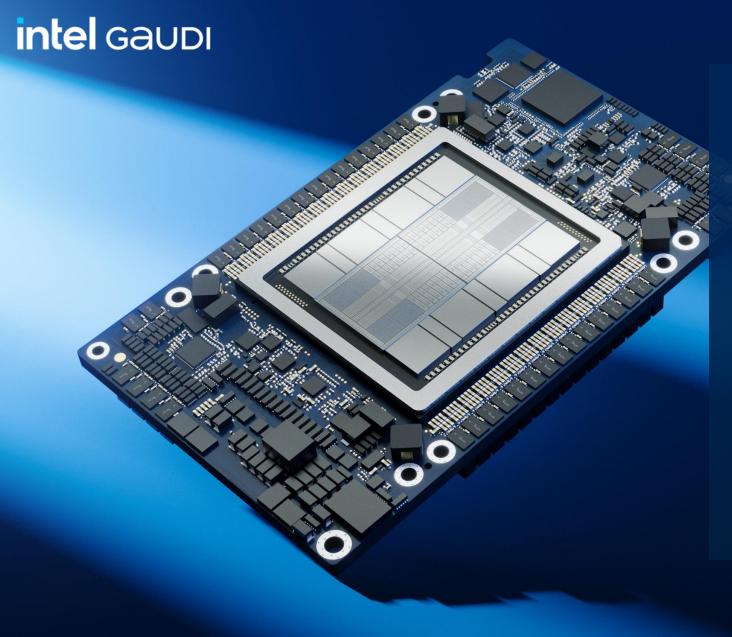
Core C++ 2024

Speeding Up Intel Gaudi Deep-Learning Accelerators Using an MLIR-Based Compiler

Dafna Mordechai, Omer Paparo Bivas



Speeding up Intel® Gaudi® deeplearning accelerators using an MLIR-based compiler

Dafna Mordechai, Omer Paparo Bivas, Jayaram Bobba, Sergei Grechanik, Tzachi Cohen, Dibyendu Das

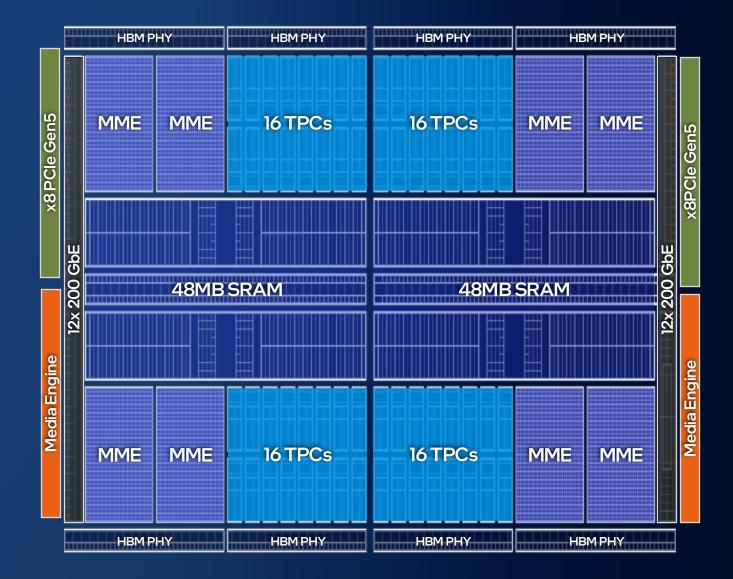
Intel/Habana Labs



- Deep Learning Compilers: Transforming a Large Computational Graph
 - into an optimized Execution Plan
- The TPC Fuser: A JIT compiler for deep learning kernels that delivers
 - significant performance improvements
- Case Study: Adjusting the TPC-Fuser to LLMs Recent Challenges

Intel Gaudi 3 Al accelerator Spec and Block Diagram

Feature/Product	Intel® Gaudi® 3 Accelerator
BF16 Matrix TFLOPs	1835
FP8 Matrix TFLOPs	1835
BF16 Vector TFLOPs	28.7
MME Units	8
TPC Units	64
HBM Capacity	128 GB
HBM Bandwidth	3.67 TB/s
On-die SRAM Capacity	96 MB
On-die SRAM Bandwidth RD+WR (L2 Cache)	19.2 TB/s
Networking	1200 GB/s bidirectional
Host Interface	PCle Gen5 x16
Host Interface Peak BW	128 GB/s bidirectional
Media Engine	Rotator + 14 Decoders (HEVC, H.264, JPEG, VP9)



MME - Matrix Multiplication Engine

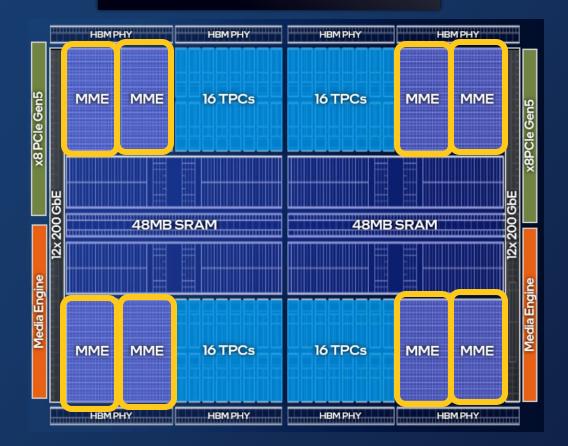
Configurable, not programmable

Each MME is a large output stationary systolic array

- 256x256 MAC structure w/ FP32 accumulators
- 64k MACs/cycle for BF16 and FP8

Large systolic array reduces intra-chip data movement, increasing efficiency Internal pipeline to maximize compute throughput





intel.

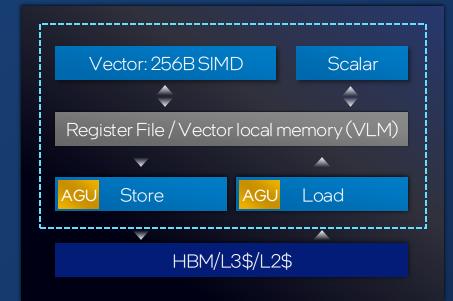
Tensor Processor Core (TPC): 256B-wide SIMD Vector Processor

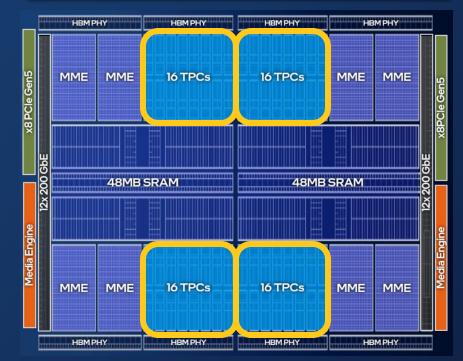
Fully Programmable using the TPC-C C enhanced with TPC intrinsics

VLIW with 4 separate pipeline slots: Vector, Scalar, Load & Store

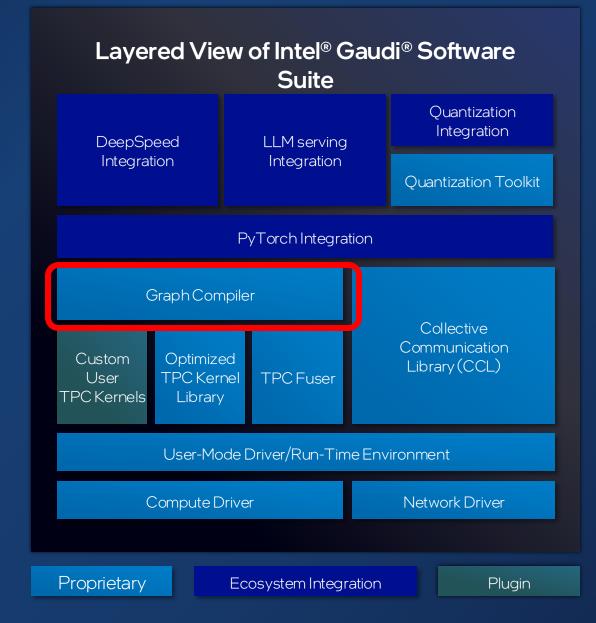
Integrated Address Generation Unit for HWaccelerated address generation

