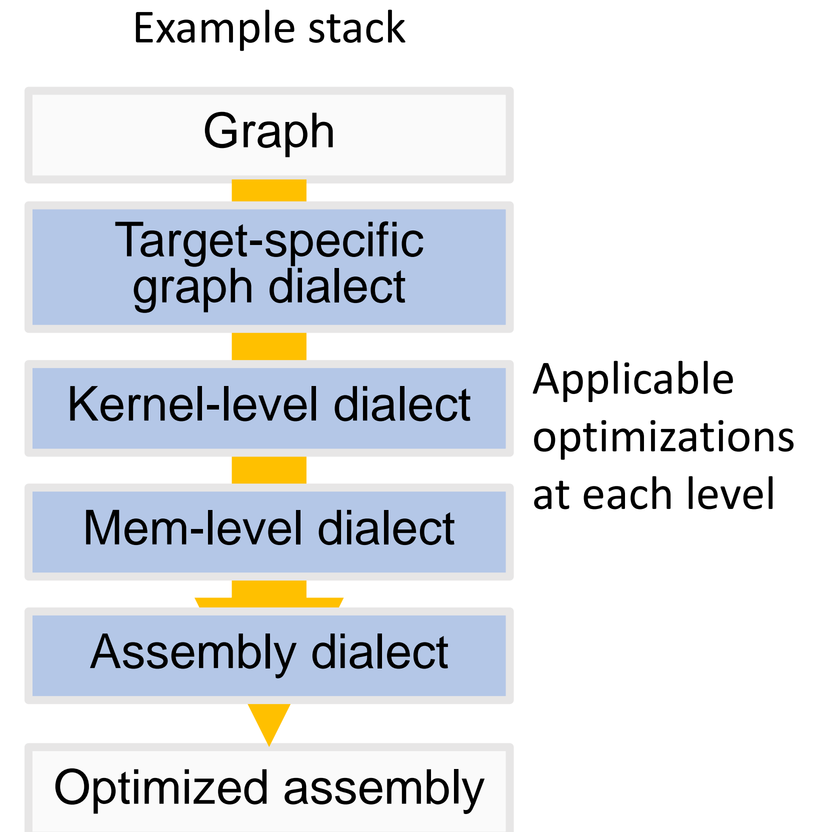
- Hand tuned, new workloads = huge effort
- Monolithic → large semantic gap
  - Must do all optimizations all at once
  - Low-level analysis difficult
  - Difficult to scale
  - Changes tend to break everything
  - Some optimizations become intractable
  - Low reusability across frameworks
  - … and more



©2020 Centaur Technology. All Rights Reserved

MLIR-based Compiler Stack

- LLVM project
- Multiple intermediate representation “dialects”
- Benefits:
  - Vast infrastructure already available
  - Implement as many or as few levels as needed
  - Different optimizations easier at different levels
  - Easy to add new op / feature support
  - Framework-agnostic
    - Implement once, reuse everywhere!
  - … and more!



