

# Optimization of a DNN program on the CPU+MIC

University of Electronic Science and Technology of China

## Abstract

This article is a part of competition proposal of ASC(Asia Supercomputer Student Challenge). We analysis the DNN program, put forward different optimization methods and point their pros and cons. In the end we talk about our shortcomings.

## 1. Introduction

In the section a program based on a standalone hybrid CPU+MIC platform called DNN(deep neural network) should be parallelized to obtain better performance. The detailed configuration about is as follows:

## 2. Analysis of the serial program

First, a call graph is generated by using Google perfools, a open source performance profiler, to have a glance though it. Every square represents a function, and the bigger square is, the more time corresponding function cost.

Obviously, the hot spot is something about MKL. After googling and searching Intel document we know that MKL provides BLAS routines, which includes a serial function named `cblas_?gemm` to compute a matrix-matrix product with general matrices.

Item	Name	Configuration	Hosts
Server	Inspur NF5280M4 x 4	CPU : Intel Xeon E5-2680v3 x 2, 2.5Ghz, 12 cores	hostname:
		Memory: 16G x8, DDR4, 2133Mhz	mic1,
		Hard disk: 1T SATA x 1	mic2,
		Accelerator card: Intel XEON PHI-31S1P ( 57 cores, 1.1GHz, 1003GFlops, 8GB GDDR5 Memory )	mic3, mic4
Network		Infiniband+Ethernet	

Figure 1. Hardware configuration

Classification	Description	Installation path	Version
OS	GNU/Linux		RHEL 7.1
Compiler	Intel Composer XE Suites	/opt/intel/composer_xe_2015.0.090	2015.0.090
MKL	Intel MKL	/opt/intel/mkl/lib/intel64	
MPI	Intel MPI	/opt/intel/impi/5.0.1.035	5.0.1.035
PBS	Torque	/opt/tsce	3.0.5

Figure 2. Software configuration

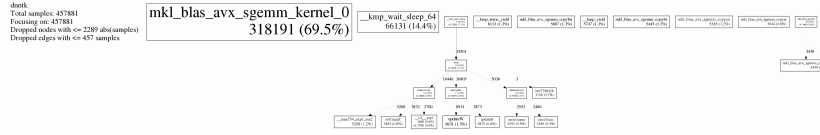


Figure 3. Google Perfools results

But giving that MKL function is well-optimized, we search for all position where `cblas_*sgemm` is called. Results show the usage of `cblas_*sgemm` appear in file `dnn_func.cpp`, more specifically, in three function:

- `extern "C" int dnnForward(NodeArg &nodeArg)`
- `extern "C" int dnnBackward(NodeArg &nodeArg)`
- `extern "C" int dnnUpdate(NodeArg &nodeArg)`

They call MKL function `cblas_sgemm` many times by `for` loop and cost almost 90% of all CPU time. So we guess that those function is what we may optimize, aka, hotspots. The report showed by Intel VTune, another profiler, proves our guess.

After a skim through the source code, a clear structure about the program is established. To simplify describe, original program could be rewritten in pseudocode:

Function Stack	CPU Time: Total											
	Effective Time by Utilization					Spin Time			Overhead Time			
	Idle	Poor	Ok	Ideal	Over	Imb...	Loc...	Other	Crea...	Sch...	Red...	Ato...
▼ Total	87.5%					12.0%	0.0%	0.5%	0.0%	0.0%	0.0%	0.0%
▼ _start	87.5%					1.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▼ _libc_start_main	87.5%					1.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▼ main	87.5%					1.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ dnnForward	31.6%					0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ dnnUpdate	29.7%					0.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ dnnBackward	26.1%					0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ FetchOneChunk	0.0%					0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ InitNodeConfig	0.0%					0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ __memcpy_ssse3_rep	0.0%					0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
▶ WriteWts	0.0%					0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Figure 4. Intel VTune top-down tree

```

1. GetInitFileConfig(cpuArg)
2. While FetchOneChunk(cpuArg, onChunk) do:
3.     While FetchOneBunch(oneChunk, nodeArg) do:
4.         dnnForward(nodeArg)
5.         dnnBackward(nodeArg)
6.         dnnUpate(nodeArg)
7. WriteWts(nodeArg, cpuArg)
8. UninitProgramConfig(cpuArg)

```

There are two nested loop before `dnn*()` processing function, and in each of those processing function many matrix-matrix product are executed. Whether those hotspots could be parallelized or not depends on data scale, dependency and so on. In the rest of this article some methods are considered and weighed their pros and cons.

### 3. Fine grain parallelism

In fine grain parallelism a thorough check is necessary. It's better to look through the whole top-down tree rendered by Intel VTune and to find out performance-critical loop. Attention should be given to the `dnn*` function series.

In function `dnnForward`, it's easy to observe there is a `for` loop calling `cblas_sgemm`, which nearly cost all CPU time consumed by this function. But there is some detail should be consider in before optimization.

#### 3.1. Matrix size

All `cblas_sgemm` is called like this:

```

cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, numN, numA[i], numA[i-1], \
            one, d_Y[i-1], numA[i-1], d_W[i], numA[i], one, d_Y[i], numA[i]);

```

The arguments `numN`, `numA[i]`, `numA[i-1]` indicating the size of the matrices:

- `d_Y[i-1]` is a `numN` row by `numA[i]` column matrix;
- `d_W[i]` is a `numN` row by `numA[i-1]` column matrix;
- `d_Y[i]` is a `numA[i-1]` row by `numA[i]` column matrix.

As we known the bigger matrix size is, the higher degree of MKL parallelism is. But in the DNN program, the size of matrix is decided by `bunchSize`, a constant integer, and `dnnLayerArr`, a constant integer array. The two integers are configured by specified file, and we are not allowed to modify it. For this reason there are no sufficiently large matrix to enable `auto offload model` to speed up DNN.<sup>[1]</sup>

### 3.2. Cycles index

## 4. Coarse grain parallelism

To impenment coarse grain parallelism we hope that each thread/process finish large subcomponents. To achieve this goal DNN program should be divided into (mostly) independent and similar proportions, and every proportion should be as large as possible.

## References

- [1] Noah Clemons. *Intel MKL Resource*. Intel, <https://software.intel.com/en-us/articles/recommendations-to-choose-the-right-mkl-usage-model-for-xeon-phi>. Mar. 2013.