Supports main 1/2/4-Byte datatypes: Floating Point and Integer





Intel Gaudi Software Suite



intel. 7

Compilers 101

Compiler(Source Code) --> Machine Code

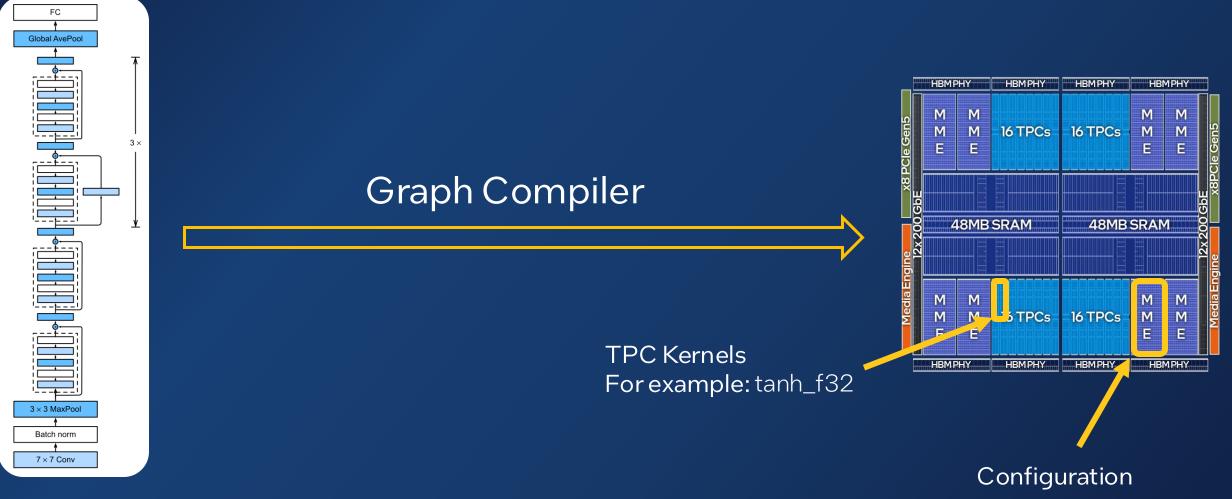
[semantic preserving] Compiler == (Translations + Transformations)

COMPILER Add • More •	Templates
C++ source #1 🖉	□ x86-64 gcc 14.2 (Editor #1) 🖉 ×
A - 🖬 + - V ,≋ 🖈	x86-64 gcc 14.2 🔹 🗹 🥝 Compiler opti
🤕 C++ 🔍 ▽	A 🔹 🏟 Output 👻 🟹 Filter 🍷 🔚 Libraries 🛛 🔑 Overrides
1 // Type your code here, or lo	1 square(int):
<pre>2 int square(int num) {</pre>	2 push rbp
3 return num * num;	3 mov rbp, rsp
4 }	4 mov DWORD PTR [rbp-4], edi
	5 mov eax, DWORD PTR [rbp-4]
	6 imul eax, eax
	7 pop rbp 8 ret

LLVM - Low Level Virtual Machine

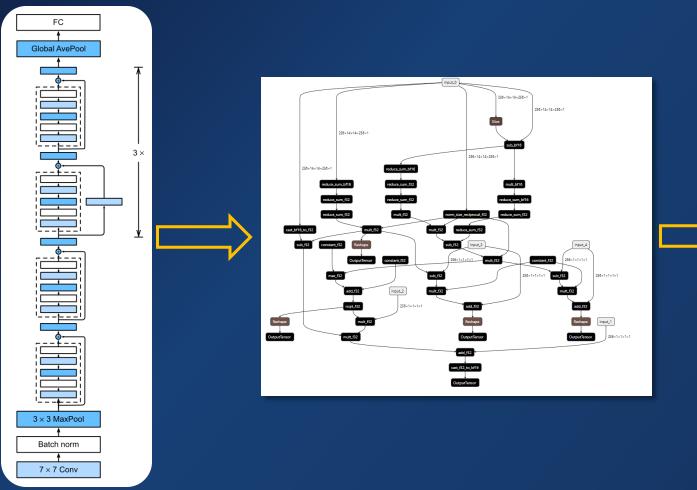
Clang - Front-end for languages in the C language family (C, C++, Objective C/C++, OpenCL, and CUDA)

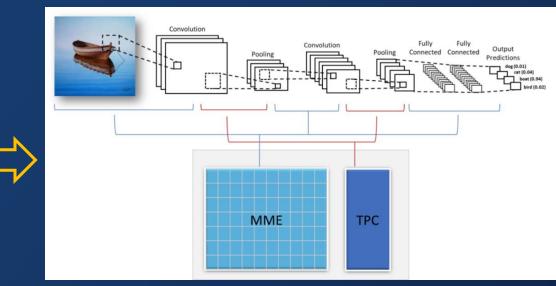
Deep Learning Compilation 101



The original Resnet-18 architecture. Up to 152 layers were trained in the original publication (as "ResNet-152") - Wikipedia

Graph Compilation Flow: Transforming a Deep Learning Computational Graph into an Intel-Gaudi Execution Plan





Neural Network Hardware Mapping – Use of MME and TPC

The original Resnet-18 architecture. Up to 152 layers were trained in the original publication (as "ResNet-152") - Wikipedia

TPC Kernel Library Providers

Pre-Compiled Library (TPC-C)

Intel-Gaudi optimized TPC kernel library

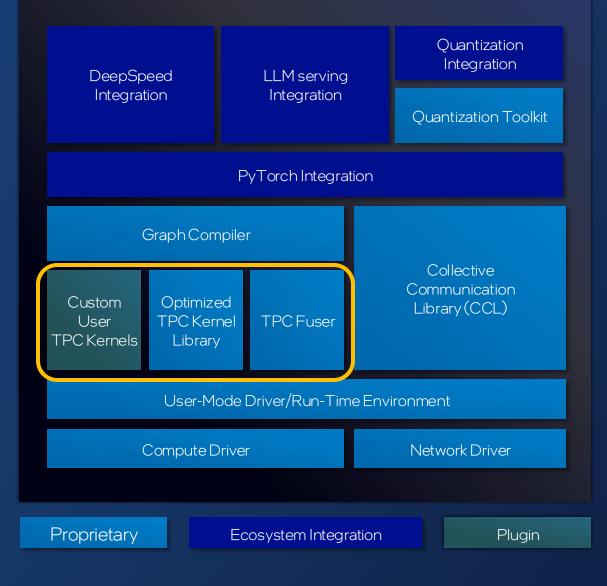
Custom user kernels

JIT Library

Auto-generated fused kernels, compiled during graph compilation, using the MLIR-based JIT compiler

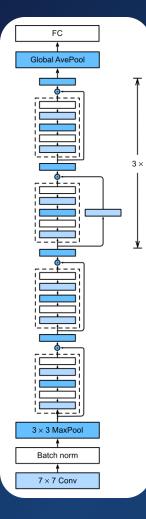
All the kernels are compiled using the Clang-based TPC Compiler

Layered View of Intel® Gaudi® Software Suite



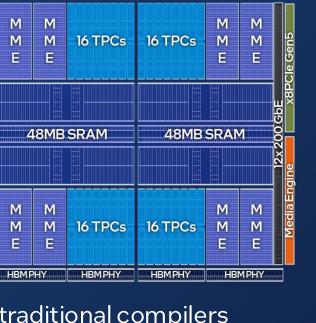
11

Graph Compilation Flow



- **Processes** the deep learning topology to allocate operations across MME, TPC, and DMA engines
 - Generate MME Configurations
 - Select kernels from the different kernel library providers
- Optimizes the computational graph using techniques similar to traditional compilers
 - Schedules operations while accounting for memory constraints and dependencies
 - **Configures** hardware registers and system settings based on the execution plane (recipe)

The original Resnet-18 architecture. Up to 152 layers were trained in the original publication (as "ResNet-152") - Wikipedia



HBMPHY

HBMPHY

HBMPHY

HBM PHY

What's an Ideal Execution?

