Bolt: Bridging the Gap Between Auto-Tuners and Hardware-Native Performance

Jiarong Xing, Leyuan Wang, Shang Zhang, Jack Chen, Ang Chen, Yibo Zhu

R244: Large Scale Data Processing and Optimization
Pedro Sousa

Background

- Auto-tuners transform a tensor program into an equivalent but structurally different tensor for higher performance on a target hardware.
- Hardware-agnostic auto-tuners (e.g. Ansor) generate efficient tensor programs by searching a large search space.
- For certain workloads, they perform worst than hardware-native vendor tuned libraries (e.g. cuBLAS and cuDNN).
- E.g.: NVIDIA GPUs have special hardware architectures that Ansor cannot leverage since it uses an opaque hardware model.
- However, hardware libraries are not perfect either since they have fixed primitives and lack the customization of auto-tuners.

Solution

- **Opportunity:**
  - Emerging trends of modularized, templated libraries (e.g. NVIDIA CUTLASS).
  - Templates are parameterized and can be initiated to fit different hardware and workloads.
  - Templated libraries consider device details and extract hardware performance.

BOLT enables end-to-end optimizations that bridge the gap between auto-tuners and hardware-native performance.

BOLT Key Differentiators

1. Leverages templated libraries for joint graph, operator, and model level optimizations.
2. Enables novel persistent kernel fusion for deeper operator fusion at graph level.
3. Automates templated code generation via light-weight profiler and direct code generation.
4. System-friendly model design principles (e.g. exploring activations).

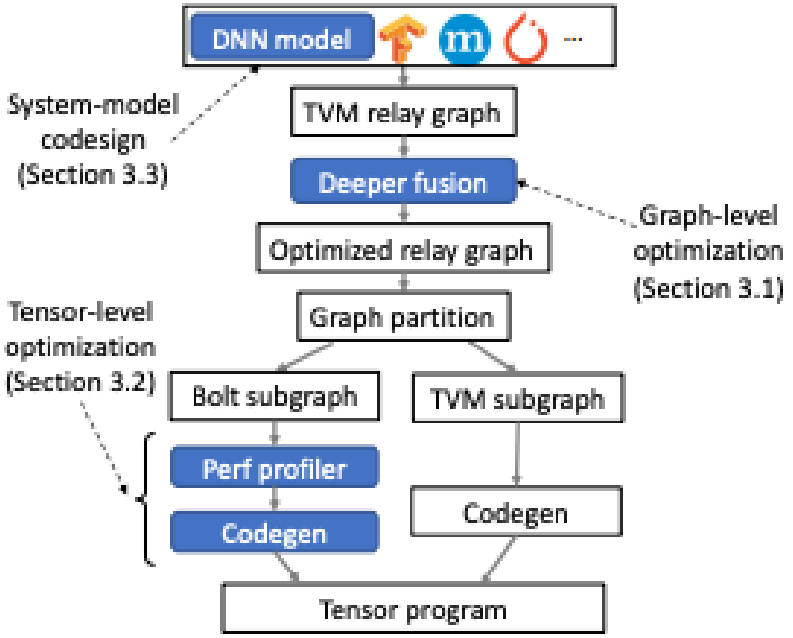


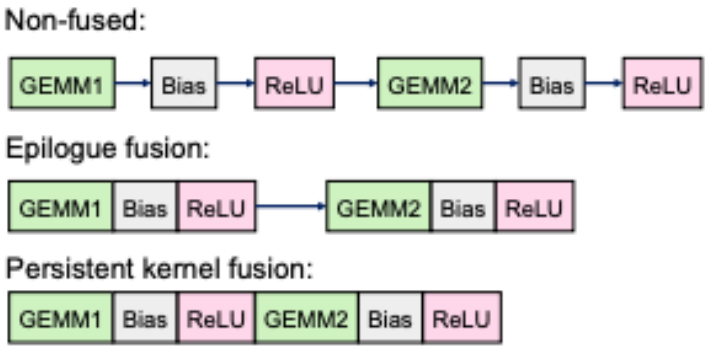
Figure 3. The workflow of Bolt. Blue boxes are our contributions.