©2020 Centaur Technology. All Rights Reserved

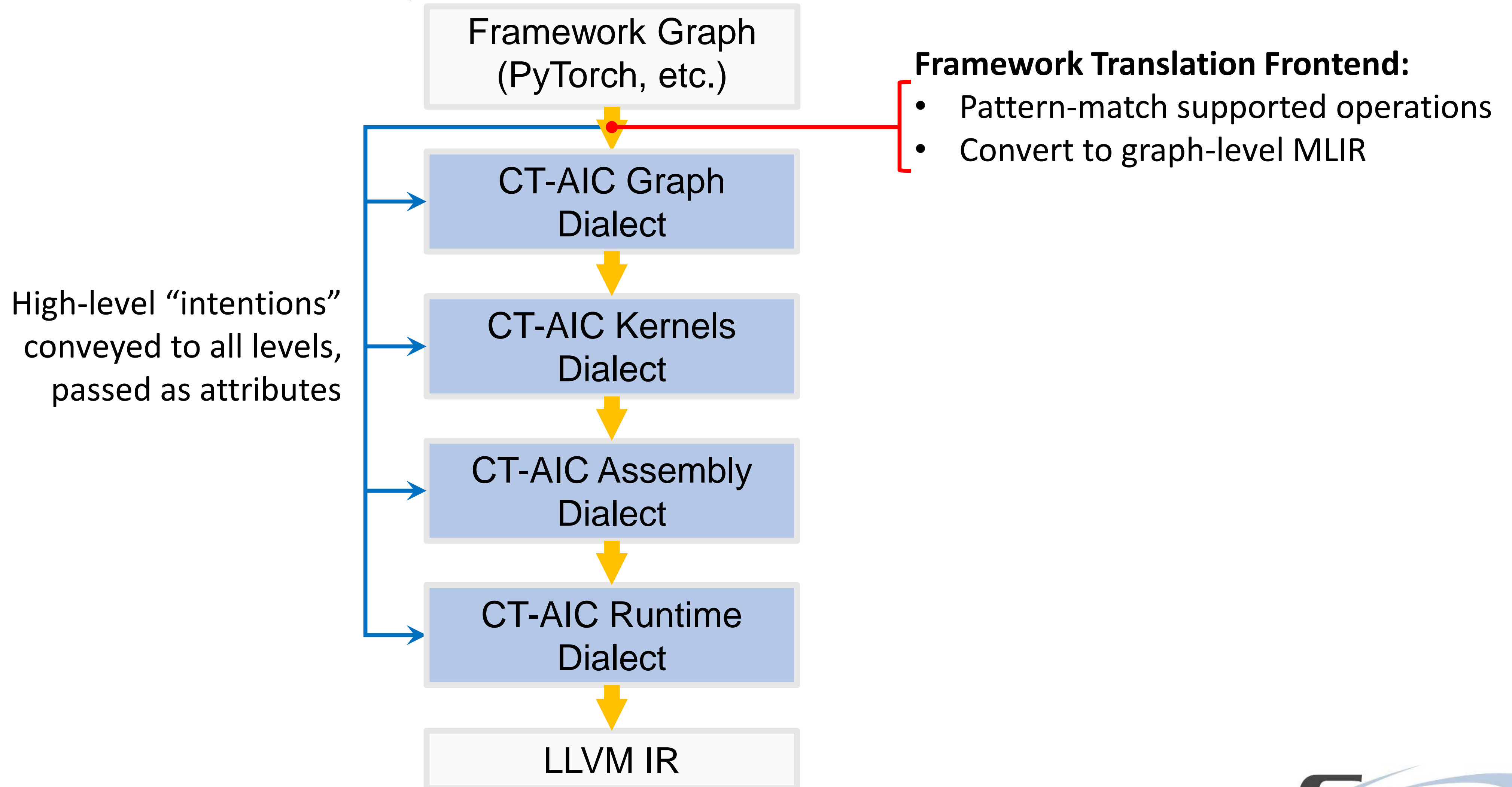
MLIR-based Compiler Stack

- MLIR (LLVM project) is not a passing fad
  - It helps solve problems common across many companies and domains
  - Extremely competent people from many companies contributing
  - Common problems already being solved, reusable
- Centaur is collaborating / contributing to the MLIR project
- MLIR's growing list of partners:

AMD	IBM	SambaNova
ARM	Intel / Habana	Samsung
Cerebras	Mediatek	Xiaomi
Google	NVIDIA	Xilinx
Graphcore	Qualcomm	

©2020 Centaur Technology. All Rights Reserved
CenTaur Technology

Centaur AIC Compiler Stack: MLIR



©2020 Centaur Technology. All Rights Reserved

Frontend Translation

```python
import torch
import aictorch # CT-AIC PyTorch frontend

def calibrate_and_convert_to_quantized(model, in_size, num_samples=10, min=-1.0, max=1.0):
    # ... Calbration / quantization code...

class QuantizableTwoConvModel(torch.nn.Module):
    def __init__(self):
        super(QuantizableTwoConvModel, self).__init__()
        self.qconfig = torch.quantization.get_default_qconfig("fbgemm")
        self.quant = torch.quantization.QuantStub()
        self.conv0 = torch.nn.Conv2d(4, 4, [3, 3], padding=1)
        self.conv1 = torch.nn.Conv2d(4, 4, [3, 3], padding=1)
        self.dequant = torch.quantization.DeQuantStub()

    def forward(self, x):
        qx = self.quant(x)
        conv0_out = self.conv0(qx)
        conv1_out = self.conv1(conv0_out)
        out = self.dequant(conv1_out)
        return out

# Calibrate and Quantize simple 2-conv model
in_size_nchw = (1, 4, 7, 7)
model = QuantizableTwoConvModel()
model = calibrate_and_convert_to_quantized(model, in_size_nchw)
sample_data = torch.rand(in_size_nchw)
traced = torch.jit.trace(model, sample_data)

# Compile the model for CT-AIC
aic_model = aictorch.compile(
    traced, {
    "input_shapes": [sample_data.shape],
    "output_tensor_infos": [ { "shape": [1, 4, 7, 7], }, ]
})
print(aic_model.mlir())
```

