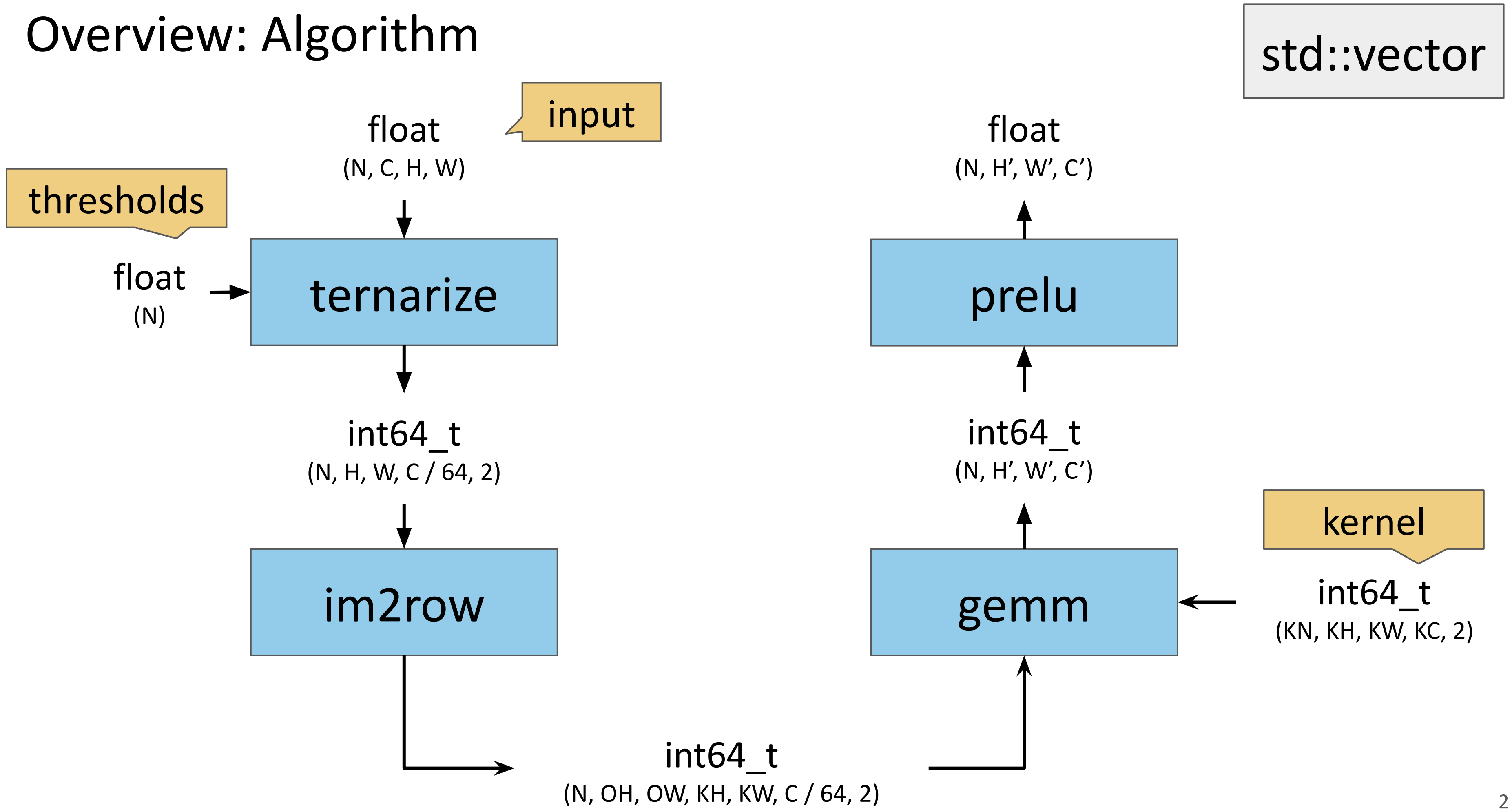
Bitwise Convolution for Ternary Neural Networks

Felix Möller, Daniel Nezamabadi, Rudy Peterson, Luca Tagliavini
Supervisor: Shien Zhu

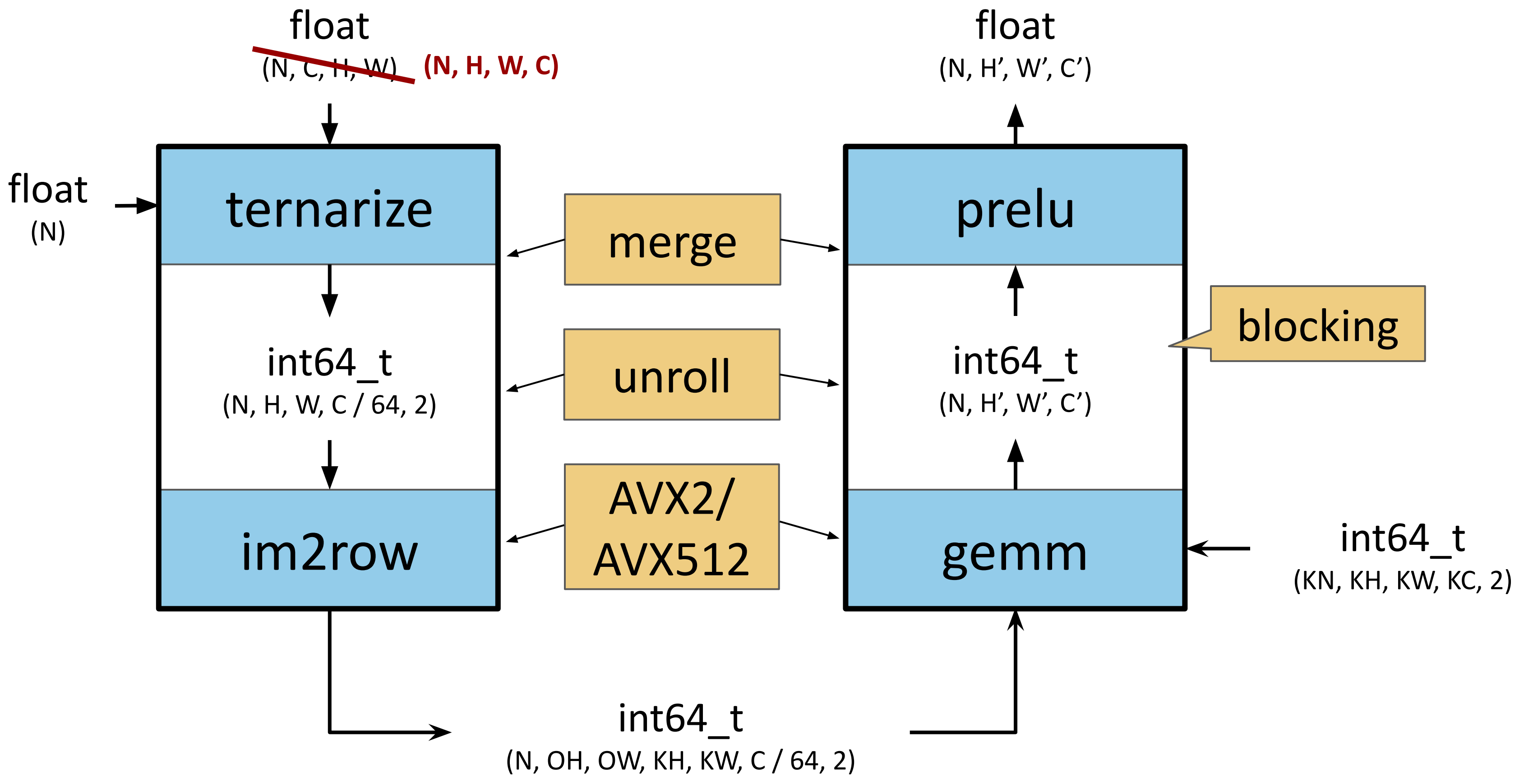
Advanced Systems Lab - Final Presentation, ETH Zurich
June 7th, 2024

Overview: Algorithm



Overview: Our Work

Tensor ~~std::vector~~



Using Tensors

```
class Tensor3D {
  T *data;
  const size_t dim1;
  const size_t dim2;
  const size_t dim3;
}
```