Graph-level: deeper operator fusion 1/3

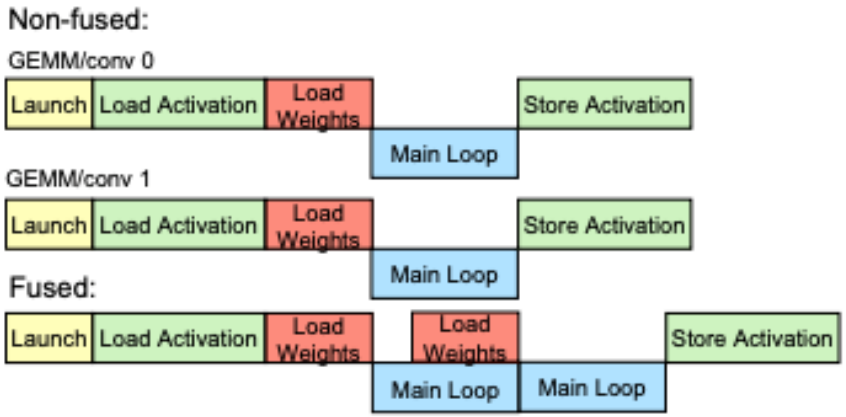
- New optimizations introduced to the device libraries via template customization.
- **Persistent kernel fusion:** computes multiple operators using only one kernel, leading to deeper operator fusion.
- Benefits of fusing *GEMMs* or *Convs* into a single operator:
  1. No memory traffic for storing and loading inter-layer activations.
  2. No launch latency.
  3. Larger optimization scope.

Graph-level: deeper operator fusion 2/3

- When 2 *GEMM*/*Conv* operations are fused together, the main computation loops run back-to-back in a single fused kernel.
- The output activation of the first *GEMM*/*Conv* stays in faster GPU memory.
- Bolt automatically detects opportunities to use persistent kernels in the computational graph and generates the CUDA code by creating new templates in CUDLASS.



(a) The graph view of persistent kernel fusion



(b) The kernel view of persistent kernel fusion

Graph-level: deeper operator fusion 3/3

Threadblock residence:

- Used when the threadblock size remains the same between operations.
- Each threadblock works independently on a tile of the output matrix/activation.

Register-file (RF) residence fusion:

- Used when the next operator's weights can fit entirely within a threadblock's warp tiles.
- Keeps the output activations from the previous op in the register file without writing to shared memory.

Shared-memory resident fusion:

- Used when next *GEMM*/*Conv* needs data from multiple warp tiles, writing the output activation tiles from RF to shared memory.

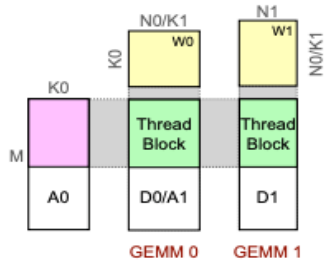


Figure 5. Illustration of threadblock-residence of GEMM fusion. Colored boxes represent one single threadblock. This requires ThreadBlock0_N = N0, ThreadBlock1_N = N1.

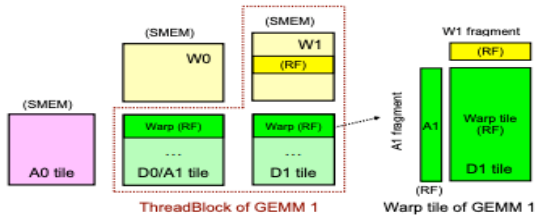


Figure 6. RF-resident fusion in a threadblock of back-to-back GEMMs. The threadblock and warp size requirements are: Warp0_N=ThreadBlock0_N=N0, Warp1_N=ThreadBlock1_N=N1.