PyTorch → MLIR (graph-level)

```
module {
func @main(%X: memref<1x7x7x4xf32>, %out0: memref<1x7x7x4xf32>) -> ()

  %input.1 = "aic_graph.quantize"(%X) {} : (memref<1x7x7x4xf32>) -> (t
  %conv_0_weights = "aic_graph.pseudo_const"() { value = dense<"0x7f35
  %conv_0_bias = "aic_graph.pseudo_const"() { value = dense<"0x3ae2fff
  %35 = "aic_graph.conv2d"(%input.1, %conv_0_weights, %conv_0_bias) {d
  %conv_1_weights = "aic_graph.pseudo_const"() { value = dense<"0x3ec9
  %conv_1_bias = "aic_graph.pseudo_const"() { value = dense<"0x6708000
  %42 = "aic_graph.conv2d"(%35, %conv_1_weights, %conv_1_bias) {depthw
  %22 = "aic_graph.dequantize"(%42) {} : (tensor<1x7x7x4x!quant.unifor

  "aic_runtime.tensor_store"(%22, %out0) {} : (tensor<1x7x7x4xf32>, me
  return

} // main
} // module
```

(Attributes truncated for space)

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Frontend Translation

- Centaur's AICtorch is custom CT-AIC frontend
- NVIDIA's open-source TRTorch was huge help for initial efforts to parse / convert
- LLVM's incubator project NPComp is slated to include frontend translation that we can use
- MLIR translation is agnostic to PyTorch or any other framework
  - TensorFlow and TF-Lite are moving to become MLIR-based
  - Multiple frameworks, one CT-AIC MLIR compiler stack!

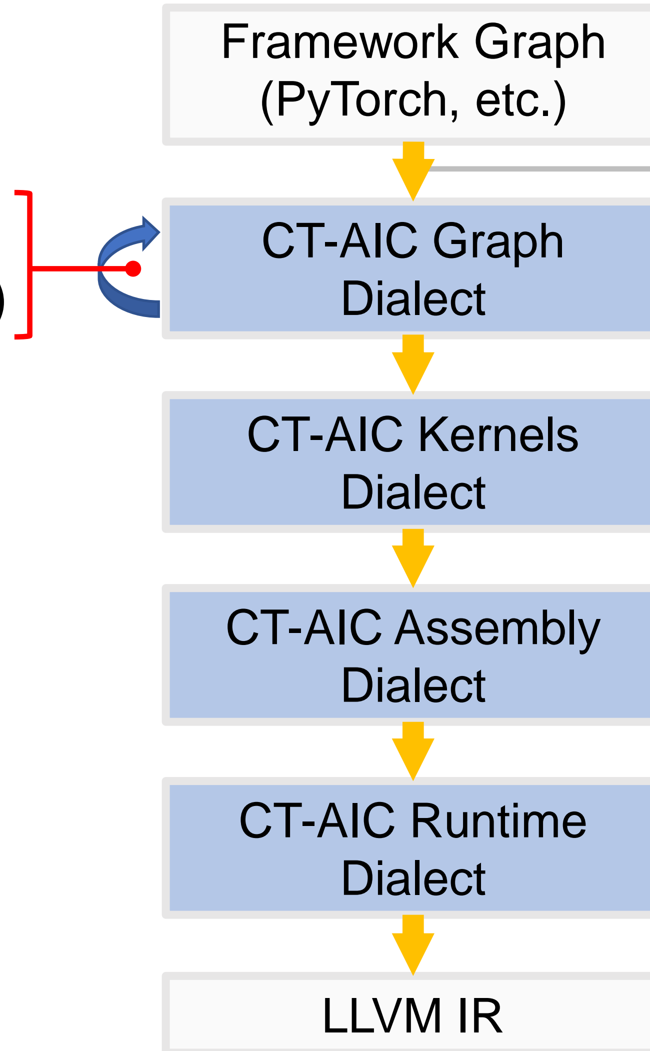
©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Centaur AIC Compiler Stack: MLIR

CT-AIC Graph:

- Quantization constant propagation
- Merge high-level ops (Conv / ReLU, etc.)



