OPTIMIZING TORCH PERFORMANCE FOR INTEL® XEON PHI™ PROCESSORS

Colfax International — *colfaxresearch.com*

September 2016

SCOPE OF THE WEBINAR

- What you can expect to learn:
  - What code modernization is.
  - How performance optimization affects Machine Learning speed.
  - Practical knowledge of parallelism and how to take advantage of it.
    - OpenMP and thread parallelism.
    - Automatic Vectorization.
- What will not be covered:
  - Theory behind Convolutional Neural Networks (CNN) or how to use it.
  - Discussion of scoring performance.
  - Basics of how to use Torch and its packages.

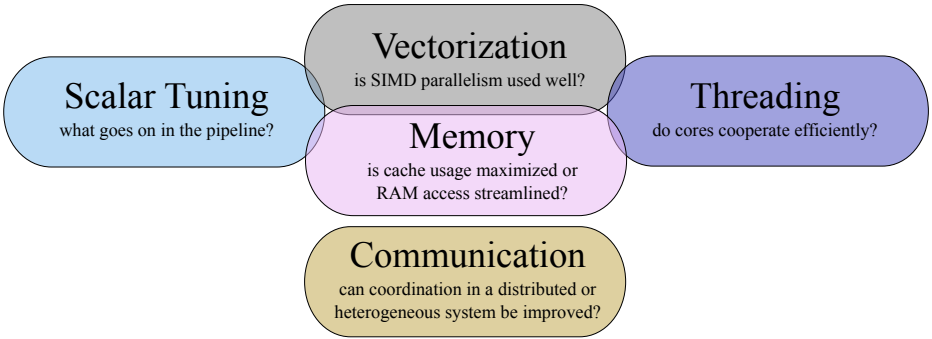
§2. CODE MODERNIZATION & MACHINE LEARNING

CODE MODERNIZATION

WHAT IS CODE MODERNIZATION

Code Modernization

Optimizing software to better utilize features available in modern computer architectures.



CASE STUDY: VGGNET ON TORCH

IMAGE RECOGNITION: CONVOLUTIONAL NEURAL NETWORKS

Image Recognition: identifying objects based on image data.

a man is playing tennis on a tennis court


a train is traveling down the tracks at a train station


a cake with a slice cut out of it


a bench sitting on a patch of grass next to a sidewalk

source: Karpathy, A. NeuralTalk2 GitHub page. https://github.com/karpathy/neuraltalk2

- Deep Convolutional Neural Networks (CNN)
  - Gained prominence in this domain in the recent years:
  - Ex. AlexNet, VGG-Net, GoogLeNet, ResNet etc.

(BRIEF) INTRODUCTION TO VGG ARCHITECTURE

VGG-Net architecture: Gained prominence through ImageNet challenge

Visual Geometry Group's page on VGG-Net

- Convolution:
  - Matrix Multiplication
  - Memcopy
- ReLU:
  - Streaming data read
  - Comparisons
- Maxpooling:
  - Streaming data read
  - Max

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Source: Simonyan, K and Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition", Visual Geometry Group, Department of Engineering Science, University of Oxford, Conference paper at ICLR 2015 [PDF]

TORCH FRAMEWORK

Torch Machine Learning Framework:

- Languages
  - Lua front-end
  - C back-end
- Extended with additional packages
  - nn - Neural network package
    - https://github.com/torch/nn
  - image - Utility package for images
    - https://github.com/torch/image
  - nngraph - graphs for nn
    - https://github.com/torch/nngraph

Download:
https://github.com/torch/torch7

SAMPLE IMPLEMENTATION

Defining the neural network architecture

```
  -- Sequential is a container where the added layers --
  -- are applied sequentially                          --
  local model = nn.Sequential()
  model:add(nn.SpatialConvolution(3,64, 3,3, 1,1, 1,1))
  model:add(nn.ReLU(true))
  model:add(nn.SpatialConvolution(64,64, 3,3, 1,1, 1,1))
  model:add(nn.ReLU(true))
  model:add(nn.SpatialMaxPooling(2,2,2,2):ceil())
  -- ... rest of the layers ...  --
```

Using the network for inference

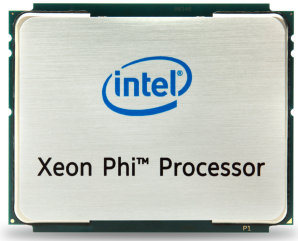
```
  vgg_model:training()
-- Forward Pass (Batch mode)
  local feat = vgg_model:forward(imgs)
  local obj = criterion:forward(feat, labels)
```

PLATFORM AND BENCHMARKING METHOD

INTEL XEON PHI PROCESSORS (2ND GEN)

Specialized platform for demanding computing applications.