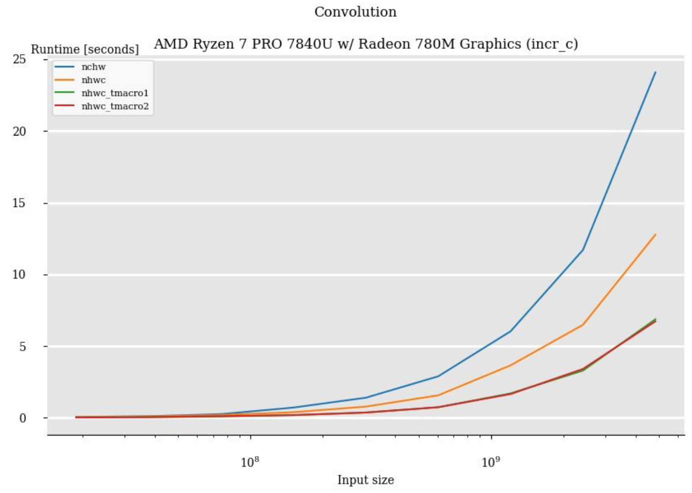
nhwc_tmacro1

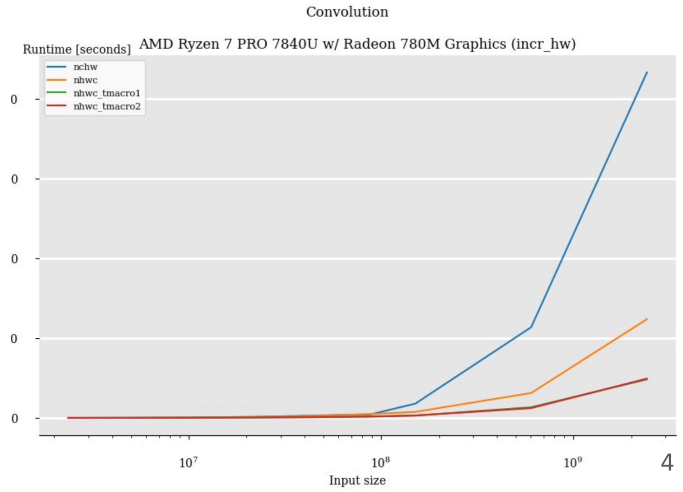
```
#define tensor3d_set(data, dim2, dim3, value, i, j, k)          \
  ((data)[((i) * ((dim2) * (dim3))) + ((j) * (dim3)) + (k)] = (value))
```

nhwc_tmacro2

```
#define tensor3d_set(data, dim2, dim3, value, i, j, k)          \
  ((data)[((i) * (dim2) + (j)) * (dim3) + (k)] = (value))
```

gcc with O3 and disabled vectorization



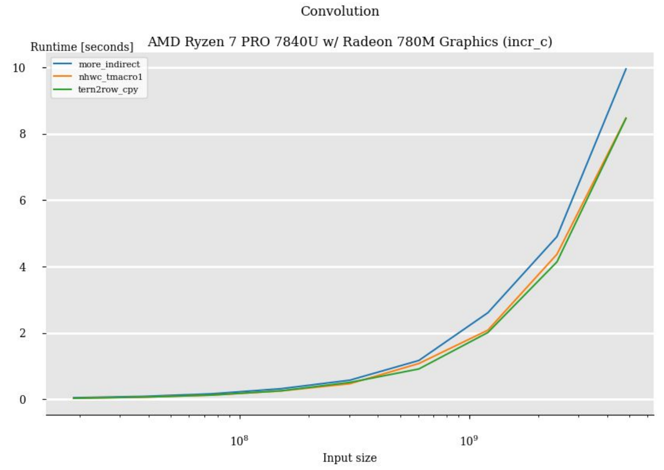


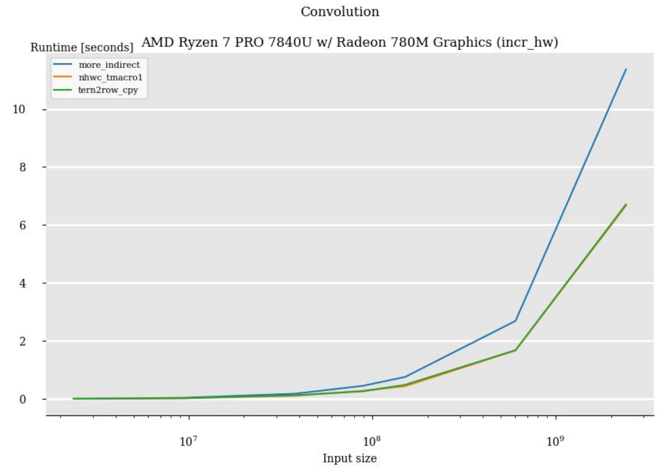
Merging im2row

Don’t merge im2row

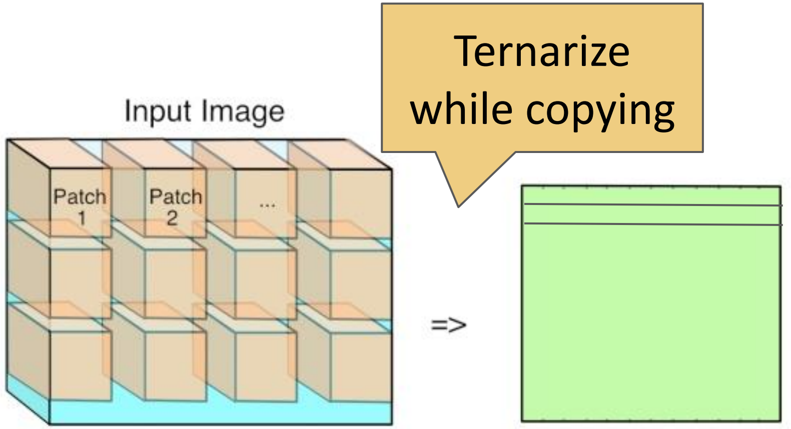
Merge ternarize and im2row

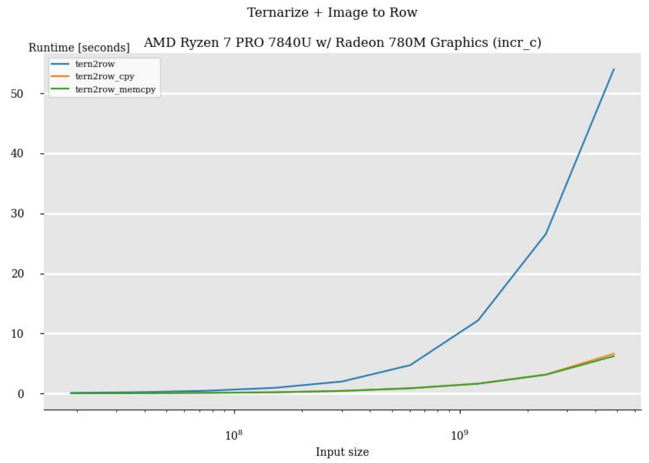
Merge im2row and gemm (indirect)