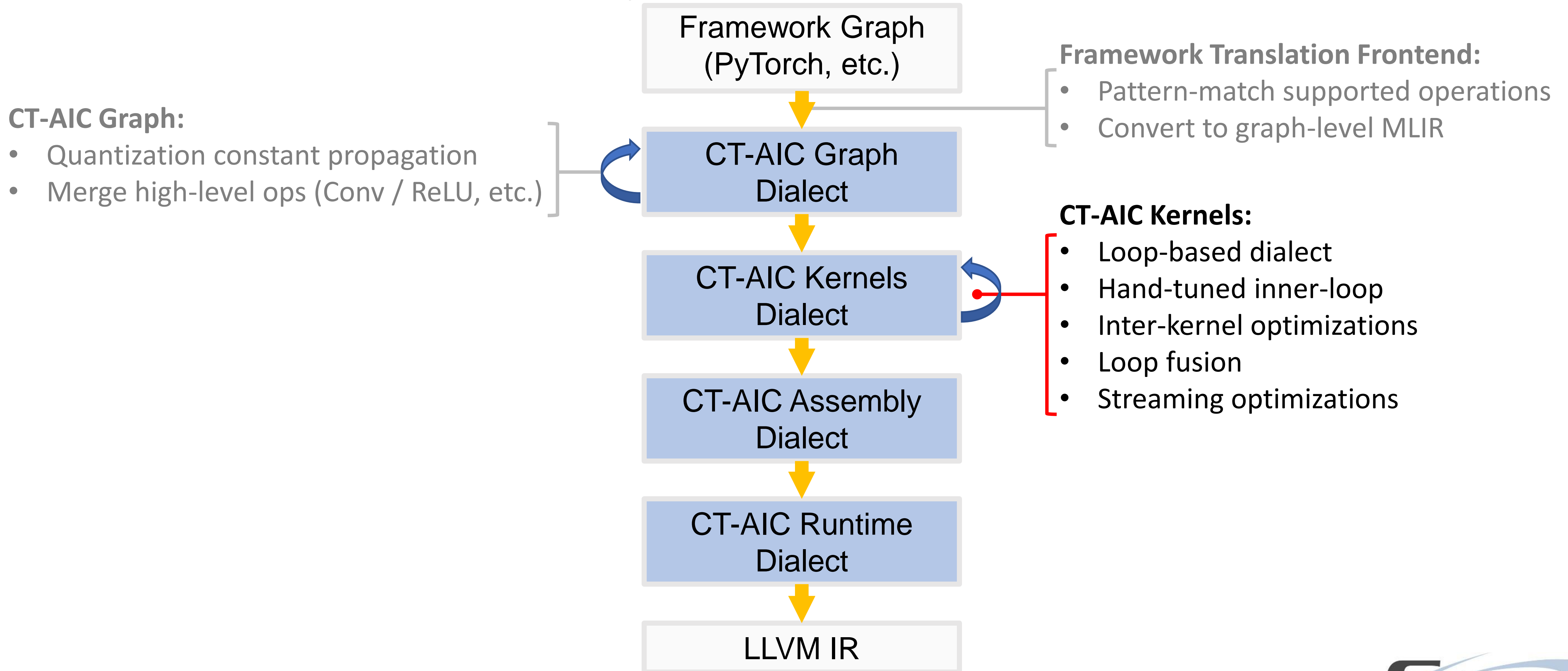
Framework Translation Frontend:

- Pattern-match supported operations
- Convert to graph-level MLIR

©2020 Centaur Technology. All Rights Reserved

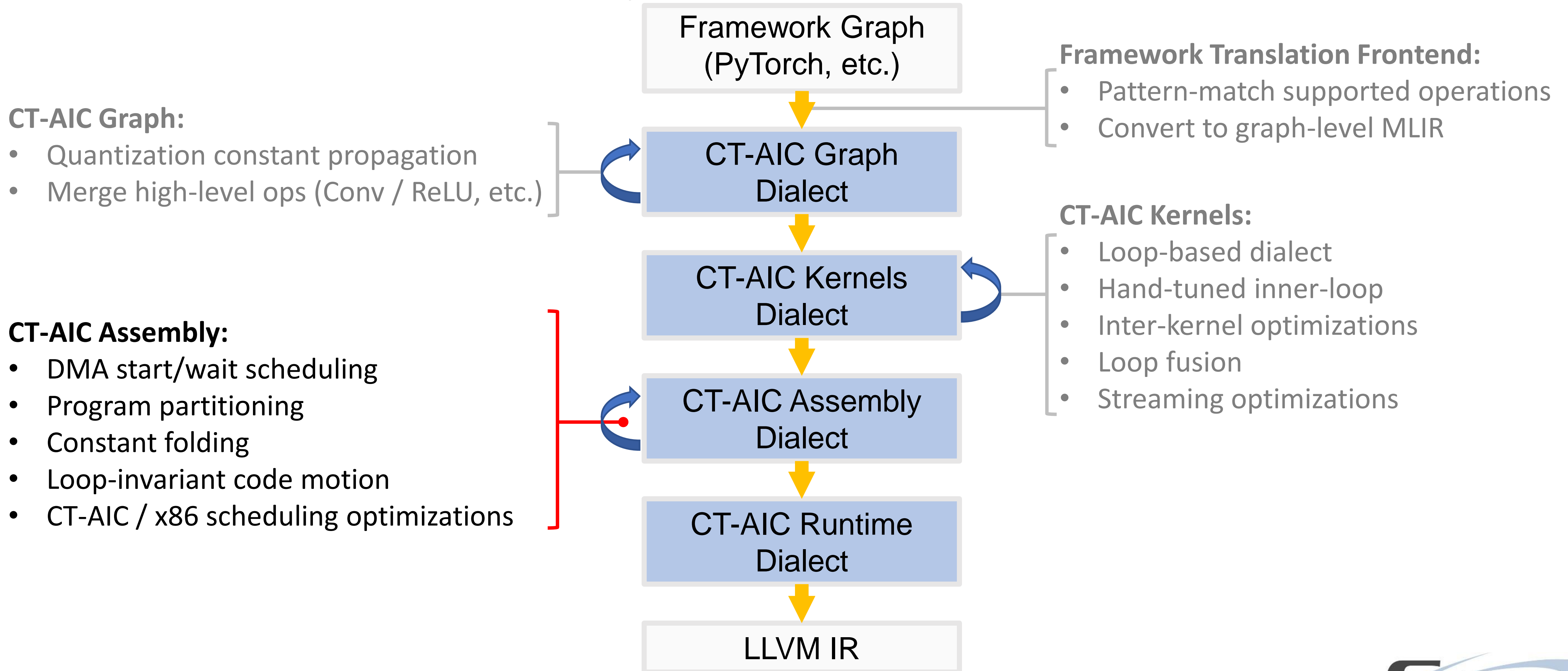
CenTaur Technology

Centaur AIC Compiler Stack: MLIR



©2020 Centaur Technology. All Rights Reserved