- Socket version or coprocessor
- 64-72 cores × 4 HT at 1.3-1.5 GHz
- 3+ TFLOP/s in DP (FMA)
- 6+ TFLOP/s in SP (FMA)
- ≤ 384 GiB DDR4 (> 90 GB/s)
- 16 GiB HBM (MCDRAM, > 400 GB/s)
- Binary-compatible with Xeon
- Common OS (RHEL/CentOS/Fedora/SUSE/Windows)



BENCHMARKING METHODOLOGY

`walltime.c`: a custom timer for benchmarking

```c
#include <omp.h>
static int walltime(lua_State * L) {
    double time = omp_get_wtime();
    lua_pushnumber(L, time);
    return 1; }
```

```
for i=1,#self.modules do
   local t0 = walltime()
   currentOutput = self:rethrowErrors(self.modules[i], i, 'updateOutput' ...
   local t1 = walltime()
end
```

```
local t0 = walltime()
local feat = vgg_model:forward(imgs)
local t1 = walltime()
```

ADDITIONAL NOTES ON BENCHMARKING

System Configurations:

- Intel® Xeon Phi™ Processor 7210
  - Quadrant/Cache mode
- Intel® C/C++ Compiler 2017

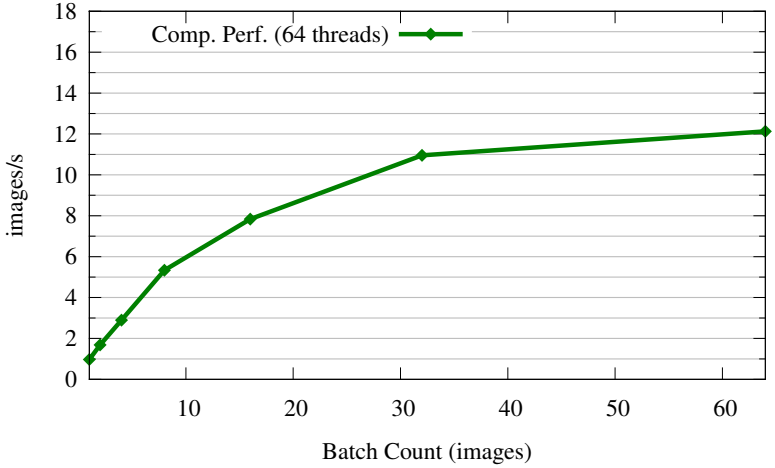
Torch/THNN:

- Torch 7: git commit 9ed072f
- THNN: git commit 09121e9
- Batch Computation with THNN

Environment Variables:

```
user@knl% KMP_AFFINITY=scatter
user@knl% OMP_NUM_THREADS=64
user@knl% OMP_BLOCKTIME=infinity
```

BASE COMPUTATIONAL PERFORMANCE



By Layer:

- ReLU: 66%
- Conv: 30%
- MaxPool: 3%
- Other: <1%

§3. OPTIMIZATION OF RELU LAYER

DIFFICULTY WITH THE EXISTING IMPLEMENTATION

ReLU: ~/torch/extra/nn/lib/THNN/generic/Threshold.c

```
1     THTensor_(resizeAs)(output, input);
2     TH_TENSOR_APPLY2(real, output, real, input,
3       *output_data = (*input_data > threshold) ? *input_data : val;
4     );
```

Tensor_Apply_2 (≈150 lines): ~/torch/pkg/torch/lib/TH/THTensorApply.h

```
1  #define TH_TENSOR_APPLY2(TYPE1, TENSOR1, TYPE2, TENSOR2, CODE) \
2  ..... \
3      for(; TENSOR1##_i < TENSOR1##_size && TENSOR2##_i < TENSOR2##_size;        \
4              TENSOR1##_i++, TENSOR2##_i++, TENSOR1##_data += TENSOR1##_stride, \
5                                            TENSOR2##_data += TENSOR2##_stride) \
6      { \
7        CODE \
8      } \
```

VECTORIZATION

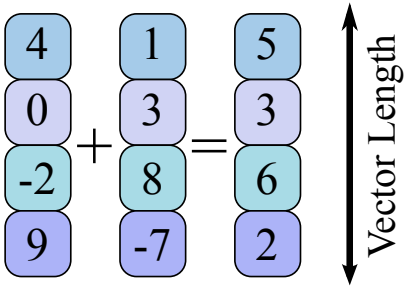
SHORT VECTOR SUPPORT

Vector instructions – one of the implementations of SIMD (Single Instruction Multiple Data) parallelism.