*MME (accel0)					
^[D0] MME	bert/e bert/ bert/e ber	t/ bert/e bert/ bert/e bert/.	/ bert/e bert/ bert/e bert/ bert/	/e bert/e bert/ bert/e bert/ bert/e bert/ bert/e	bert/e bert/ bert/e bert/e bert/e bert/ brt/e
MMEs	bert/e bert/e bert/e ber	rt/ bert/e bert/ bert/e bert/e	e bert/ bert/e bert/e bert/e	t/ bert/e bert/ bert/e bert/e bert/e bert/e	bert/e bert/ bert/ bert/e bert/ bert/ bert/
	bert/e bert/e bert/e ber	rt/ bert/e bert/ bert/e bert/.	/ bert/e bert/ bert/e bert/e bert	t/ bert/e bert/ bert/e bert/e bert/e bert/.	bert/e bert/ bert/ bert/e bert/ bert/ bert/
^[D3] MME	bert/e bert/ bert/e ber	rt/ bert/e bert/ bert/e bert/e	e bert/ bert/e bert/ bert/e bert	t/ bert/e bert/ bert/e bert/e bert/ bert/e bert/.	bert/e bert/ bert/e bert/e bert/ bert/ bert/
✓ *PDMA (accel0)					
▲ *TPC (accel0)					
^[D0] TPC 0 fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP fusedT
^[D0] TPC 1 fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTP		_bundle	fusedTP fusedTP fusedT
^[D0] TPC 2 fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTP())CE	eration e	execution	fusedTP fusedTP fusedT
[D0] TPC 3	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_07_Junale		earrewaae_o_o7_banale	fusedT fused fusedT
fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP fusedT
fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP fusedT
[D1] TPC 0 fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fuse	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP
[D1] TPC 1 fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fuse	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP fusedT
[D1] TPC 2	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedT
[D1] TPC 3	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedT fusedT
^[D1] TPC 4 fusedTPCNode_0_87_bundle	e fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle	fusedTPCNode_0_87_bundle fus	edTPCNode_0_87_bundle fusedTPCNode_0_87_bundle	fusedTP fusedTP fusedT

Graph Compiler: Slicing + Bundling

The Graph Compiler is designed to partition data and group operations efficiently to achieve overlapping execution between the MME and TPC units.

This optimization maximizes the use of SRAM and local caches, enhancing data transfer efficiency.

▲ *MME (accel0)																											
▲[D0] MME		bert/e	bert/ be	rt/e be	t/ bert/e	bert/	bert/e	bert/	bert/e	bert/ be	rt/e be	rt/ bert/e	bert/e	. bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/e	bert/	bert/e	bert/	bert/e	bert/e	bert/	bert/e
▲[D1] MME		bert/e	bert/e be	rt/e be	t/ bert/e	bert/	bert/e	bert/e	bert/	bert/ be	ert/e be	rt/e bert/	bert/e	. bert/	bert/e	bert/	bert/e	bert/e	bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/
▲[D2] MME		bert/e	bert/e be	rt/e be	t/ bert/e	bert/	bert/e	bert/	bert/e	bert/ be	rt/e be	rt/e bert/	bert/e	. bert/	bert/e	bert/	bert/e	bert/e	bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/
▲[D3] MME		bert/e	bert/ be	rt/e be	t/ bert/e	bert/	bert/e	bert/e	bert/ b	ert/e b	ert/ be	rt/e bert/	bert/e	. bert/	bert/e	bert/e	bert/	bert/e	bert/	bert/e	bert/	bert/e	bert/e	bert/e	bert/	bert/e	bert/
✓ *PDMA (accel0)																											
▲ *TPC (accel0)		Cache/SI	RAM		Cache/	SRAM															n.						
▲[D0] TPC 0	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b		fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	.0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fus	edT				
▲[D0] TPC 1	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fus	edT				
▲[D0] TPC 2	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fuse	edT				
▲[D0] TPC 3	fusedTPCNode_0_87_bundle	fusedTPCN	lode_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedT	fused	fusedT					
▲[D0] TPC 4	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fus	edT				
▲[D0] TPC 5	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fus	edT				
▲[D1] TPC 0	fusedTPCNode_0_87_bundle	fusedTPCN	lode_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	e	fusedTF	P fused	dTP fus	edT				
▲[D1] TPC 1	fusedTPCNode_0_87_bundle	fusedTPCN	Node_0_87_b	undle	fused	PCNode_0	_87_bundle	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	e	fusedTF	P fused	dTP fus	edT				
▲[D1] TPC 2	fusedTPCNode_0_87_bundle	fusedTPCN	ode_0_87_bu	ndle	fused	PCNode_0	_87_bundl	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fuse	edT				
▲[D1] TPC 3	Slice 1	S	lice 2		fused	PCNode_0	_87_bundl	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedT	fusedT.	fusedT					
▲[D1] TPC 4				nuic	fused	PCNode_0	_87_bundl	e	fusedTPC	Node_0_87	_bundle	fuse	dTPCNode_	0_87_bun	dle	fusedT	PCNode_0	_87_bundl	le	fusedTF	P fused	dTP fus	edT				

JIT Compiler: Key Benefits and Advantages

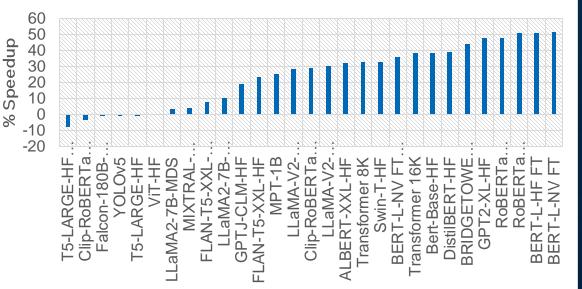
- Spare HBM bandwidth
- Enhances caches and local memory efficiency
- Spare kernel-to-kernel invocation latency
- Applies shape-based optimizations



The TPC Fuser Supports element-wise operations, reductions and normalizations

TPC-Fuser Performance Improvements

End-to-end model execution



1.3X Avg Perf Improvement at model level

1.5X Avg Perf Improvement in device execution time

4 4 5 7 3 3 3 3

Model Trace Number

61 65

2

73

Device execution times

160

140

120

100 80

60

40

20

0

 $\mathbf{\sigma}$

ပာတက

-20

Speedup

%

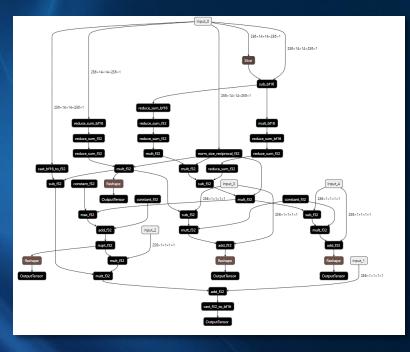
Measured on Intel Gaudi2

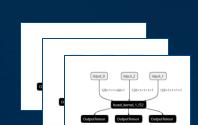
88 93 97

81 85

How Is It Done?

Clustering





Codegen

+

Let's Take an Example:

∨ module {

syn_rt.graph(%in0: !syn_rt.tensor<4x2x128xf32>, %in1: !syn_rt.tensor<4x2x128xf32>) -> !syn_rt.tensor<4x2x128xf32>{
 %out0 = tpckernel.add %in0, %in1 : (!syn_rt.tensor<4x2x128xf32>, !syn_rt.tensor<4x2x128xf32>) -> <f32> -> (!syn_rt.tensor<4x2x128xf32>)
 %out0_0 = tpckernel.sin %out0 : (!syn_rt.tensor<4x2x128xf32>) -> <f32> -> (!syn_rt.tensor<4x2x128xf32>)
 syn_rt.exit %out0_0 : !syn_rt.tensor<4x2x128xf32>)