ternarize + im2row





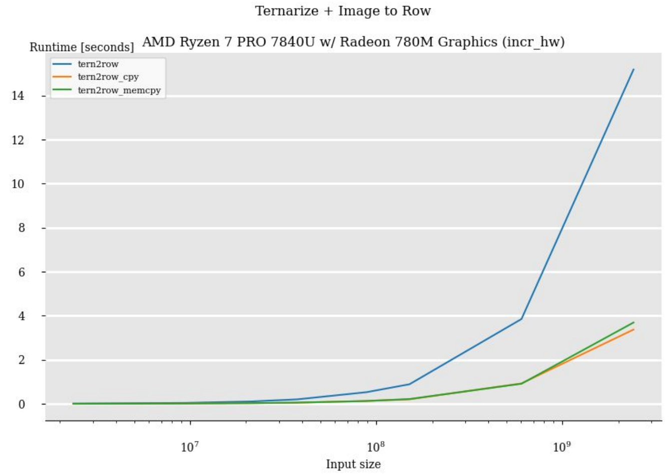
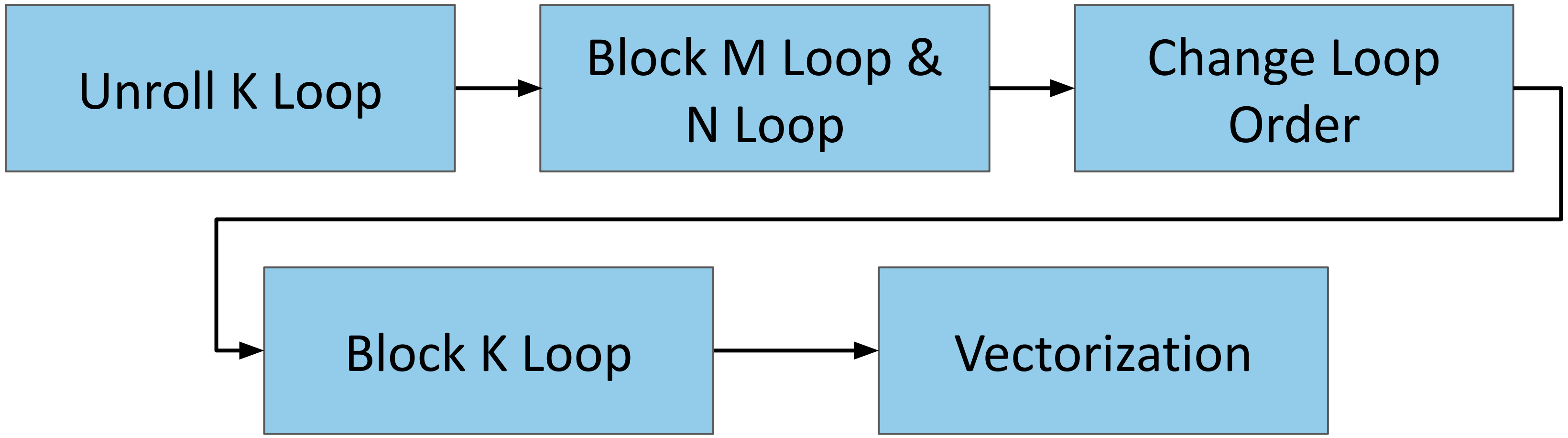
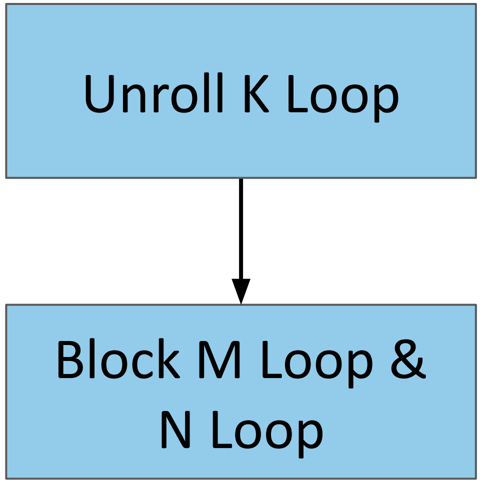


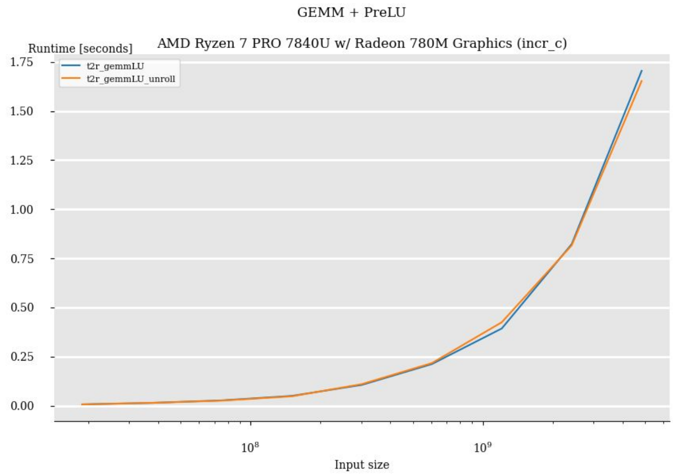
Image adapted from https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/convolution_layer/making_faster

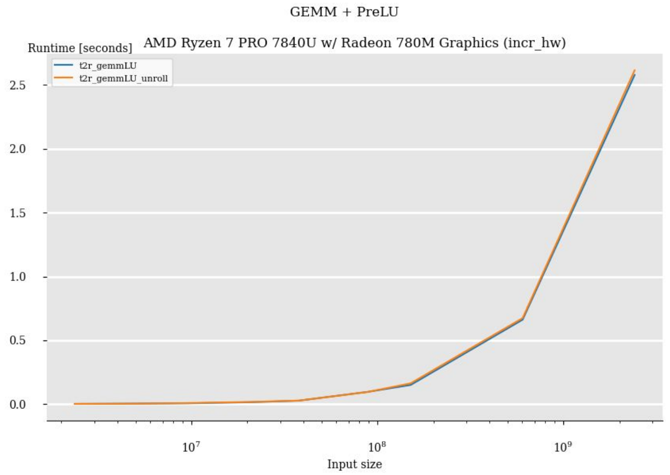
GEMM + PReLU - Optimization Plan



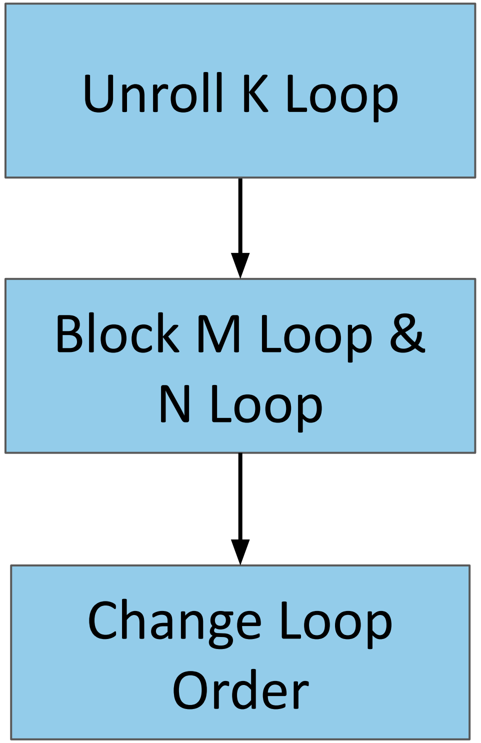
GEMM + PReLU - Unrolling over K & Blocking over M and N