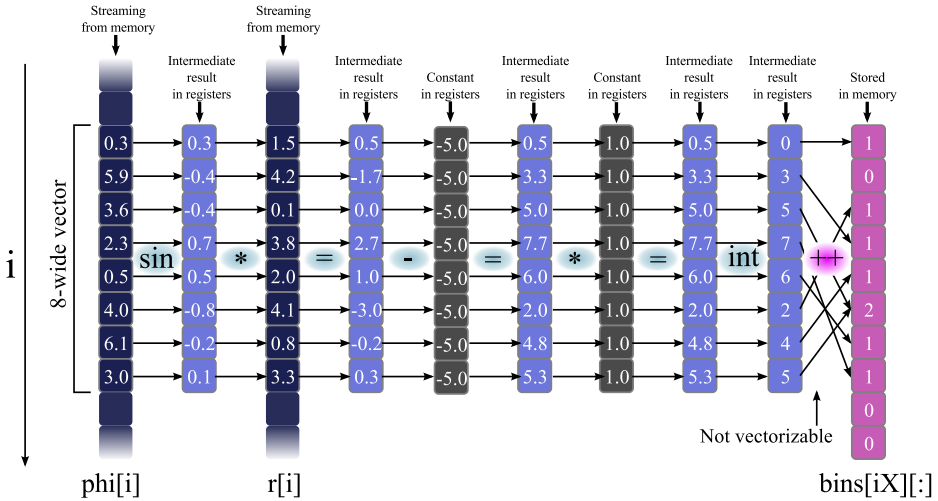
Scalar Instructions

$$4 + 1 = 5$$
$$0 + 3 = 3$$
$$-2 + 8 = 6$$
$$9 + -7 = 2$$

Vector Instructions



VECTORIZED LOOPS MAY BE COMPLEX



VECTORIZATION: TWO APPROACHES

Automatic Vectorization:

- Vectorization w/ compiler.
- Portable: just recompile.
- Tuning with directives.

```
1 double A[vec_width], B[vec_width];
2 // ...
3
4
5 // This loop will be auto-vectorized
6 for(int i = 0; i < vec_width; i++)
7   A[i]+=B[i];
```

```
1 double A[vec_width], B[vec_width];
2 // ...
3 // This is explicitly vectorized
4 __m512d A_vec = _mm512_load_pd(A);
5 __m512d B_vec = _mm512_load_pd(B);
6 A_vec = _mm512_add_pd(A_vec,B_vec);
7 _mm512_store_pd(A,A_vec);
```

Explicit Vectorization:

- Vectorization w/ intrinsics
- Full control over instructions
- Limited portability

AUTOMATIC VECTORIZATION

```
1 // ... foo.cc ... //
2 double A[n], B[n];
3 for(int i = 0; i < n; i++)
4   B[i] = A[i] + B[i];
```

Intel Compilers apply automatic vectorization by default.

```
user@knl% icpc -qopt-report foo.cc -xMIC-AVX512
user@knl% cat foo.optrpt
LOOP BEGIN at foo.cc(14,3)
  remark #15300: LOOP WAS VECTORIZED
```

Enable automatic vectorization for GCC with -O3

```
user@knl% g++ -S foo.cc -mavx512f -O3
user@knl% cat foo.s
  vmovapd -16432(%rbp,%rax), %zmm0
  vaddpd -8240(%rbp,%rax), %zmm0, %zmm0
```

OPENMP SIMD CONSTRUCT

Part of the OpenMP 4.0 standard.

```
1 // ... foo.cc ... //
2 double A[n], B[n];
3 #pragma omp simd
4 for(int i = 0; i < n; i++)
5   B[i] = A[i] + B[i];
```

Instructs compiler to vectorize the loop that follows. (Still requires `-O3` for GCC)

Also has built-in support for some common vectorization patterns such as reduction. Refer to the OpenMP documentation for more [pdf].