module {

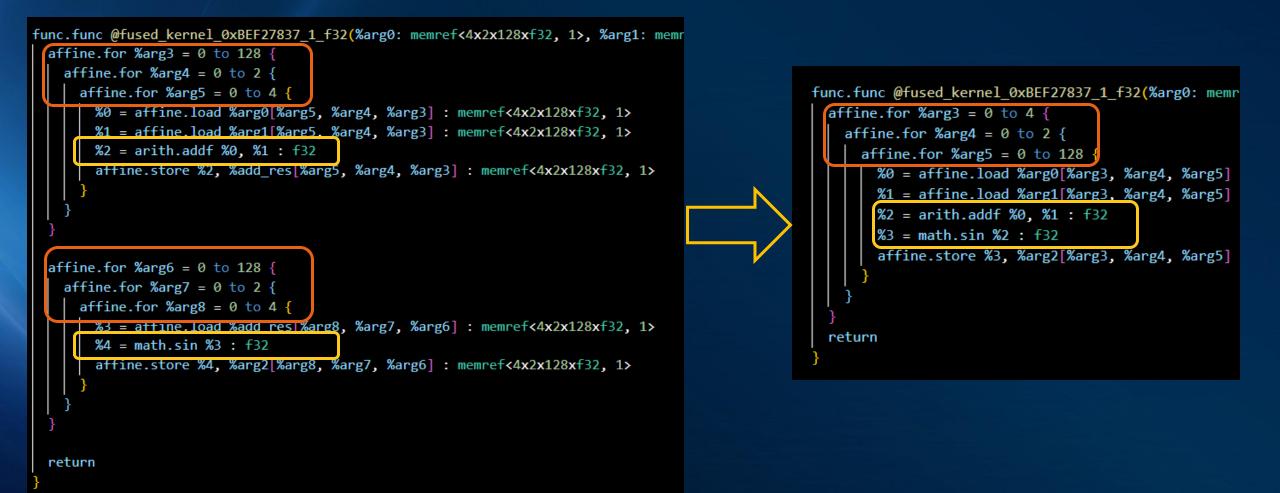
```
func.func @fused_kernel_0xBEF27837_1_f32(%arg0: tensor<4x2x128xf32>, %arg1: tensor<4x2x128xf32>) -> tensor<4x2x128xf32> attributes {syn.expected_arch = "
    %0 = syn.add %arg0, %arg1 : tensor<4x2x128xf32>
    %1 = syn.sin %0 : tensor<4x2x128xf32>
    return %1 : tensor<4x2x128xf32>
    }
    syn_rt.graph(%in0: !syn_rt.tensor<4x2x128xf32>, %in1: !syn_rt.tensor<4x2x128xf32>) -> !syn_rt.tensor<4x2x128xf32>{
        %0 = syn rt.launch tpc func @fused kernel 0xBEF27837 1 f32(%in0, %in1) : (!syn rt.tensor<4x2x128xf32>, !syn rt.tensor<4x2x128xf32>) -> !syn rt.tensor<4x2x128xf32>)
```

```
syn_rt.exit %0 : !syn_rt.tensor<4x2x128xf32>
```

Loop Fusion

The compiler merges multiple loops with the same iteration range into a single loop

Loop Fusion

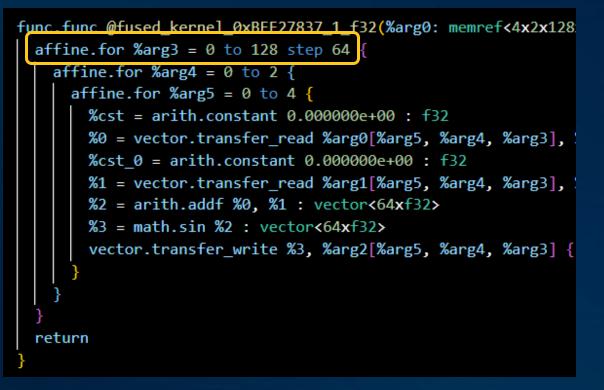


Vectorization

The compiler converts independent loop iterations into SIMD instructions, based on the specific hardware vector width

Vectorization

```
func.func @fused kernel 0xBEF27837_1_f32(%arg0: memr
affine.for %arg3 = 0 to 128 {
    affine.for %arg4 = 0 to 2 {
        affine.for %arg5 = 0 to 4 {
            %0 = affine.load %arg0[%arg5, %arg4, %arg3]
            %1 = affine.load %arg1[%arg5, %arg4, %arg3]
            %2 = arith.addf %0, %1 : f32
            %3 = math.sin %2 : f32
            affine.store %3, %arg2[%arg5, %arg4, %arg3]
        }
    }
    return
}
```

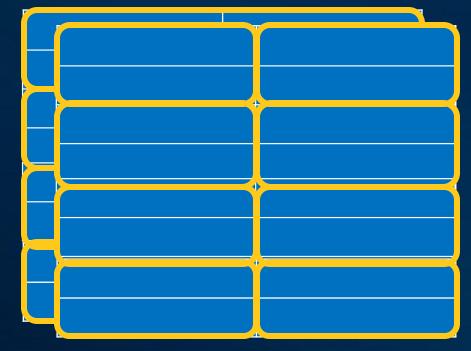


Loop Unroll

The compiler combines operations from consecutive iterations and merges them into a single iteration

-		

unroll_factor=2



Loop Unroll

```
func.func @fused_kernel_0xBEF27837_1_f32(%arg0: memref-
affine.for %arg3 = 0 to 4 {
    affine.for %arg4 = 0 to 2 {
        affine.for %arg5 = 0 to 128 {
            %0 = affine.load %arg0[%arg3, %arg4, %arg5] : 1
            %1 = affine.load %arg1[%arg3, %arg4, %arg5] : 1
            %2 = arith.addf %0, %1 : f32
            %3 = math.sin %2 : f32
            affine.store %3, %arg2[%arg3, %arg4, %arg5] : 1
        }
    }
}
```

return

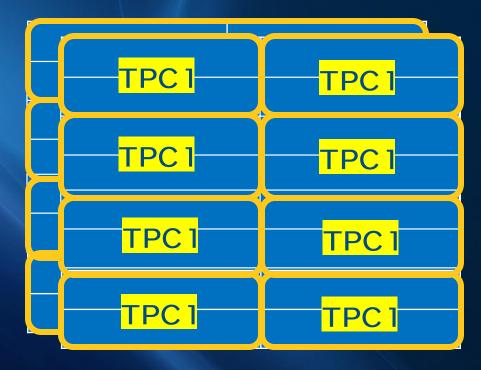
unroll_factor= 4 "fully unroll" one of the loops

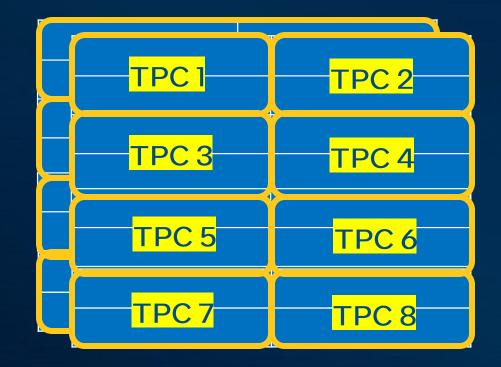
func.func @fused kernel 0xBEF27837 1 f32(%arg0: memref<4x2x2 %c0 = arith.constant 0 : index affine.for %arg3 = 0 to 128 step 64 { affine.for %arg4 = 0 to 2 { %0 = affine.apply affine map<(d0) -> (d0 + 3)>(%c0)%1 = affine.apply affine map<(d0) -> (d0 + 2)>(%c0)%2 = affine.apply affine map<(d0) -> (d0 + 1)>(%c0)%3 = affine.vector load %arg0[%c0, %arg4, %arg3] : mer %4 = affine.vector load %arg0[%2, %arg4, %arg3] : memu %5 = affine.vector load %arg0[%1, %arg4, %arg3] : mem %6 = affine.vector load %arg0[%0, %arg4, %arg3] : mem %7 = affine.vector load %arg1[%c0, %arg4, %arg3] : mer %8 = affine.vector load %arg1[%2, %arg4, %arg3] : memm %9 = affine.vector load %arg1[%1, %arg4, %arg3] : memm %10 = affine.vector load %arg1[%0, %arg4, %arg3] : mer %11 = arith.addf %3, %7 : vector<64xf32> %12 = arith.addf %4, %8 : vector<64xf32> %13 = arith.addf %5, %9 : vector<64xf32> %14 = arith.addf %6, %10 : vector<64xf32> %15 = math.sin %11 : vector<64xf32> %16 = math.sin %12 : vector<64xf32> %17 = math.sin %13 : vector<64xf32> %18 = math.sin %14 : vector<64xf32> affine.vector store %15, %arg2[%c0, %arg4, %arg3] : me affine.vector store %16, %arg2[%2, %arg4, %arg3] : mer affine.vector_store %17, %arg2[%1, %arg4, %arg3] : mer affine.vector store %18, %arg2[%0, %arg4, %arg3] : mer