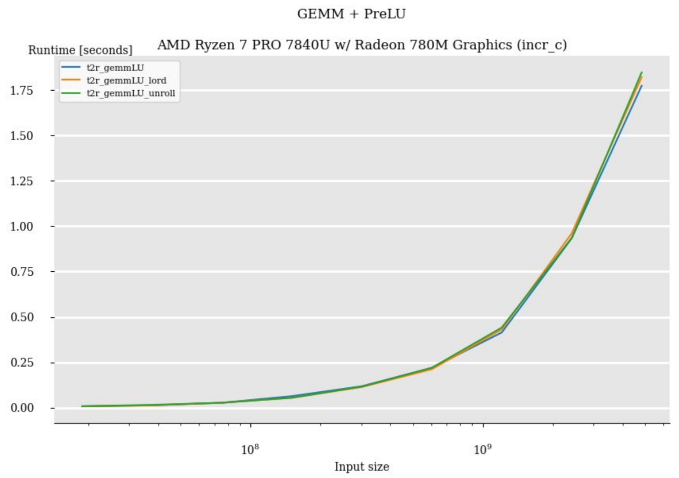


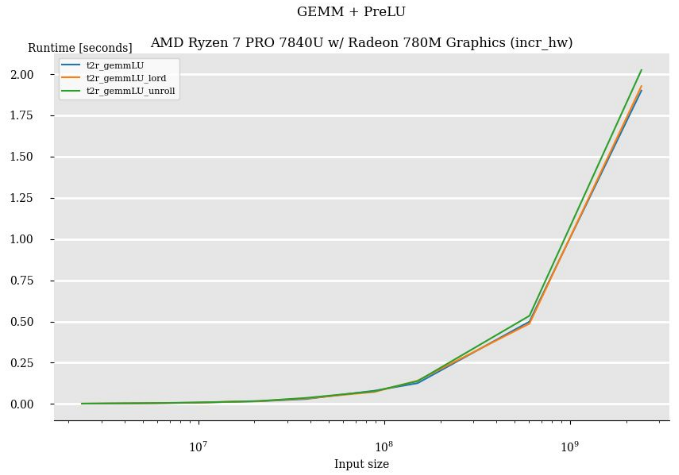




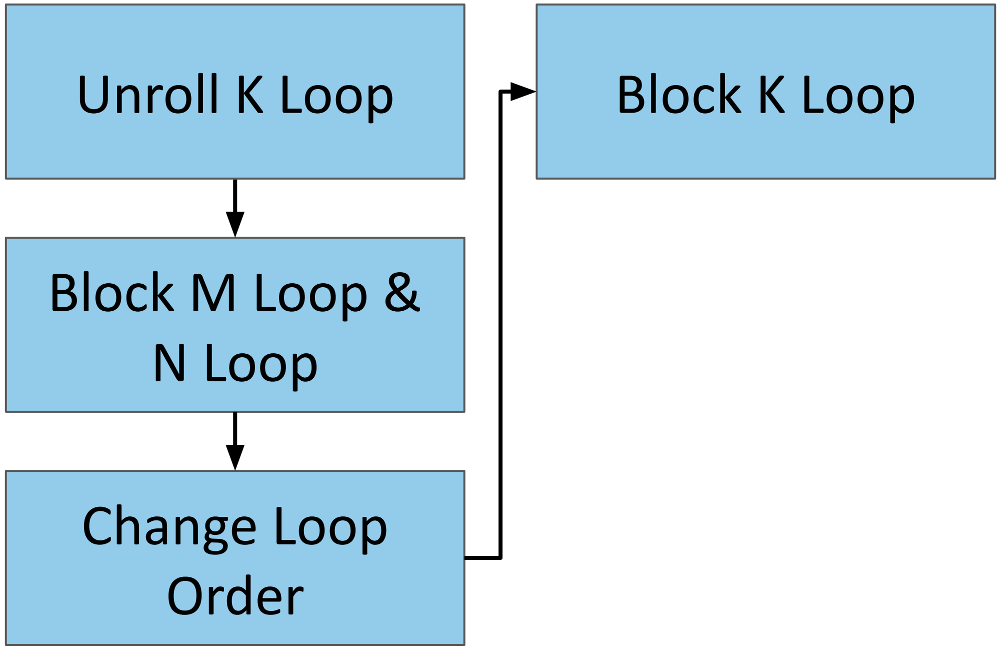
GEMM + PReLU - Changing the Loop Order

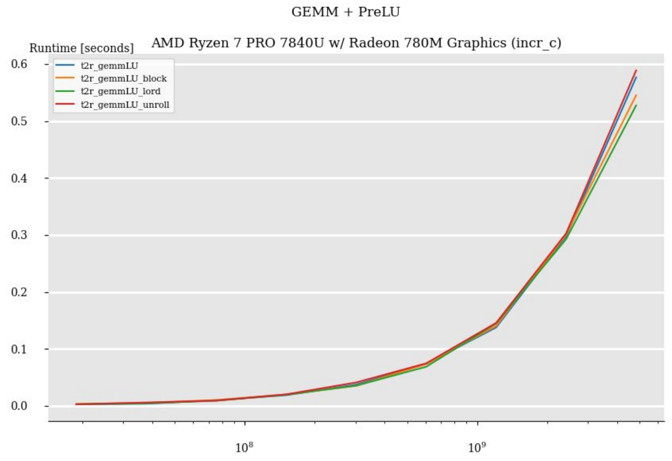


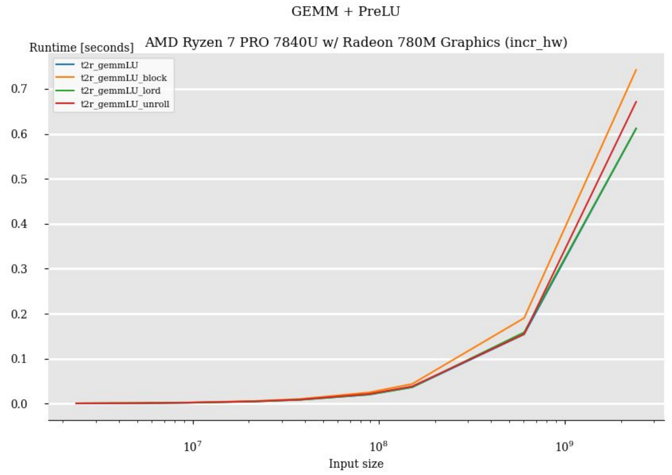




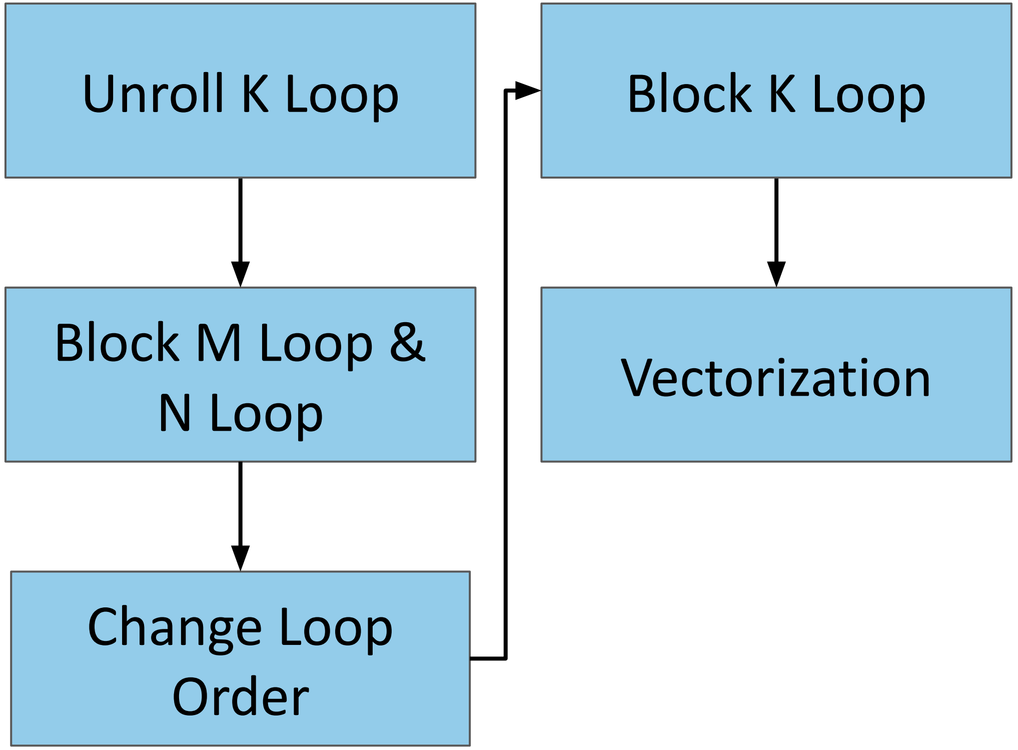
GEMM + PReLU - Blocking over K

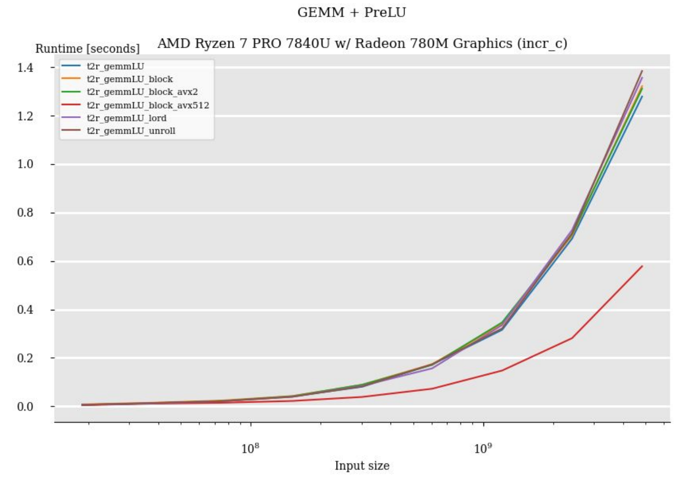


