Optimization of the DNN program on the CPU+MIC Platform

University of Electronic Secience and Technology of China

1. Preface

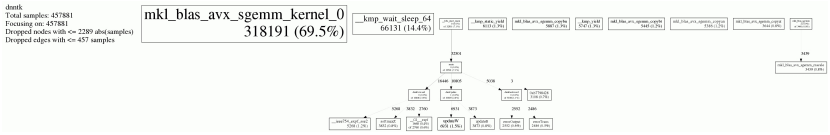
In the section we are required to optimize a `DNN(deep neural network)` program based on a standalone hybrid CPU+MIC platform. The detailed configuration is as follows:

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

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

2. Analysis of the serial program

First, we generate a call graph 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 routinues`, which includes a

serial funcition named `cblas_?gemm` to compute a matrix-matrix product with general matrices.

Then we search for `cblas_*gemm`, results show the usage of `cblas_*gemm` 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)`

So we guess that those function is what we should optimize, aka, hotspots. The report showed by`Intel VTune`, another profiler, proves our guess.

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

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

Then