return

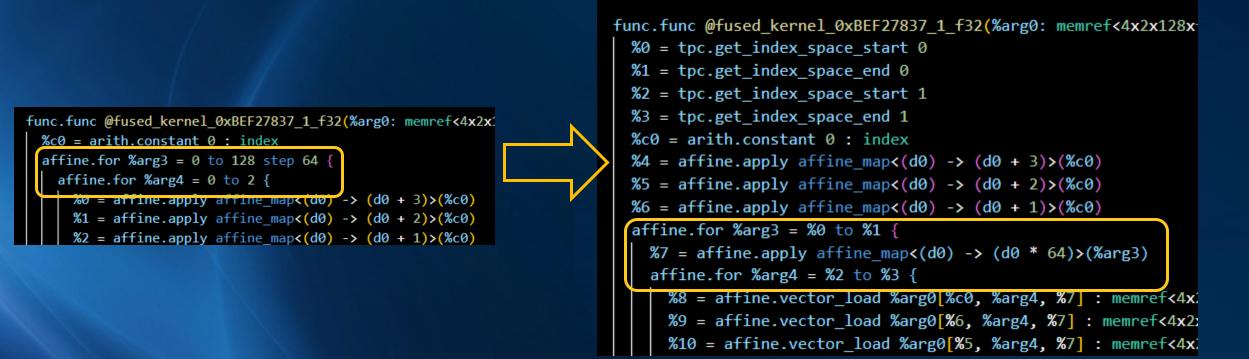
Parallelization

The compiler transforms the constant bounds of a loop into variables, enabling scalable parallel execution across multiple processing units





Parallelization



Case Study: Adjusting the TPC-Fuser to LLMs Recent Challenges

Large Language Models and Triangular Softmax

• Make predictions about the next word, for each prefix of the sentence.

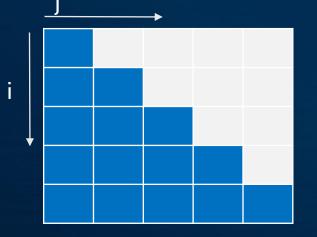
										kon	Pr/	hah	sility
										oko			bility
The										Tok			
The											kon		aability bability
The	goal										oken		obability
The	goal	of									Tokon Tokon		Probability Probability
The	goal	of	the								Token		Probability
The	goal	of	the	model							Token		Probability
The	goal	of	the	model	is						word		0.73
The	goal	of	the	model	is	to					senter	nce	0.12
The	goal	of	the	model	is	to	predict			L	level		0.07
The	goal	of	the	model	is	to	predict	the			time		0.05
The	goal	of	the	model	is	to	predict	the	next		step		0.0229
											banan	а	0.0001

Traditional vs. Triangular Access to the Data

• Supporting a new type of data access created challenges.

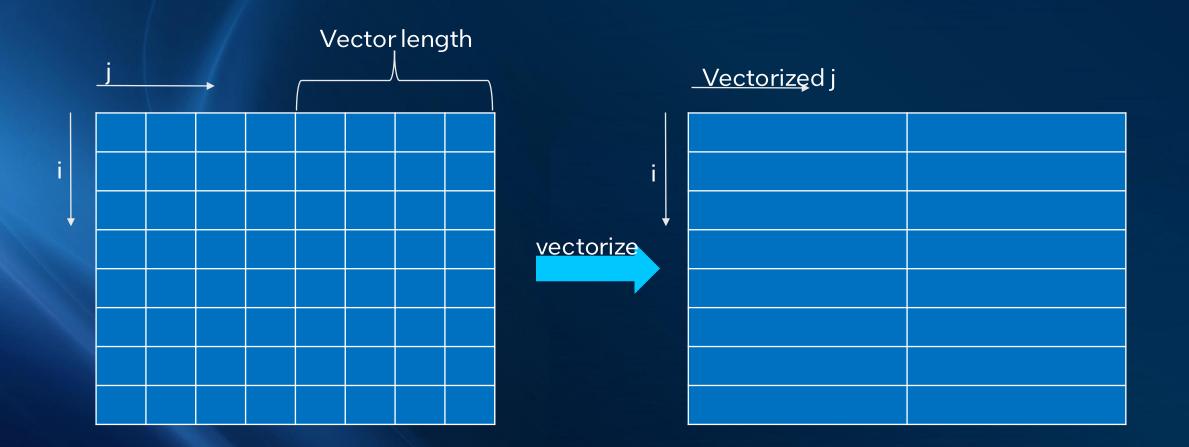
```
Traditional access
for (i = 0; i < N; ++i) {</pre>
  for (j = 0; j < M; ++j) {
    // Access A[i][j]
```

Triangular access



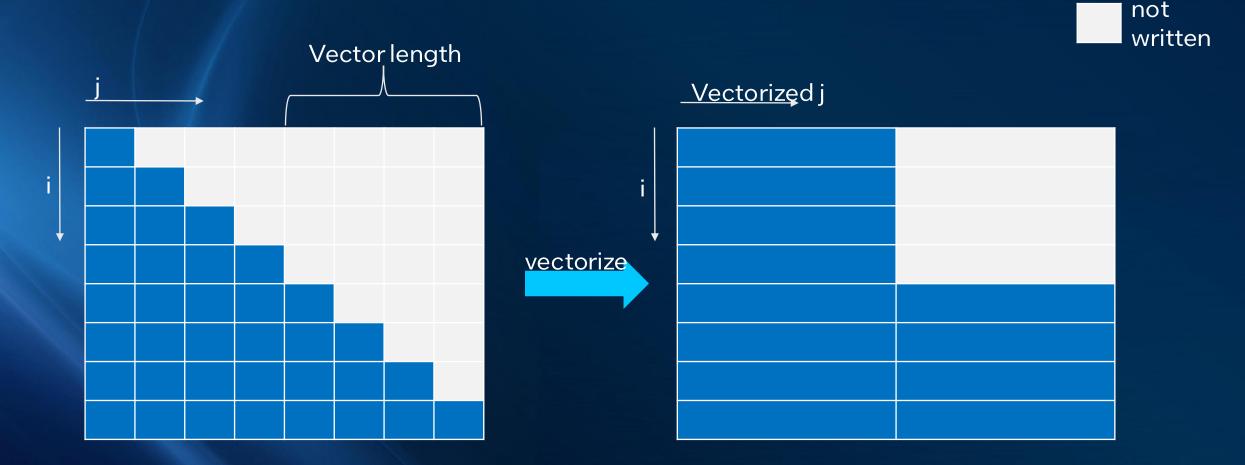
Vectorizing triangular loops

• Traditional vectorization, for rectangular (regular) tensor access:



Vectorizing triangular loops

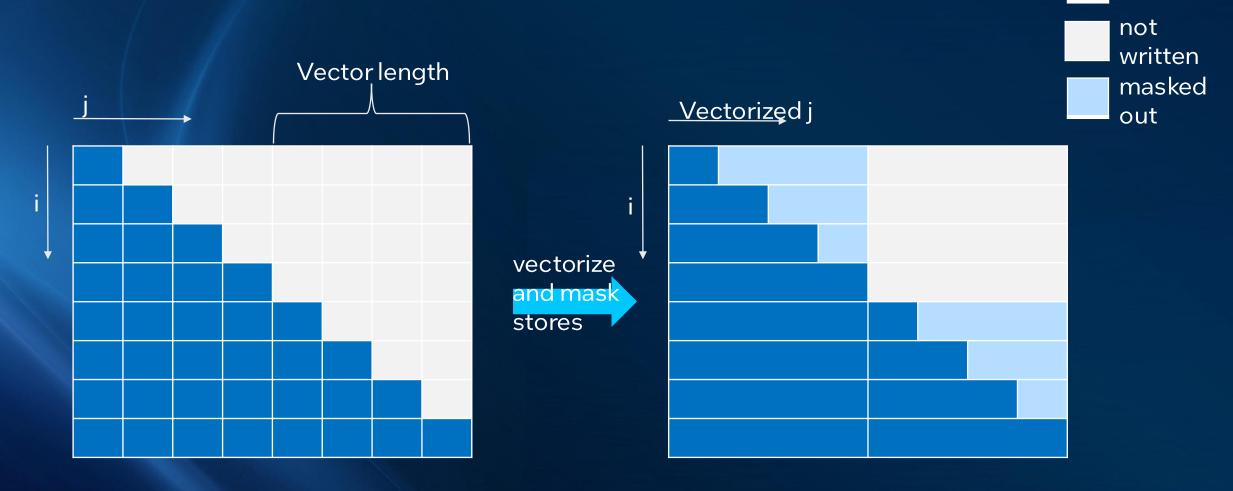
• **Problem:** Vectorizing triangular loops will result in a vector-level granularity triangle.