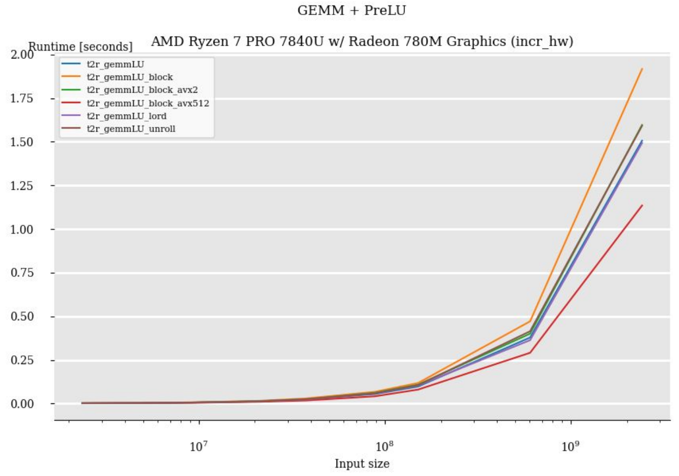




GEMM + PReLU - Vectorization with AVX2 and AVX512





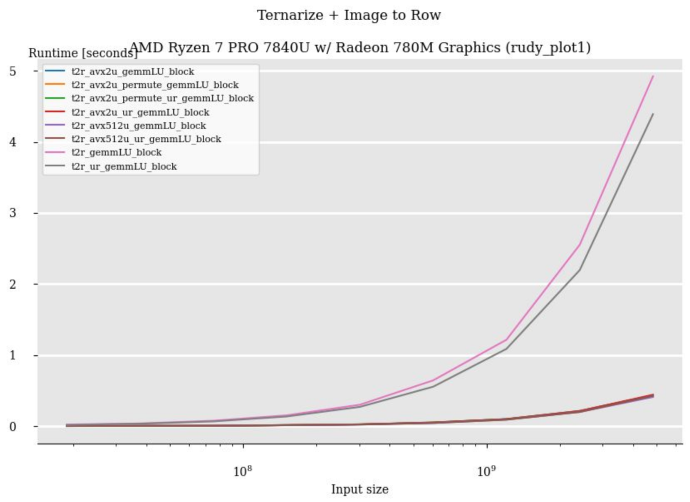


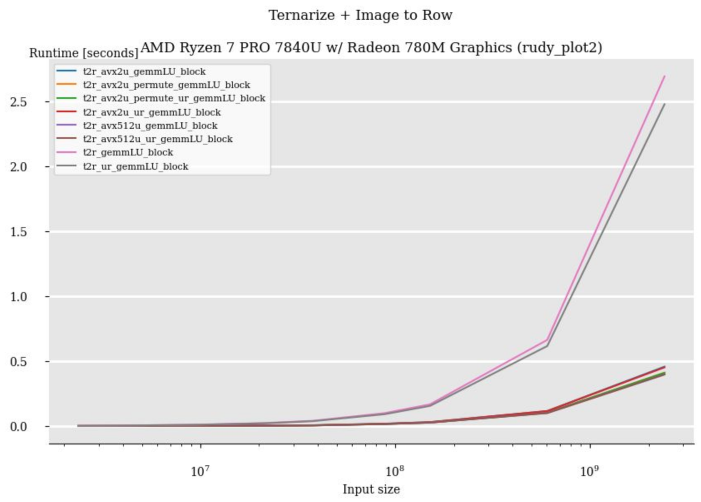
Optimizing Ternarize

Loop Unrolling

AVX2

AVX512



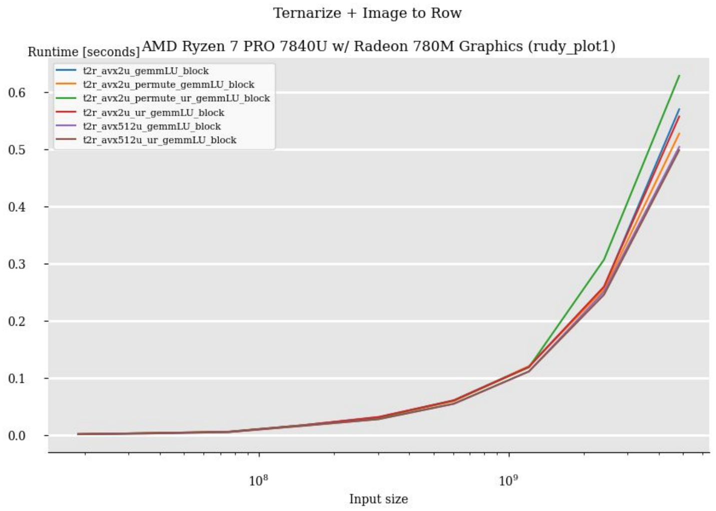


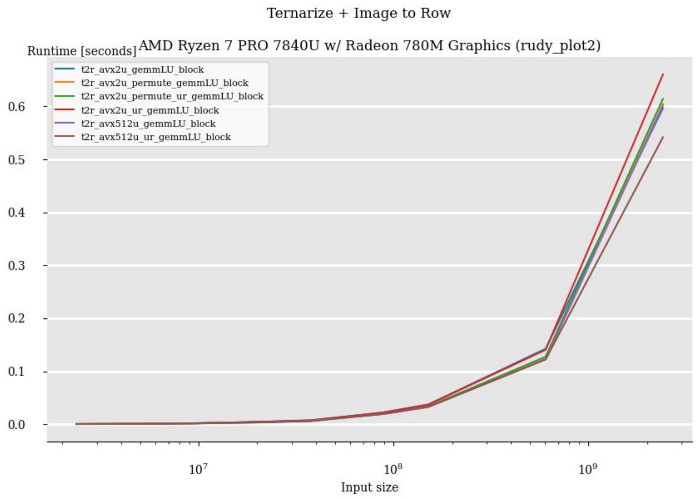
Optimizing Ternarize

Loop Unrolling

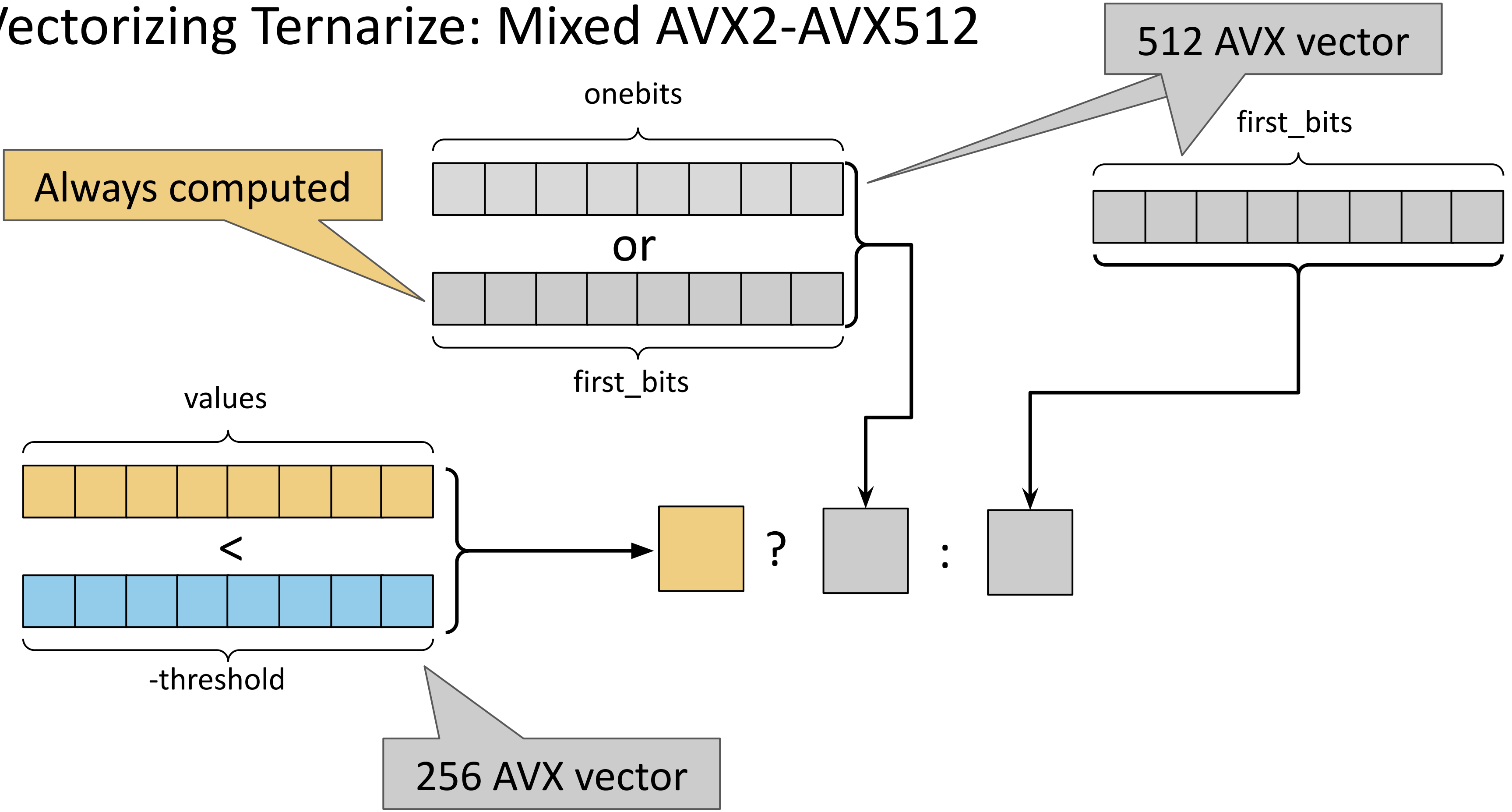
AVX2