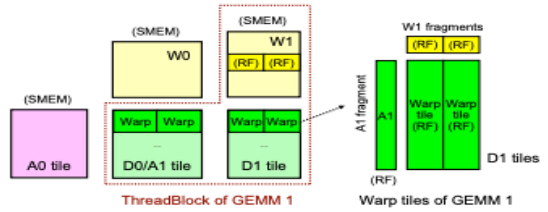


Figure 7. Shared memory-resident fusion in a threadblock of back-to-back GEMMs. The threadblock size requirements are: ThreadBlock0_N = N0 ≠ Warp0_N, ThreadBlock1_N = N1 ≠ Warp1_N.

Operator-level: automating templated code generation 1/2

- Templates only support a subset of operators, struggling to provide complete functionality for end-to-end models.
- Bolt uses **Bring Your Own Compiler**:
  - Reuse existing compiler stacks (e.g. TVM) and focus only on the optimization and generating code for the target device templates.
- **Lightweight Profiler**:
  - Searches template parameters space efficiently using hardware details.
  - Generates samples covering possible threadblocks/warp sizes, data types, etc.

Operator-level: automating templated code generation 2/2

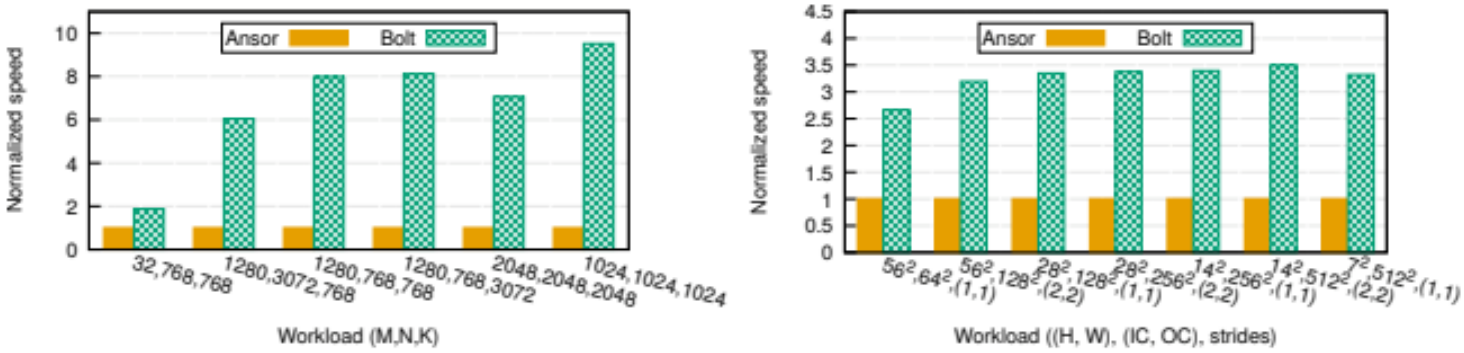
- Low-level tensor implementations by instantiating templates using best parameters from profiler.
- Generates CUDA code following template conventions instead of invoking blackbox functions.
- Extend templates to support new customized optimizations:
  - **Layout transforms**: change compute layout without graph edits.
  - **Padding for alignment**: pad unaligned tensors to use alignment 8, enabling tensor core acceleration and reducing memory loading time.

Model-level: designing system-friendly models

- *System-model codesign* helps build models that run more efficiently.
- Three main principles identified by Blt:
  1. Exploring different activation functions with epilogue fusion.
  2. Deepening models with 1x1 Convs.
  3. Aligning tensor shapes to use GPUs more efficiently.

Evaluating GEMM/Conv2D performance

Bolt achieves significantly higher performance when compared to Ansor, motivated by its tuning strategy based on hardware-native templates.



(a) GEMMs performance.

(b) Conv2D performance.

Figure 8. The performance of Bolt on GEMMs and Conv2Ds. Figure 8a shows the speed of GEMMs in BERT with batch size=32 and sequence length=40 and two square GEMMs. Figure 8b shows the speed of 3 × 3 Conv2Ds in ResNet-50. The batch size=32 and all Conv2Ds use (1, 1) zero padding.