written

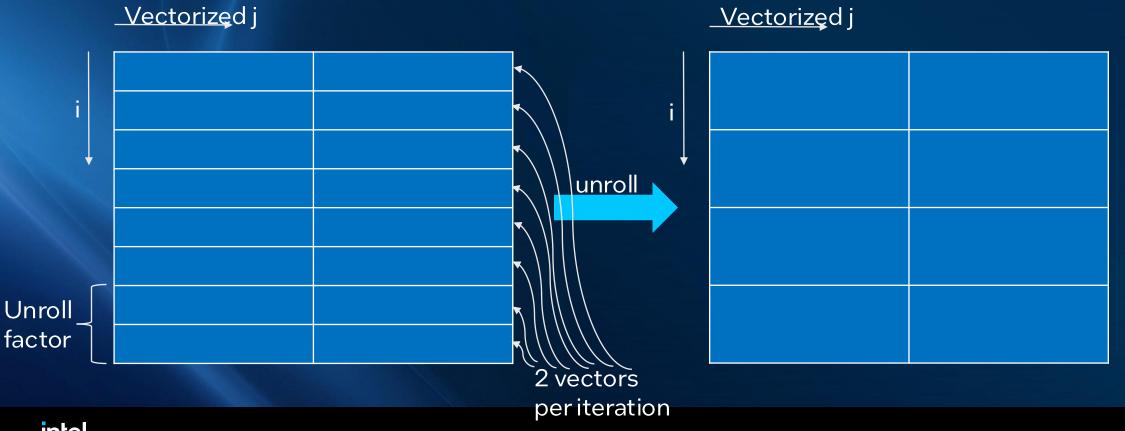
Vectorizing triangular loops

• Solution: Need to add masks to each vector write:

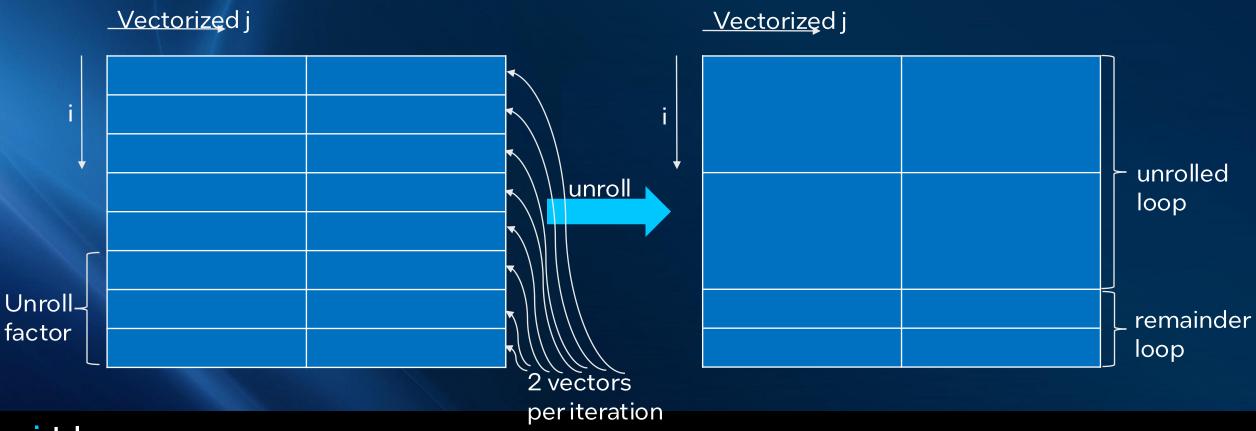


written

• Traditional unrolling, for rectangular (regular) tensor access with unroll factor = 2:

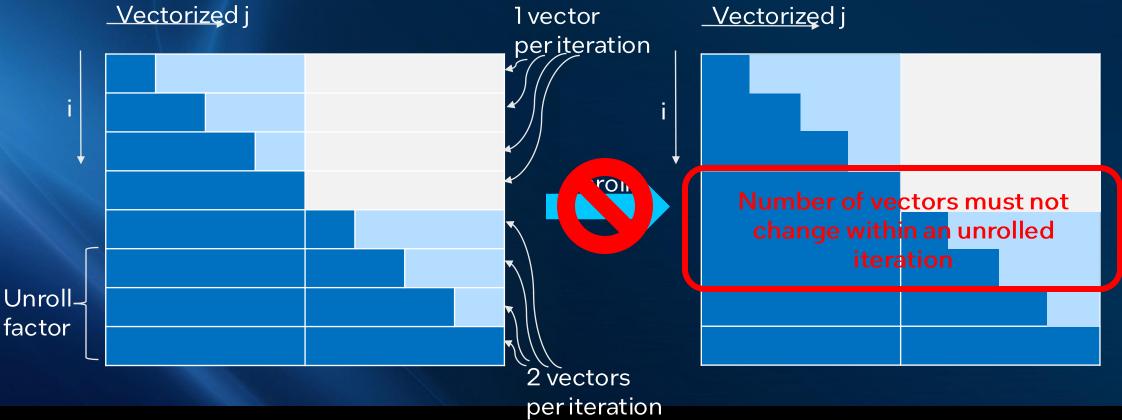


• Traditional unrolling, for rectangular (regular) tensor access with unroll factor = 3:

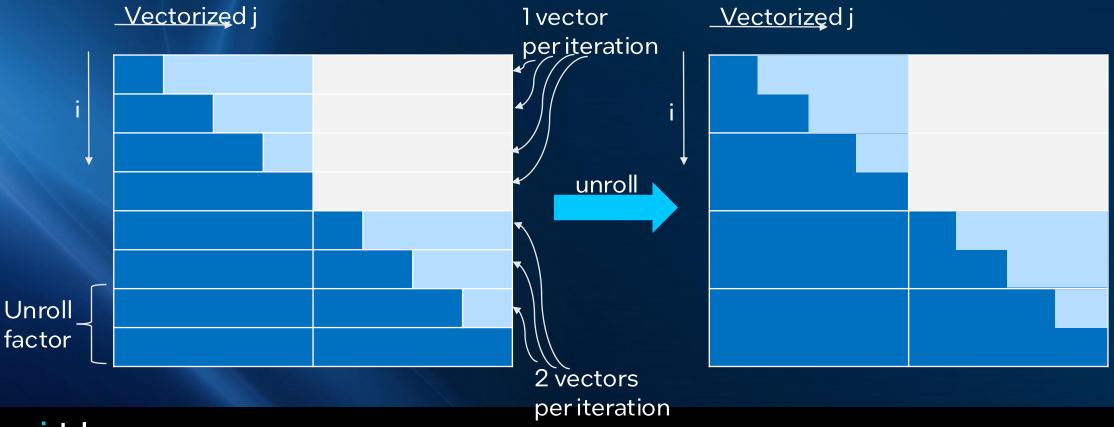


intel

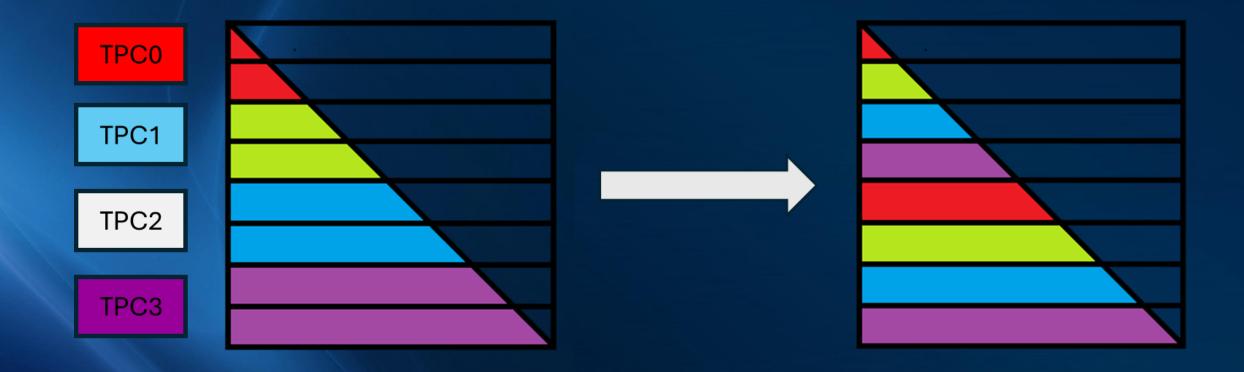
- **Problem:** Some unroll factors may result in operating on different number of vectors
- Unrolling triangular tensor access with unroll factor = 3 is forbidden:



- Solution: Allow only legal unroll factors:
 vector_size % unroll_factor == 0
- Unrolling triangular tensor access with unroll factor = 2 is allowed:



Parallelizing triangular loops



Parallelizing Triangular Loops

Performance with naïve parallelization – $219.6\mu s$

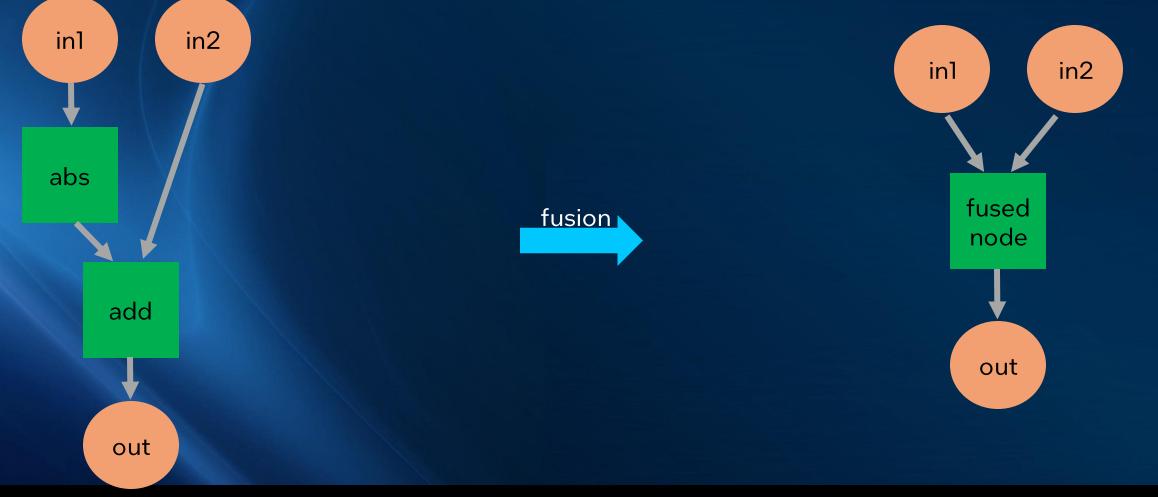
1008.5 s +	31,274 ms	31,274.02 ms	31,274.04 ms	31,274.06 ms	31,274.08 ms	31,274.1 ms	31,274.12 ms	31,274,14 ms	31,274.16 ms	31,274,18 ms	31,274.2 ms	31,274.22 ms
×												
 *TPC (accel0) 												
▲[D0] TPC 0	fusedTPCN fu	tufusedTPCNode_0.	-	fusedTPCNode_	fusedTPC	Node_	fusedTPCNo	de	tu tu tu	M		
▲[D0] TPC 1	fusedTPCNo f	us fus fusedTPCNod	e_0_0_bu	fusedTPCNode_0_0_t	fusedTPC	CNode_0_0_bu	fusedTPCNo	de_0_0_bu	fu., fus., fus	sM		
▲[D0] TPC 2	fusedTPCNod_	fus fus fusedTPCM	lode_0_0_bundle_1/	fusedTPCNode_0_0_	bundle_1/ fusedTP	CNode_0_0_bundle_1/_	fusedTPCN	ode_0_0_bundle_1/	fus fus f	íus 🙀		
▲[D0] TPC 3	fusedTPCNode_	fuse fuse fus	edTPCNode_0_0_bundle	_1/op_1 fusedT	PCNode_0_0_bundle_1/	op_3 fusedTPC	Node_0_0_bundle_1/op_	5 fusedTPCNor	le_0_0_bundle_1/op_7	fuse fuse fu	se_	
▲[D0] TPC 4	fusedTPCNode_	fusedfusedfused	IPCNode	fusedTPCNode	fusedTPC	Node	fusedTPCNo	de	fused fused	i fusedM		
▲[D0] TPC 5	fusedTPCNode_0) fusedT fusedT fu	sedTPCNode_0_0_bu	fusedTPCNode_0_0_t	fusedTPC	CNode_0_0_bu_	fusedTPCNo	de_0_0_bu	fusedTfuse	dT fusedT🙀		
▲[D1] TPC 0	fusedTPCNode_	. fusedTP fusedTP	fusedTPCNode_0_0_bu	ndle_1/ fusedTPCNo	ode_0_0_bundle_1/f	fusedTPCNode_0_0_bur	ndle_1/ fusedTPCNo	ode_0_0_bundle_1/	fusedTP fuse	edTPfusedTP		
▲[D1] TPC 1	fusedTPCNode_	fusedTPC fusedTP	fusedTPCNode_	0_0_bundle_1/op_1	fusedTPCNode_0_0	_bundle_1/op_3	fusedTPCNode_0_0_bu	ndle_1/op_5 fu	sedTPCNode_0_0_bund	dle_1/op_7 fusedTP	C fusedTPC fusedTPC	
▲[D1] TPC 2	fusedTPCNode_0	fusedTPC fusedTF	C fusedTPCNode	fusedTPCNode	fusedTPC	Node	fusedTPCNo	de	fusedTPCfu	usedTPC fusedTPC	M	
▲[D1] TPC 3	fusedTPCNode_	0_0 fusedTPCN fused	TPCNfusedTPCNode	e_0_0_bu_ fusedTPCNo	ode_0_0_bu fusedTPC	Node_0_0_bu	fusedTPCNo	de_0_0_bu	fusedTPCNf	fusedTPCN fusedTPC	NM	
▲[D1] TPC 4	fusedTPCNode_0	0_0 fusedTPCNo fus	edTPCNofusedTPCN	Node_0_0_bundle_1/_	fusedTPCNode_0_0_bu	ndle_1/ fusedTPCN	lode_0_0_bundle_1/	fusedTPCNode_0_0_bu	ndle_1/ fusedTPCNc	fusedTPCNo fuse	dTPCNo_	
▲[D1] TPC 5	fusedTPCNode	fusedTfuseT_fuseT_fuseT_fuseTfuseTfuseTfuseTfuseTfuseTfuseTfuseT	usedTPCNode_0_0_bun	dle_1/op_1 fuse	dTPCNode_0_0_bundle	_1/op_3 fusedT	PCNode_0_0_bundle_1/o	p_5 fusedTPCI	Node_0_0_bundle_1/op	_7 fusedT fused	f fusedT	
▲[D2] TPC 0	fusedTPC fu	fu fusedTPCNode		fusedTPCNode_	fusedTPC	Node_	fusedTPCNo	de	fu., fu., fu.,	M.		
^[D2] TPC 1	fusedTPCNo fi	fus fusedTPCNode	_0_0_bu	fusedTPCNode_0_0_t	fusedTPC	Node_0_0_bu	fusedTPCNo	de_0_0_bu	fus fu fus	s		

Parallelizing Triangular Loops

Performance with triangular-adapted parallelization – 170.8µs. 22% speedup!