AVX512

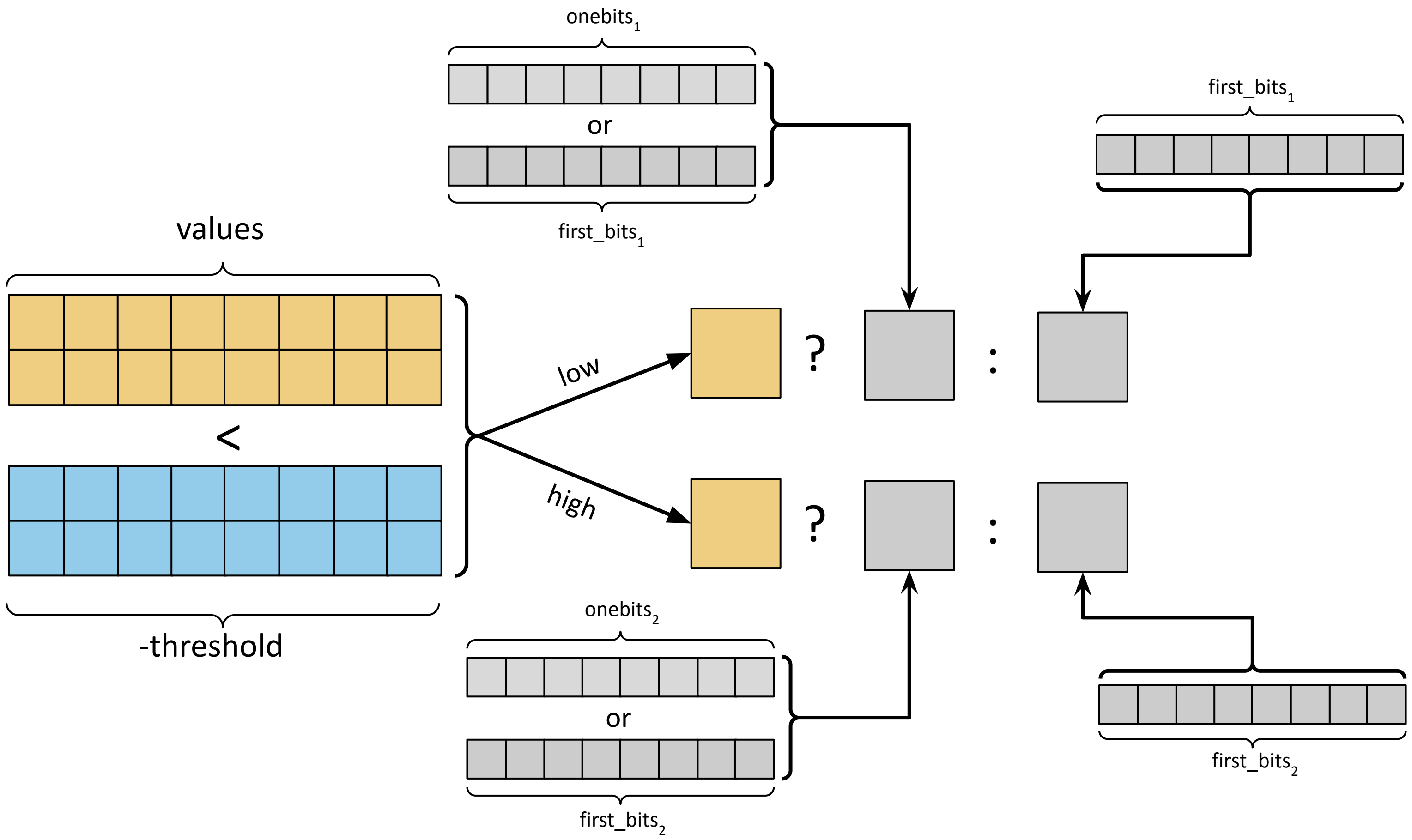




Vectorizing Ternarize: Mixed AVX2-AVX512

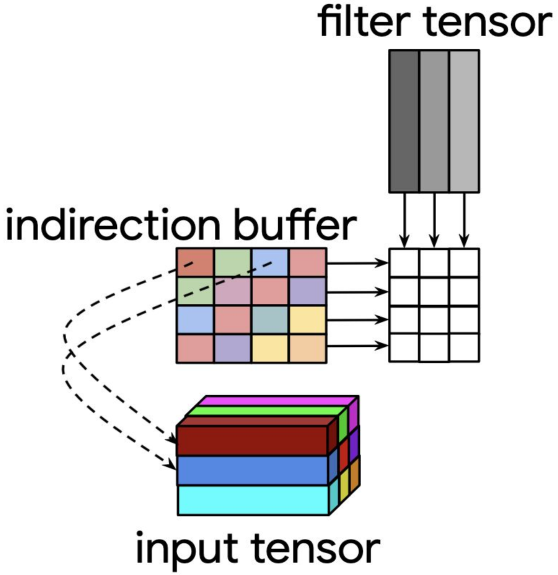


Vectorizing Ternarize: AVX512 only

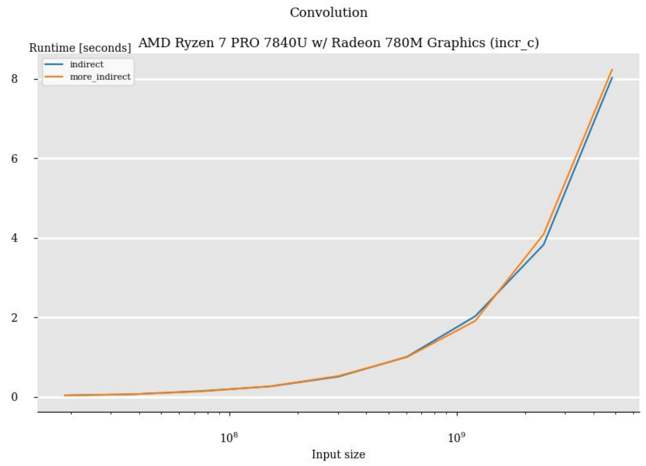


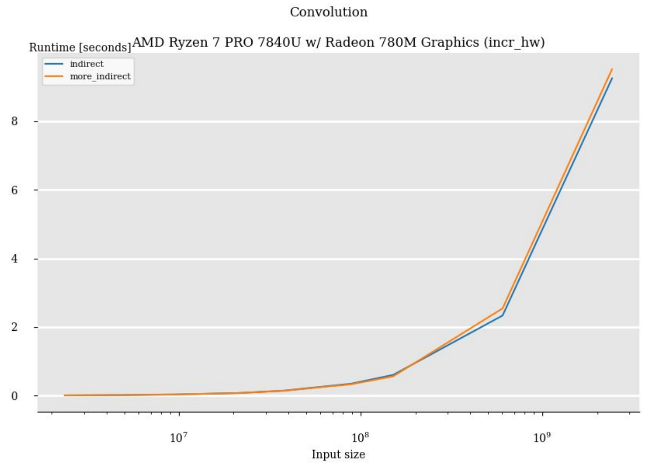
im2row + GEMM

Indirect Convolution

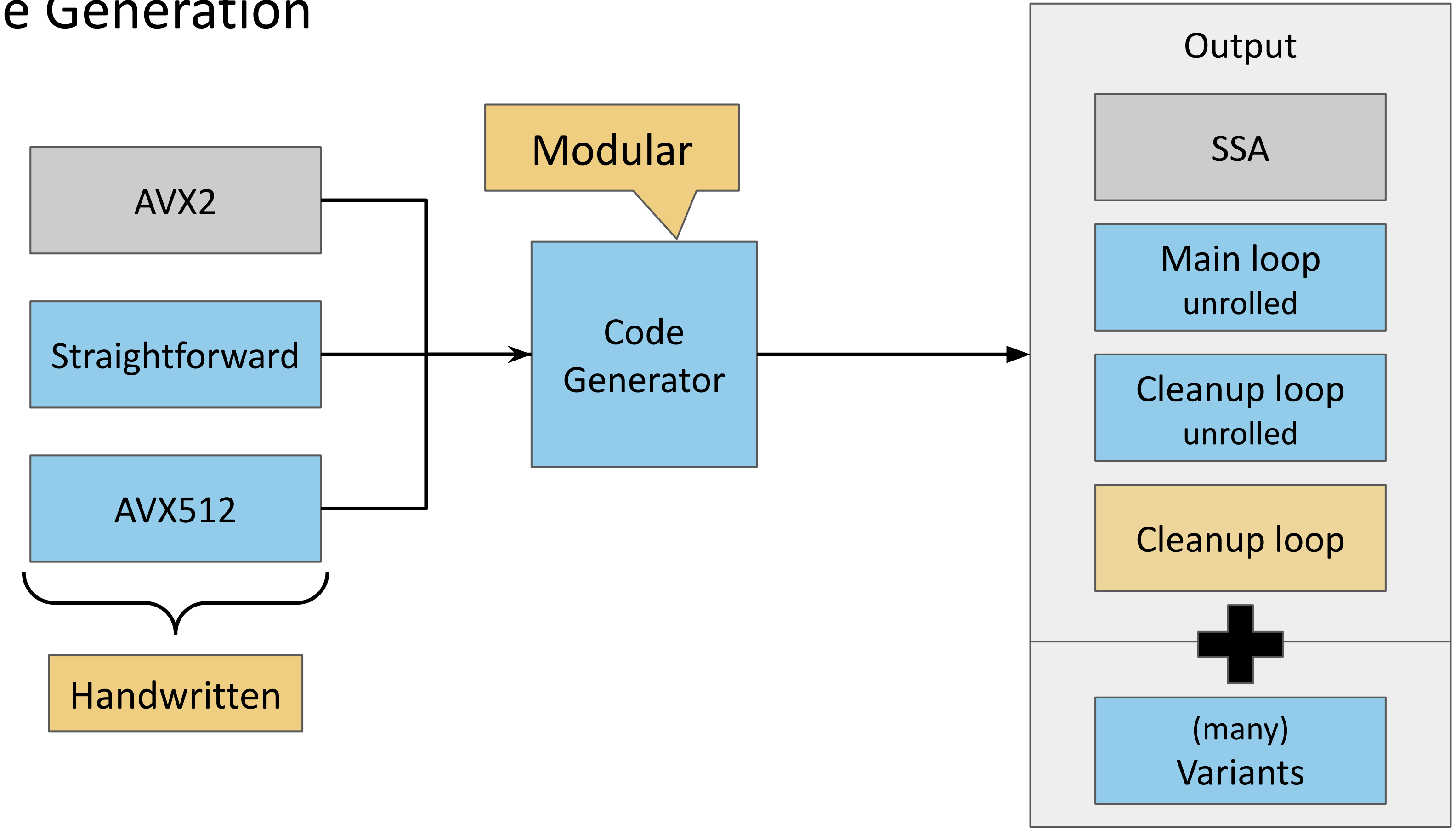


Source: M.Dukhan, The Indirect Convolution Algorithm, *arxiv 2019*