VECTORIZING RELU

LIMITATION OF AUTOMATIC VECTORIZATION

Adding the directive `omp simd` to try convince the compiler to vectorize does not work,

```
#define TH_TENSOR_APPLY2(TYPE1, TENSOR1, TYPE2, TENSOR2, CODE) \
..... \
_Pragma("omp simd") \
    for(; TENSOR1##_i < TENSOR1##_size && TENSOR2##_i < TENSOR2##_size;        \
            TENSOR1##_i++, TENSOR2##_i++, TENSOR1##_data += TENSOR1##_stride, \
                                          TENSOR2##_data += TENSOR2##_stride) \
    { \
      CODE \
    } \
```

because the automatic vectorization only allows one conditional. The compiler produces the following error.

```
/home/user/torch/pkg/torch/lib/TH/generic/THTensorCopy.c(7): error: parallel loop
condition operator must be one of <, <=, >, >=, or !=
    TH_TENSOR_APPLY2(real, tensor, real, src, *tensor_data = (real)(*src_data);)
```

REWRITING THE THRESHOLD

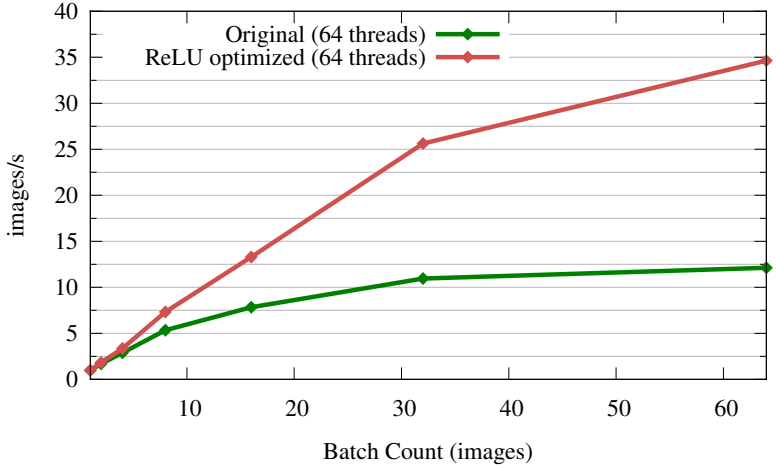
For threshold, we can simplify the code significantly by rewriting the workload.

```
1     THTensor_(resizeAs)(output, input);
2     real * in  = THTensor_(data)(input);
3     real * out = THTensor_(data)(output);
4  #pragma vector nontemporal
5  #pragma omp parallel for simd
6     for (int i = 0; i < output->storage->size; i++) {
7       out[i] = in[i] > threshold ? in[i] : val;
8     }
```

Further optimizations:

- `#pragma omp parallel for` -> For effecting thread parallelism.
- `#pragma vector nontemporal` -> For effecting streaming stores.

PERFORMANCE AFTER RELU OPTIMIZATION



RELU -> 160x boost

By Layer:

- ReLU: 1%
- Conv: 85%
- MaxPool: 11%
- Other: 3%

§4. OPTIMIZATION OF CONVOLUTION LAYER

OPENMP

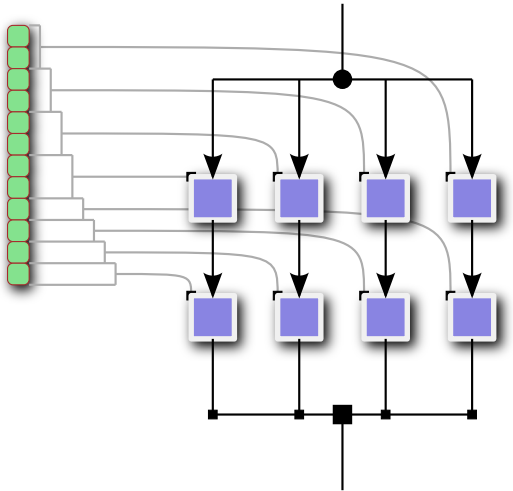
"HELLO WORLD" OPENMP PROGRAM

```
1  #include <omp.h>
2  #include <cstdio>
3
4  int main(){
5    // This code is executed by only 1 thread
6    const int nt=omp_get_max_threads();
7    printf("OpenMP with %d threads\n", nt);
8
9  #pragma omp parallel
10   {
11     // This code is executed in parallel
12     // by multiple threads
13     printf("Hello World from thread %d\n",
14                          omp_get_thread_num());
15   }
16 }
```