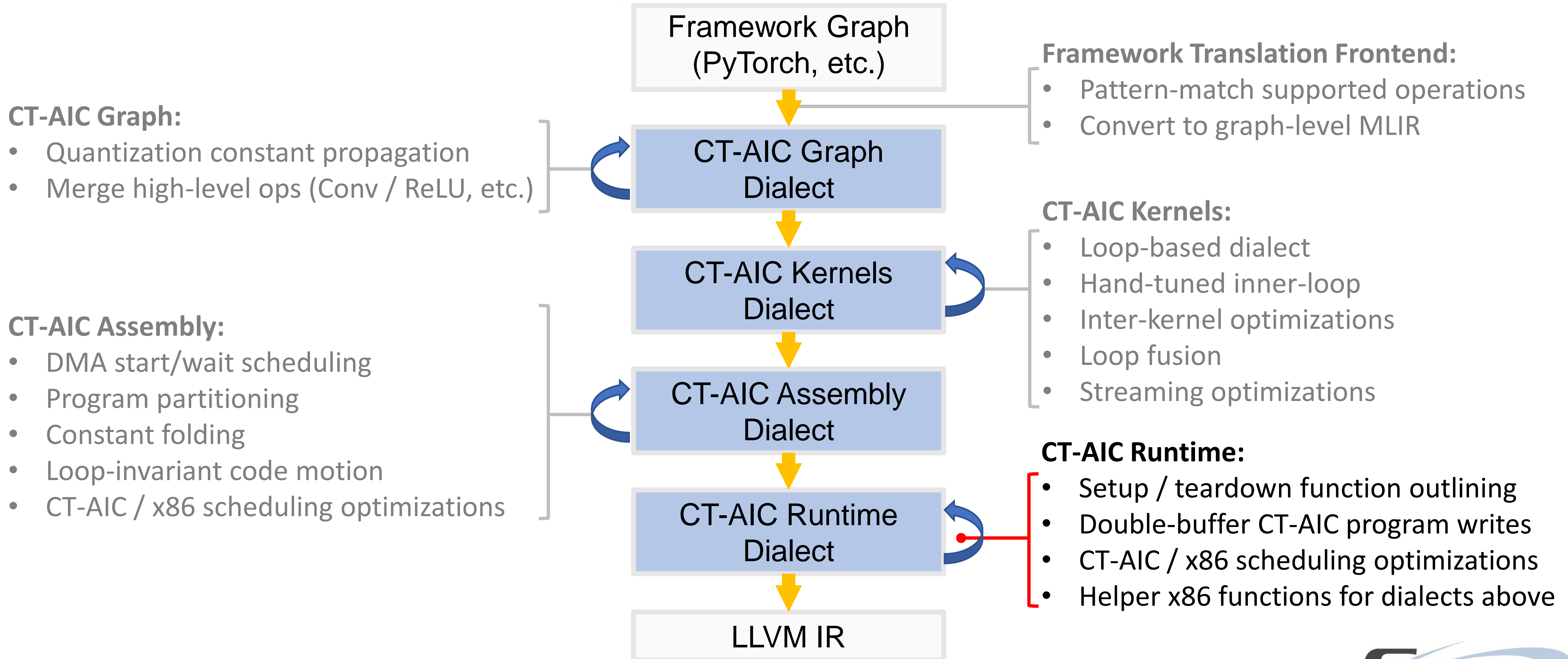
Centaur AIC Compiler Stack: MLIR



©2020 Centaur Technology. All Rights Reserved

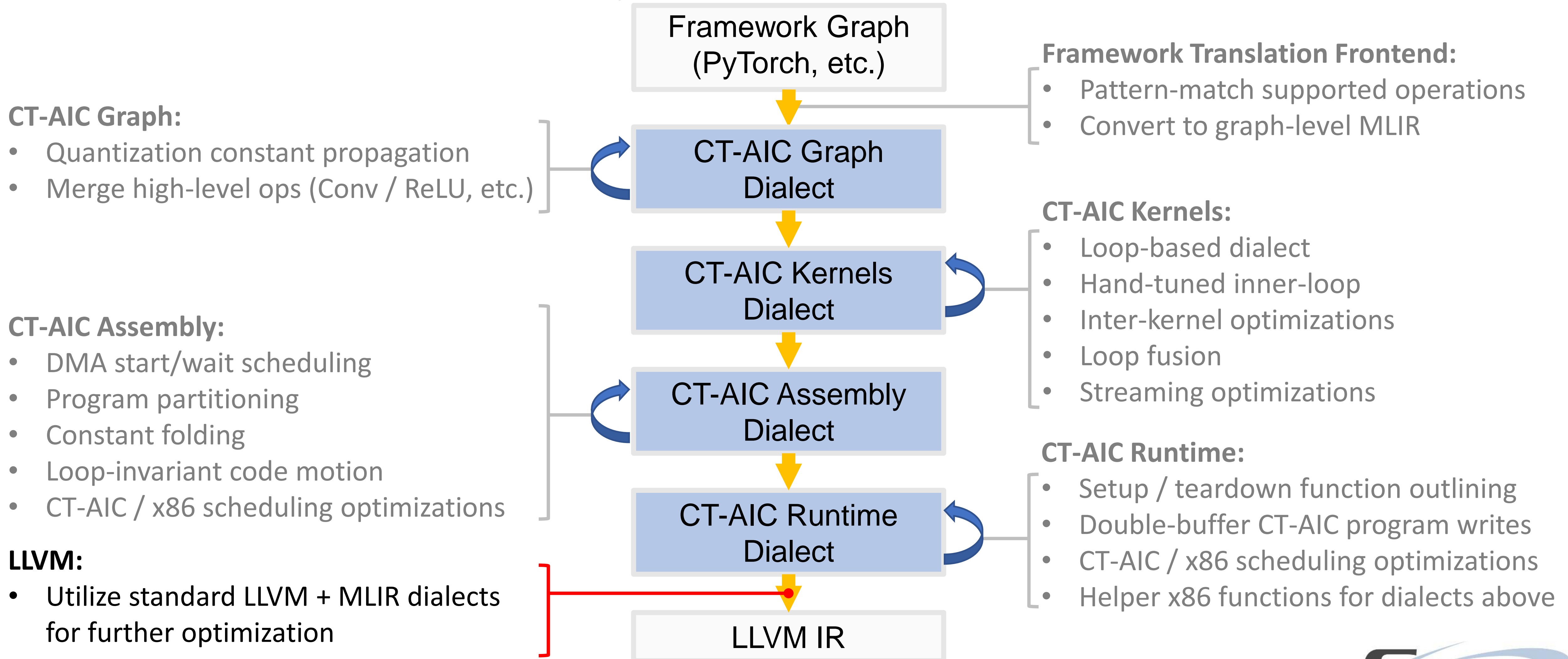
CenTaur Technology

Centaur AIC Compiler Stack: MLIR



©2020 Centaur Technology. All Rights Reserved