Code Generation



Code Generation - Variant: Autotuning + libpopcnt

TNN GEMM

Autotuning on M and N

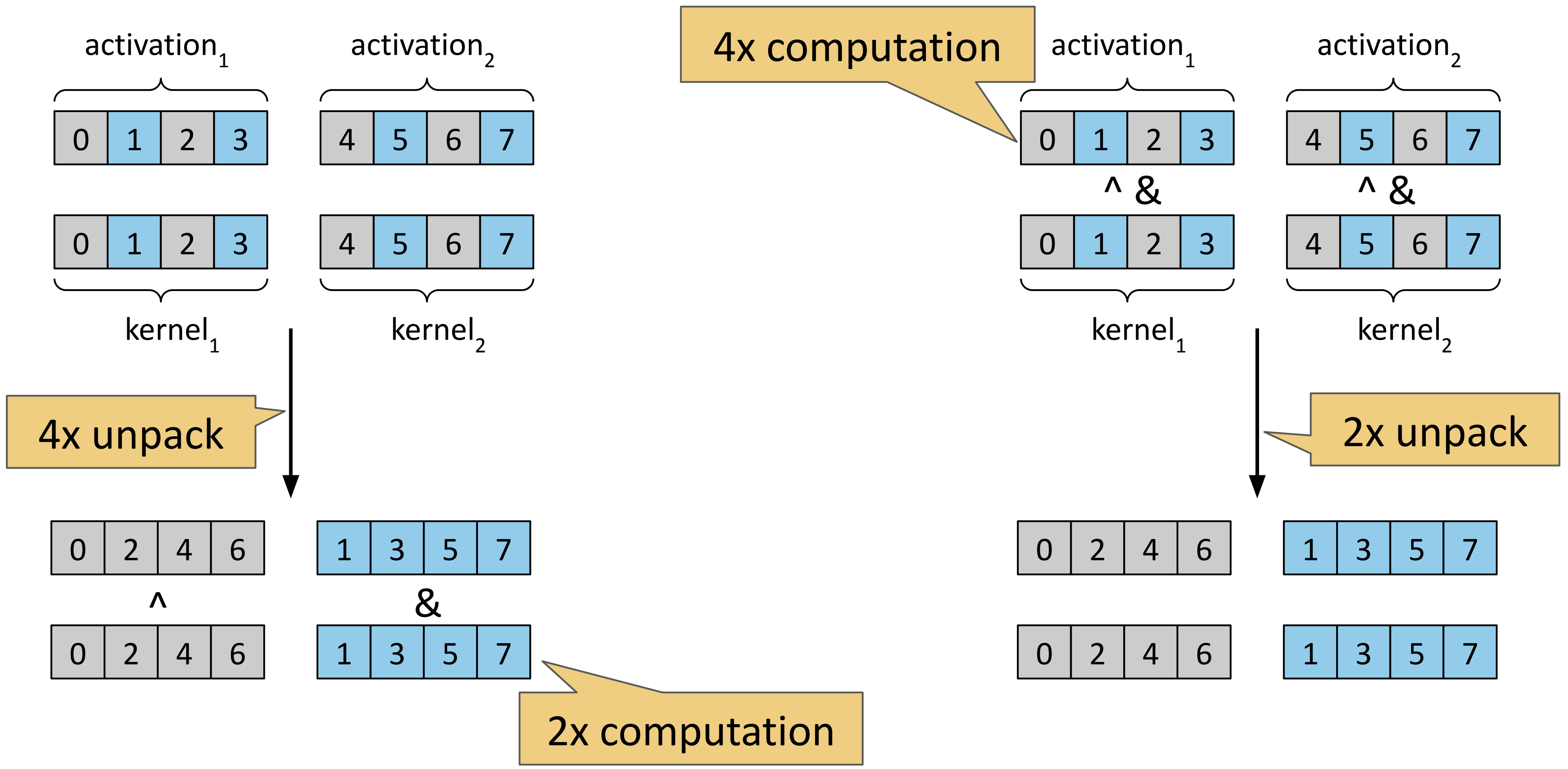
```
for m in range(0, M):
    for n in range(0, N):
        cntp1 = cntp2 = 0
        for k in range(0, K, 2):
            ...
            cntp1 += popcnt(p2)
            cntp2 += popcnt(p1 & p2)

        output[m,n] = ..
```