- OpenMP = “Open Multi-Processing” = computing-oriented framework for shared-memory programming
- Threads – streams of instructions that share memory address space
- Distribute threads across CPU cores for parallel speedup

LOOP-CENTRIC PARALLELISM: FOR-LOOPS IN OPENMP

- Simultaneously launch multiple threads
- Scheduler assigns loop iterations to threads
- Each thread processes one iteration at a time



Parallelizing a `for`-loop.

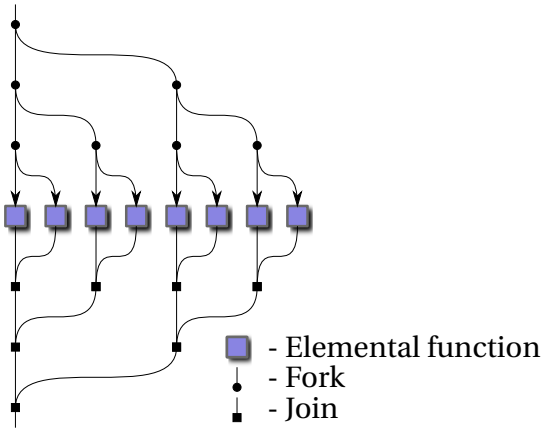
LOOP-CENTRIC PARALLELISM: FOR-LOOPS IN OPENMP

The OpenMP library will distribute the iterations of the loop following the `#pragma omp parallel for` across threads.

```
1 #pragma omp parallel for
2 for (int i = 0; i < n; i++) {
3   printf("Iteration %d is processed by thread %d\n",
4          i, omp_get_thread_num());
5   // ... iterations will be distributed across available threads...
6 }
```

FORK-JOIN MODEL OF PARALLEL EXECUTION

- Each thread can spawn daughter threads
- Available threads pick up queued tasks
- Expresses algorithms that cannot be expressed in the loop model (e.g., parallel recursion)



Fork-join model of parallel execution.

(`#pragma omp task` functionality)

TASKS IN OPENMP: EXAMPLE

```
// Starting the first task:
#pragma omp parallel
{ // Enter a parallel region
#pragma omp single
  { // Start the first task
    // from only one thread
    RecursiveWorkload(args);
  }
}
```

```
// Recursive task spawning:
void RecursiveWorkload(Arg* args) {
  if (args->size > threshold) {
    // Split work
    Arg* args1=args->FirstHalf();
    Arg* args2=args->SecondHalf();

    // Parallel divide-and-conquer
#pragma omp task firstprivate(args1)
    { RecursiveWorkload(args1); }
#pragma omp task firstprivate(args2)
    { RecursiveWorkload(args2);  }
  } else {
    // End of recursion
    args->ProcessSmallestSubTask();
  }
}
```

OPTIMIZING PARALLELISM OF CONVOLUTION

DIFFICULTY WITH THE EXISTING IMPLEMENTATION

Spatial Conv.: ~/torch/extra/nn/lib/THNN/generic/SpatialConvolutionMM.c

```
long T = input->size[0]; // Size of the batch
#pragma omp parallel for private(t)
  for(t = 0; t < T; t++) {
    THTensor *input_t = THTensor_(newSelect)(input, 0, t);
    THTensor *output_t = THTensor_(newSelect)(output, 0, t);
    THTensor *finput_t = THTensor_(newSelect)(finput, 0, t);
    THNN_(SpatialConvolutionMM_updateOutput_frame)(input_t, output_t, weight,
                                                   bias, finput_t, kW, kH, dW,
                                                   dH, padW, padH, nInputPlane,
                                                   inputWidth, inputHeight,
                                                   nOutputPlane, outputWidth,
                                                   outputHeight);
    THTensor_(free)(input_t);
    THTensor_(free)(output_t);
    THTensor_(free)(finput_t);
  }
```