Centaur AIC Compiler Stack: MLIR



©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Example: Supporting Data Streaming

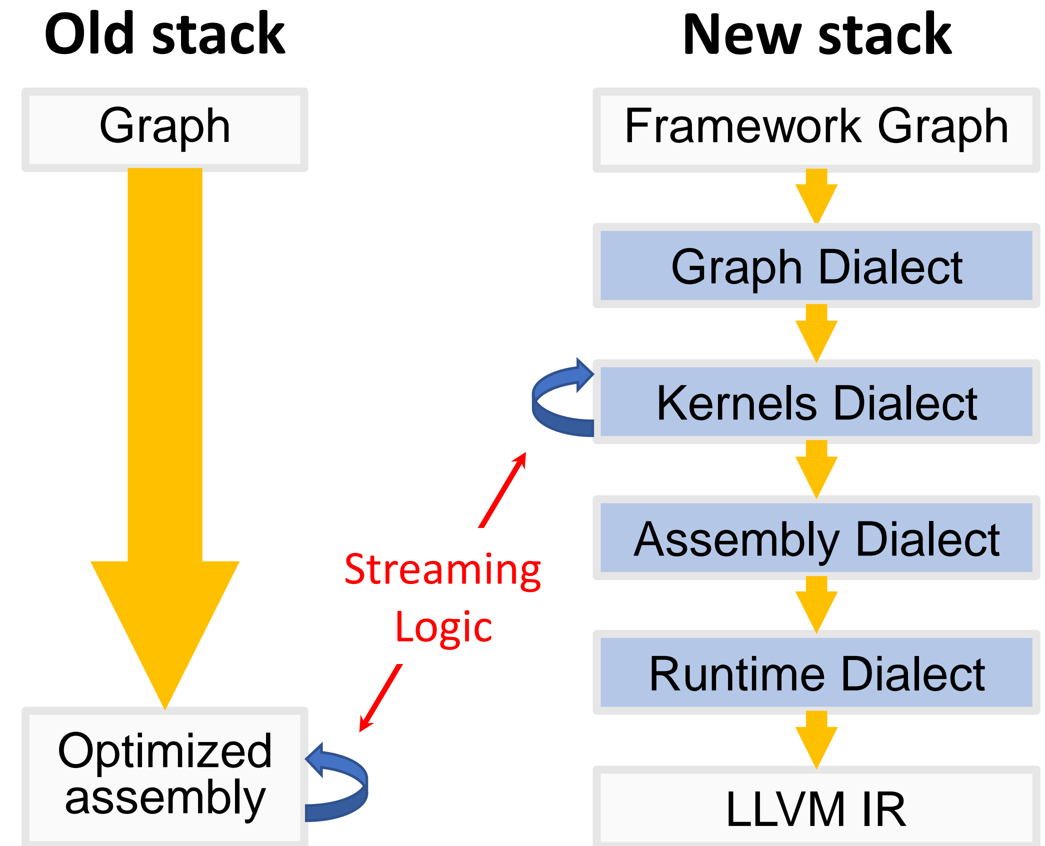
Intermediate kernel data sometimes larger than accelerator's internal RAMs