TNN GEMM

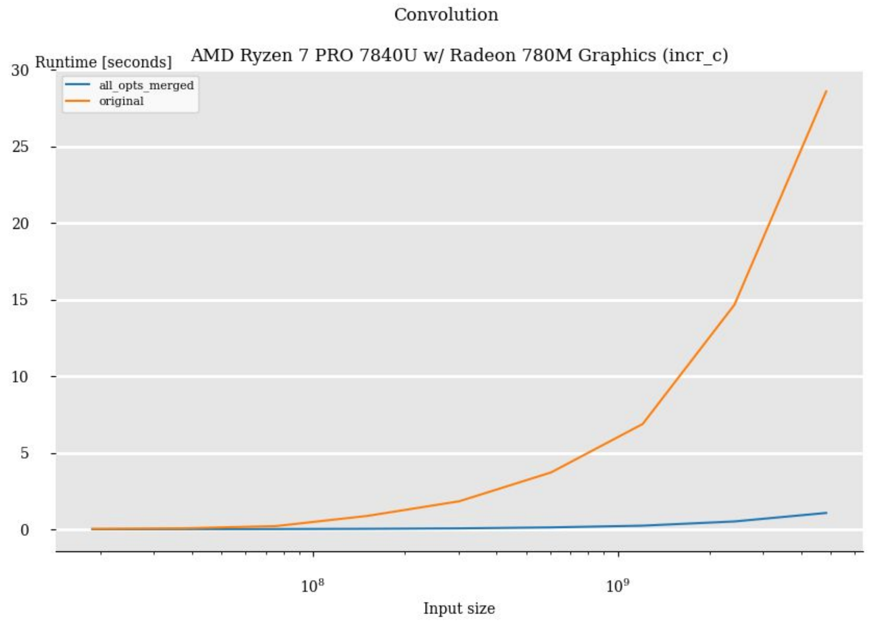
popcount on a **big** vector

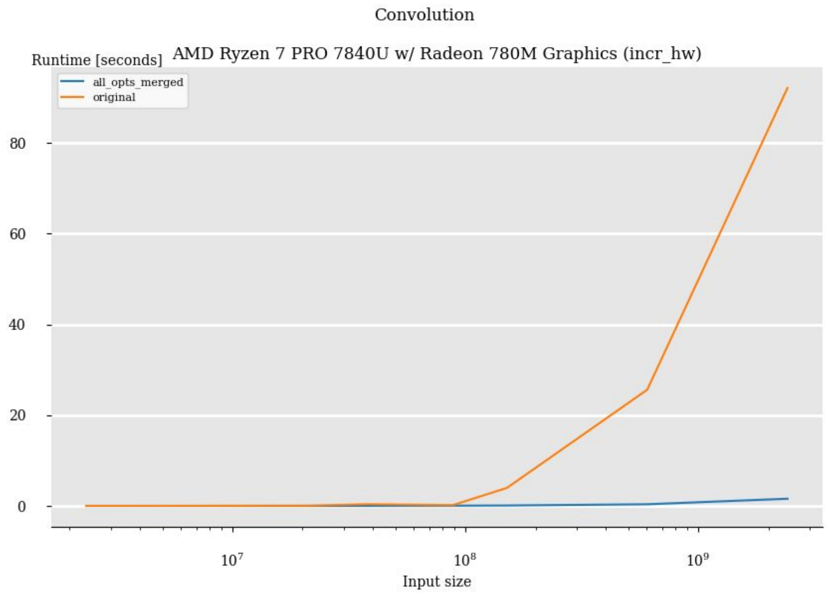
```
for m in range(0, M):
    for n in range(0, N):
        vcntp1[], vcntp2[]
        for k in range(0, K, 2):
            ...
            vcntp1[k/2] = p2
            vcntp2[k/2+1] = p1 & p2

        cntp1 = libpopcnt(vcntp1)
        ..
```

Code Generation - Variant: Less Unpacking

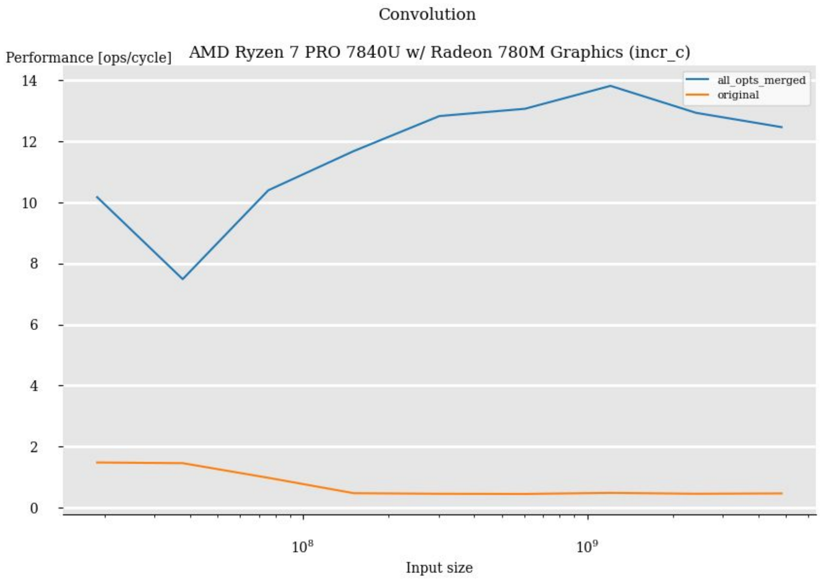


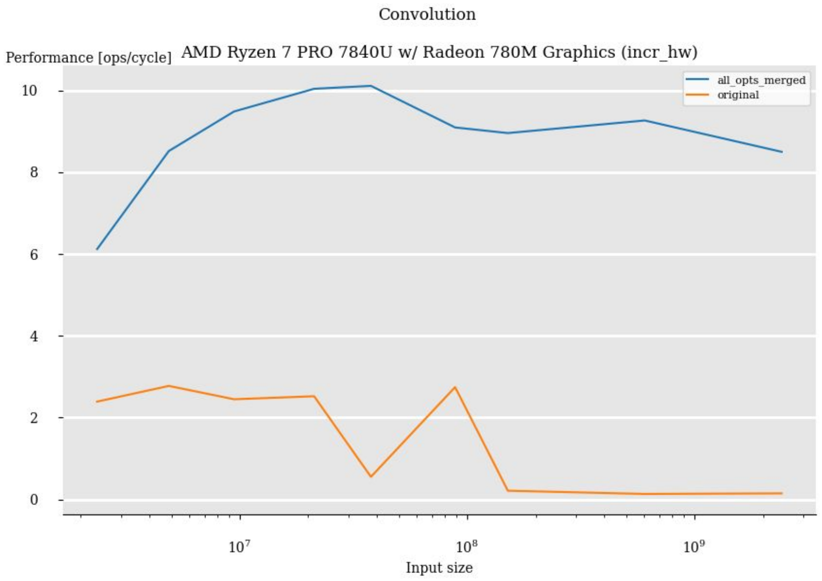
Conclusion


Convolution
AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics (incr_c)
Runtime [seconds]
all_opts_merged
original
30
25
20
15
10
5
0
10^8
10^9
Input size


Convolution
AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics (incr_hw)
Runtime [seconds]
all_opts_merged
original
80
60
40
20
0
10^7
10^8
10^9
Input size

Conclusion


Convolution
AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics (incr_c)
Performance [ops/cycle]
all_opts_merged
original
Input size
14
12
10
8
6
4
2
0
10^8
10^9


Convolution
AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics (incr_hw)
Performance [ops/cycle]
all_opts_merged
original
Input size
10
8
6
4
2
0
10^7
10^8
10^9