DIFFICULTY WITH THE EXISTING IMPLEMENTATION

Spatial Conv.: ~/torch/extra/nn/lib/THNN/generic/SpatialConvolutionMM.c

```
1  static void THNN_(SpatialConvolutionMM_updateOutput_frame)(...) {
2    long i;  THTensor *output2d;
3    THNN_(unfolded_copy)(finput, input, kW, kH, dW, dH, padW, padH, nInputPlane,
4                         inputWidth, inputHeight, outputWidth, outputHeight);
5    output2d = THTensor_(newWithStorage2d)(output->storage, output->storageOffset,
6                                           nOutputPlane, -1,
7                                           outputHeight*outputWidth, -1);
8    if (bias) {
9      for(i = 0; i < nOutputPlane; i++)
10         THVector_(fill)(output->storage->data+output->storageOffset+
11                                                  output->stride[0]*i,
12                         THTensor_(get1d)(bias, i), outputHeight*outputWidth);
13   } else {  THTensor_(zero)(output);  }
14   THTensor_(addmm)(output2d, 1, output2d, 1, weight, finput);
15   THTensor_(free)(output2d);
16 }
```

DIFFICULTY WITH THE EXISTING IMPLEMENTATION

Unfold: ~/torch/extra/nn/lib/THNN/generic/unfold.c

```
1 void THNN_(unfolded_copy)( ... ) {
2 ...
3 // looping through the data and "unfolding"
4 #pragma omp parallel for private(k)
5   for(k = 0; k < nInputPlane*kH*kW; k++) { ... }
6 }
```

Unfold: ~/torch/pkg/torch/lib/TH/generic/TensorMath.c

```
1 void THTensor_(addmm)( ... )
2 {
3   ...
4   THBlas_(gemm)( ... ); // external call to BLAS library (e.g. OpenBLAS, MKL)
5 }
```

PARALLELIZATION STRATEGY FOR UNFOLD: COLLAPSE

Loop Collapse: combine iterations spaces of multiple loops.

```
#pragma omp parallel for collapse(2)
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++) {
      // ...
      // ...
    }
```

```
#pragma omp parallel for
  for (int c = 0; c < m*n; c++)  {
    i = c / n;
    j = c % n;
    // ...
  }
```

Applied to unfold:

```
void THNN_(batch_unfolded_copy)( ... ) {
...
 long T = input->size[0]; // Size of the batch
#pragma omp parallel for private(t, k) collapse(2)
 for(t = 0; t < T; t++)
  for(k = 0; k < nInputPlane*kH*kW; k++) { ... }
}
```

PARALLELIZATION STRATEGY FOR GEMM: DIVIDE AND CONQUER

General Matrix Multiplication (GEMM) can be divided into smaller GEMM's.

For an n by n matrices $\boldsymbol{A}$ $\boldsymbol{B}$ and $\boldsymbol{C}$,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix},$$

Multiplication $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$ can be decomposed to:

$$\boldsymbol{C}_{11} = \boldsymbol{A}_{11}\boldsymbol{B}_{11} + \boldsymbol{A}_{12}\boldsymbol{B}_{21},\ \boldsymbol{C}_{12} = \boldsymbol{A}_{11}\boldsymbol{B}_{12} + \boldsymbol{A}_{12}\boldsymbol{B}_{22},$$

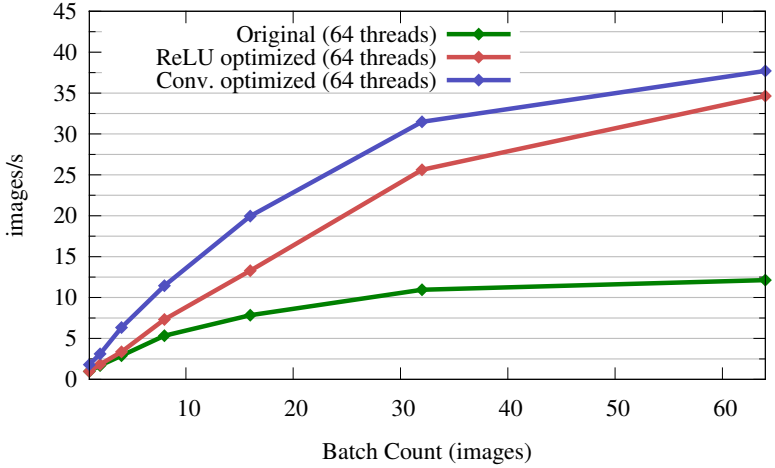
$$\boldsymbol{C}_{21} = \boldsymbol{A}_{21}\boldsymbol{B}_{11} + \boldsymbol{A}_{22}\boldsymbol{B}_{21},\ \boldsymbol{C}_{22} = \boldsymbol{A}_{21}\boldsymbol{B}_{12} + \boldsymbol{A}_{22}\boldsymbol{B}_{22}$$