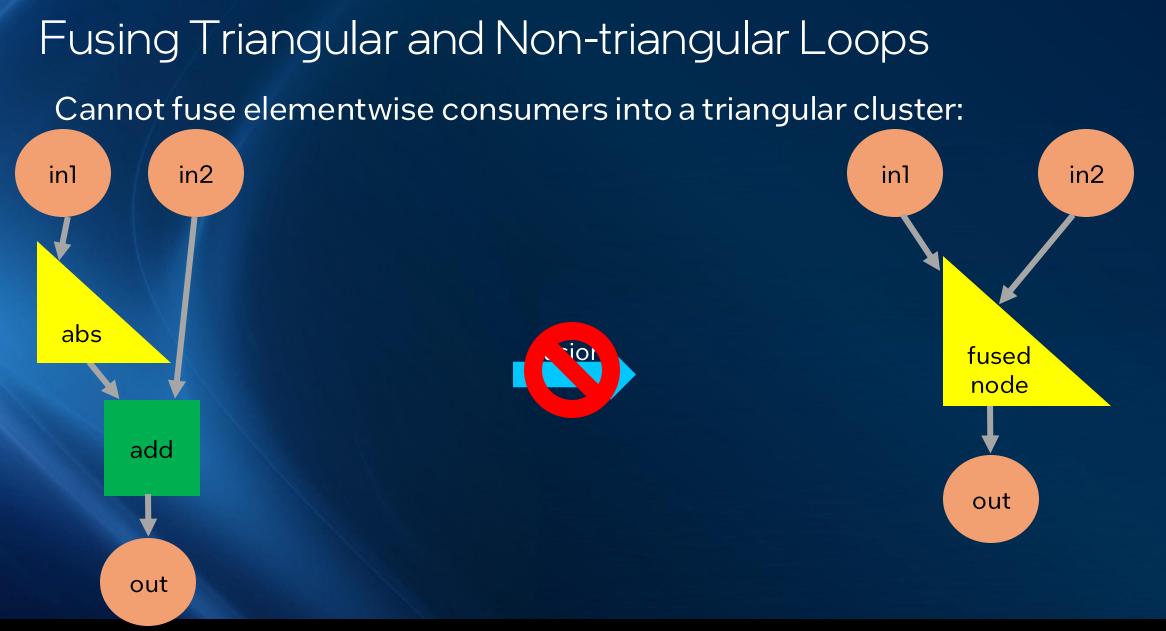
4678 s +	î	29,547.9 ms	29,547.92 ms	29,547.94 ms	29,547.96 ms	29,547.98 ms	29,548 ms	29,548.02 ms	29,548.04 ms	29,548.06 ms
X					170.0					
 *TPC (accel0) 										
▲[D1] TPC 4	fusedTf	PCNode_0_ fusedTP_ fus	edTPfusedTPCNode_0_0	_bundle_1/o_ fusedTPC	Node_0_0_bundle_1/o fu	sedTPCNode_0_0_bundle_1	/opfusedTPCNode_0_0_	bundle_1/o fusedTP_ 1	fusedTPfusedTP	
▲[D1] TPC 5	fusedTi	PCNode_0 fusedTP fus	edTPfusedTPCNode_0_0	_bundle_1 fusedTPCNoc	de_0_0_bundle_1 fusedT	PCNode_0_0_bundle_1	usedTPCNode_0_0_bundle_	1 fusedTP fusedTP	fusedTP	
▲[D2] TPC 0	fusedTi	PCNode_0_0 fusedTPC 1	fusedTPCfusedTPCNode	_0_0_bundle_1/o fused	<pre>FPCNode_0_0_bundle_1/o</pre>	fusedTPCNode_0_0_bun	dle_1/o fusedTPCNode_0	0_0_bundle_1/o fusedTP	C fusedTPC fusedTP	M
▲[D2] TPC 1	fusedTi	PCNode_0_0 fusedTPC 1	fusedTPC fusedTPCNode	_0_0_bundle_1/op fused	TPCNode_0_0_bundle_1/op	fusedTPCNode_0_0_bu	ndle_1/op fusedTPCNod	le_0_0_bundle_1/op fus	edTP fusedTPC fusedTF	PC
▲[D2] TPC 2	fusedTl	PCNode_0_0 fusedTPC	fusedTP fusedTPCNode	_0_0_bundle_1/o fused	TPCNode_0_0_bundle_1/o	fusedTPCNode_0_0_bun	dle_1/o fusedTPCNode_	0_0_bundle_1/o fusedTP	C fusedTPC fusedTPC	
^[D2] TPC 3	fusedTI	PCNode_0_0 fusedTPC	fusedTPfusedTPCNode	_0_0_bundle_1 fusedTPO	CNode_0_0_bundle_1_ fus	sedTPCNode_0_0_bundle_1.	fusedTPCNode_0_0_bun	dle_1 fusedTPC fused	TP fusedTP	
^[D2] TPC 4	fusedT	PCNode_0_0 fusedTPC	fusedTPCfusedTPCNoc	de_0_0_bundle_1/o_ fuse	dTPCNode_0_0_bundle_1/o	fusedTPCNode_0_0_bu	ndle_1/o_ fusedTPCNode	_0_0_bundle_1/o_ fused	IPC fusedTPC fusedTP	C
▲[D2] TPC 5	fusedT	PCNode_0_0 fusedTPC	fusedTPC fusedTPCNod	le_0_0_bundle_1/op fuse	edTPCNode_0_0_bundle_1/o	ppfusedTPCNode_0_0_1	oundle_1/op fusedTPCN	ode_0_0_bundle_1/opfu	sedTPC fusedTPC fused	атрс
^[D3] TPC 0	fusedTi	PCNode_0 fusedTPC fu	sedTPCfusedTPCNode_	0_0_bundle_1/o_ fusedT	PCNode_0_0_bundle_1/o_	fusedTPCNode_0_0_bund	e_1/o fusedTPCNode_0	_0_bundle_1/o_ fusedTP0	C fusedTPC fusedTPC	M
^[D3] TPC 1	fusedTi	PCNode_0 fusedTPC fu	sedTPCfusedTPCNode_l	0_0_bundle_1 fusedTPC	Node_0_0_bundle_1_ fuse	dTPCNode_0_0_bundle_1	fusedTPCNode_0_0_bund	e_1 fusedTPC fusedT	PC fusedTPC	
▲[D3] TPC 2	fusedTF	PCNode_0_ fusedTP_ fus	edTPfusedTPCNode_0_0	_bundle_1/o fusedTPC	Node_0_0_bundle_1/o fu	usedTPCNode_0_0_bundle_	1/ofusedTPCNode_0_0_	bundle_1/o_ fusedTP_ 1	fusedTP fusedTP	
^[D3] TPC 3	fusedTi	PCNode_0_ fusedTP_ fus	sedTP fusedTPCNode_0_0	_bundle_1/op fusedTPC	Node_0_0_bundle_1/op	fusedTPCNode_0_0_bundle	_1/opfusedTPCNode_0	_0_bundle_1/op fusedTP	. fusedTP fusedTP	
^[D3] TPC 4	fusedTF	PCNode_0_ fusedTP_ fus	edTPfusedTPCNode_0_0	_bundle_1/o_ fusedTPC	Node_0_0_bundle_1/of	usedTPCNode_0_0_bundle_	1/ofusedTPCNode_0_0	bundle_1/o fusedTP_ 1	fusedTPfusedTP	
^[D3] TPC 5	fusedT	PCNode_0 fusedTP fus	sedTPfusedTPCNode_0_0	_bundle_1 fusedTPCNoo	de_0_0_bundle_1 fusedT	PCNode_0_0_bundle_1	usedTPCNode_0_0_bundle_	1 fusedTP fusedTP	fusedTP_	

Fusing Triangular and Non-triangular Loops Traditional fusion, for rectangular (regular) tensor access:





intel



intel.

Triangular Data Access

The introduction of a new data access pattern introduced new challenges:

- Correctness challenges
- Performance challenges
- Operation fusion challenges



- The TPC Fuser is a JIT compiler for deep learning kernels
- It is deployed as part of <u>Gaudi Synapse SW stack</u>
- Delivers significant performance improvements
- Works in-tandem with a Graph Compiler to optimize execution

across the entire accelerator

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at <u>www.Intel.com/PerformanceIndex</u>.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel technologies may require enabled hardware, software or service activation.

Availability of accelerators varies depending on SKU. Please contact your Intel sales representative for more information.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.