Evaluating persistent kernel fusion performance

- Baseline is Bolt using only epilogue fusion that computes the two GEMMs/Conv2D sequentially.
- Persistent kernel fusion accelerates the computations by 1.2x-1.5x for back-to-back GEMMs and by 1.1x-2.0x for back-to-back Conv2Ds.

Table 1. The performance of fusing two back-to-back GEMMs using persistent kernels. Each GEMMs is followed by a ReLU epilogue and all of them will be fused into one kernel.

1st GEMM			2nd GEMM			Normalized speed	
M	N	K	M	N	K	w/o fuse.	w/ fuse.
2464	1	4	2464	4	1	1.00	1.24
16384	64	256	16384	16	64	1.00	1.34
32768	128	576	32768	64	128	1.00	1.28
128320	32	96	128320	96	32	1.00	1.46

Table 2. The performance of fusing two back-to-back Conv2Ds using persistent kernels. Each Conv2D is followed by a BiasAdd and a ReLU epilogue. The 3×3 Conv2D uses (1, 1) padding and the 1×1 Conv2D uses (1, 1) strides and does not have padding.

3×3 Conv2D			1×1 Conv2D		Normalized speed	
H, W	IC, OC	strides	H, W	IC, OC	w/o fuse.	w/ fuse.
224^2	3, 48	(2, 2)	112^2	48, 48	1.00	1.10
112^2	48, 48	(2, 2)	56^2	48, 48	1.00	1.41
56^2	48, 48	(1, 1)	56^2	48, 48	1.00	1.87
224^2	3, 64	(2, 2)	112^2	64, 64	1.00	1.24
112^2	64, 64	(2, 2)	56^2	64, 64	1.00	1.12
56^2	64, 64	(1, 1)	56^2	64, 64	1.00	2.02

Evaluating padding performance and overhead

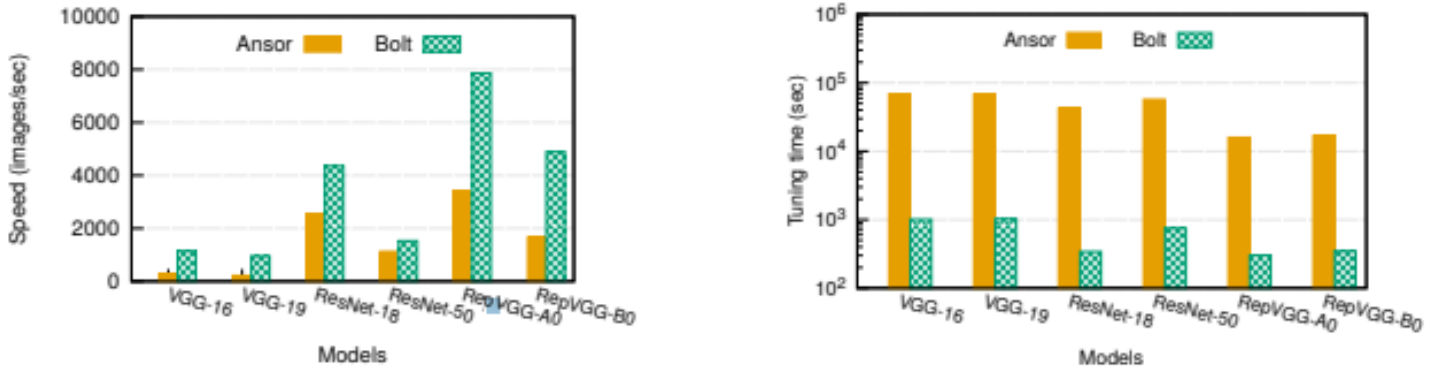
- When comparing the performance of Bolt with and without padding, we see that the speed can be improved by 1.8x on average.
- However, the padding adds extra overhead of 16% on average

Table 3. The performance and overhead of Bolt's automated padding. Unpadded Conv2Ds are computed with alignment=2; after being padded, alignement=8 can be used. The cost of padding is the time spent on the padding over the total computation time (padding+Conv2D).