PARALLELIZATION STRATEGY FOR GEMM: DIVIDE AND CONQUER

Recursively split matrices using recursive OpenMP parallelism.

```
1  void THNN_(mul_ker)( ... ){
2    ...
3    if(n<=ITILE && k<=KTILE && m<=JTILE){
4      THBlas_(gemm)(trans, trans, m,n,k,alpha,a, lda,b, ldb,beta, c, ldc);
5    }else{
6      if(m > JTILE){ // Divide along m
7        #pragma omp task firstprivate(m/2,n,k,lda,ldb,ldc)
8        {
9          THNN_(mul_ker)(m/2,n,k,a,lda,b,ldb,c,ldc);
10       }
11       THNN_(mul_ker)( m/2,n,k,a+m/2,lda,b,ldb,c+m/2,ldc);
12     } else if(n > ITILE){ // Divide along n
13     } else { ... } // Divide along k
14   }
15 }
```

PERFORMANCE AFTER CONVOLUTION OPTIMIZATION

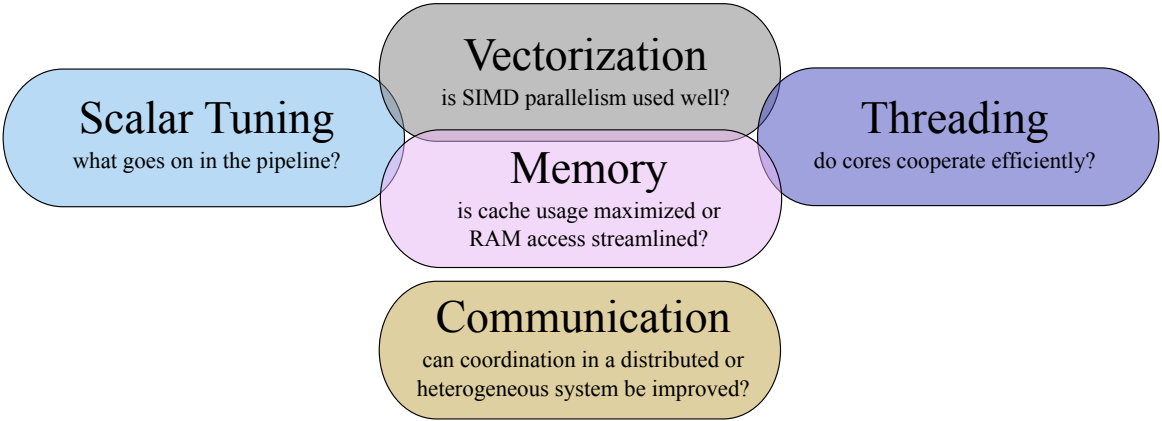


By Layer:

- ReLU: 1%
- Conv: 83%
- MaxPool: 13%
- Other: 3%

§5. LEARN MORE

IMPORTANCE OF CODE MODERNIZATION

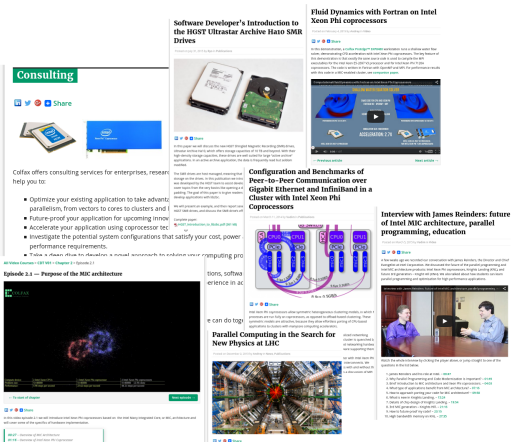


HOW SERIES

colfaxresearch.com/how-series

LEARN MORE

COLFAX RESEARCH



http://colfaxresearch.com/