Old stack: Difficult & Time-consuming

- Add logic during assembly generation
- Complex, unique logic per kernel
- Weeks to debug assembly!

New stack: Easy & Rapid

- Simple pass added to existing dialect
- Utilizes common, reusable MLIR interface
- Only days to implement & debug!
- Can be lowered to any target (x86, AIC, etc.)



©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Breaking the Host/Device Barrier

Simplifies software development for heterogeneous systems

- Selectively promote x86 to AIC *inline*
- Example: Standard MLIR & AIC x86
  - Leave as x86, or
  - Promote to DMA in AIC execution

Standard MLIR x86 control

CT-AIC runtime x86 data movement

CT-AIC execution

```
%0 = alloc() {alignment = 64 : i64} : memref<8x4096xi8>
%1 = alloc() {alignment = 64 : i64} : memref<1x8x8x4xi8>
%2 = alloc() {alignment = 64 : i64} : memref<1x8x8x4xi8>
%3 = "aic_runtime.get_dram"() : () -> memref<2048x4096xi8>
%4 = "aic_runtime.get_wtram"() : () -> memref<2048x4096xi8>
%5 = alloc() {alignment = 64 : i64} : memref<8x4096xi8>
"aic_runtime.pack_add"(%5, %2) {block_layout = {block_layout_block_
%6 = "aic_kernels.allocate_dram"() {size = 2 : index} : () -> memre
%7 = alloc() {alignment = 64 : i64} : memref<8x4096xi8>
"aic_runtime.pack_add"(%7, %1) {block_layout = {block_layout_block_
%8 = "aic_kernels.allocate_dram"() {size = 2 : index} : () -> memre
%9 = "aic_kernels.allocate_dram"() {size = 2 : index} : () -> memre

affine.for %arg0 = 0 to 8 {
  %15 = affine.apply #map0(%arg0)
  %17 = affine.apply #map0(%arg0)
  %18 = affine.apply #map0(%arg0)
  %c1_12 = constant 1 : index
  "aic_runtime.write_ram"(%5, %15, %6, %15, %c1_12) : (memref<8x4096
  %c1_13 = constant 1 : index
  "aic_runtime.write_ram"(%7, %17, %8, %17, %c1_13) : (memref<8x4096
  "aic_kernels.add_inner_nkl"(%6, %15, %8, %17, %9, %18) {aicAddData
  %c1_14 = constant 1 : index
  "aic_runtime.read_ram"(%9, %18, %0, %18, %c1_14) : (memref<2x4096
}

%10 = alloc() {alignment = 64 : i64} : memref<8x4096xi8>
%11 = alloc() {alignment = 64 : i64} : memref<1x8x8x4xi8>
"aic_runtime.unpack_add"(%0, %11) {block_layout = {block_layout_blo
```

(Attributes truncated for space)

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Conclusions

Centaur's AIC + x86 SoC

- High performance
- Ubiquitous platform
- Across the board speedup in MLPerf v0.7

New ML Compiler Stack via MLIR

- Multiple levels of Centaur AIC dialects & optimizations
- Modular, expandable, reusable
- Quickening support for new models and frameworks
- Heterogeneous compilation in one place: It's the future!

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology

Thank you!

©2020 Centaur Technology. All Rights Reserved

CenTaur Technology