N	H, W	IC, OC	kernel	padding	Norm. speed		Cost
					unpad	pad	
32	20, 26	46, 32	(3, 3)	(1, 1)	1.00	1.62	18%
32	20, 26	46, 32	(5, 5)	(2, 2)	1.00	1.95	9%
128	14, 19	46, 32	(5, 7)	(0, 0)	1.00	1.77	15%
288	11, 15	46, 32	(5, 7)	(0, 0)	1.00	1.71	18%
32	20, 26	174, 64	(3, 3)	(1, 1)	1.00	1.60	24%
32	20, 26	174, 64	(5, 5)	(2, 2)	1.00	1.99	12%

Evaluating end-to-end optimization

- Baseline is Ansor with 900 x # of tasks for the tunning trials.
- Bolt is 4.2x faster on VGG models, 1.5x on ResNet, and 2.6x on RepVGG. On average, it improves inference speed by 2.8x.
- Bolt finishes tuning withing 20 minutes, against 12 hours average on Ansor.



(a) Inference speed.

(b) Tunning time.

Figure 10. The normalized inference speed and tuning time for widely used convolutional neural networks.

System-friendly models: RepVGG case study

Changing activation functions:

- Augment RepVGG by trying different activation functions.
- Results show that activation functions can indeed impact accuracy.
- Inference speed stays more or less unaffected.

Deepening the model:

- It can improve accuracy with minimal speed lost.
- Accuracy increases by up to 0.82% and the average speed drop is 15.3%.

Combined effect: results show that designing models in a system-friendly way can improve accuracy more efficiently.

Table 4. The top-1 accuracy and speed of RepVGG-A0 using different activation functions (120 epochs + simple data augmentation).

Activation	Top-1 accuracy	Speed (images/sec)
ReLU	72.31	5909
GELU	72.38	5645
Hardswish	72.98	5713
Softplus	72.57	5453

Table 5. The top-1 accuracy and speed of original RepVGG models and their augmentation with 1×1 Conv2Ds (200 epochs + simple data augmentation).

Model	Top-1 accuracy	Speed	Params
RepVGG-A0	73.05	7861	8.31
RepVGG-A1	74.75	6253	12.79
RepVGG-B0	75.28	4888	14.34
RepVGGAug-A0	73.87	6716	13.35
RepVGGAug-A1	75.52	5241	21.7
RepVGGAug-B0	76.02	4145	24.85

Table 6. The top-1 accuracy and speed of original RepVGG models and their augmentation with 1×1 Conv2Ds+Hardswish (300 epochs + advanced augmentation, label smoothing, and mixup).

Model	Top-1 accuracy	Speed (images/sec)
RepVGG-A0	73.41	7861
RepVGG-A1	74.89	6253
RepVGG-B0	75.89	4888
RepVGGAug-A0	74.54	6338
RepVGGAug-A1	76.72	4868
RepVGGAug-B0	77.22	3842

Pros

- Bolt is pioneer in bridging the gap between auto-tuners and device library information.
- Unlike existing operator fusion approaches that only fuses one GEMM/Conv and its adjacent operators (e.g. BiasAdd, ReLU), persistent kernel fusion can fuse sequences of GEMMs and Convs.
- Achieves both significant inference speed ups over Ansor, while also decreasing the tuning time.
- Automates finding optimal parameters for templates using lightweight profiler.
- Introduces new best practices for model design for hardware efficiency.

Cons

- Focused only on NVIDIA GPUs/CUTLASS library, not demonstrated for other platforms.
- Persistent fusion benefits memory-bound operations; tradeoffs for compute-bound operations not quantified.
- Evaluated for inference only, training use cases not explored.
- No analysis of energy efficiency – important for mobile/embedded deployment.
- Overhead of kernel launch with persistent fusion not analyzed, only for padding.