Simple NN with CUDA/GPU

Bin ZHOU @ USTC

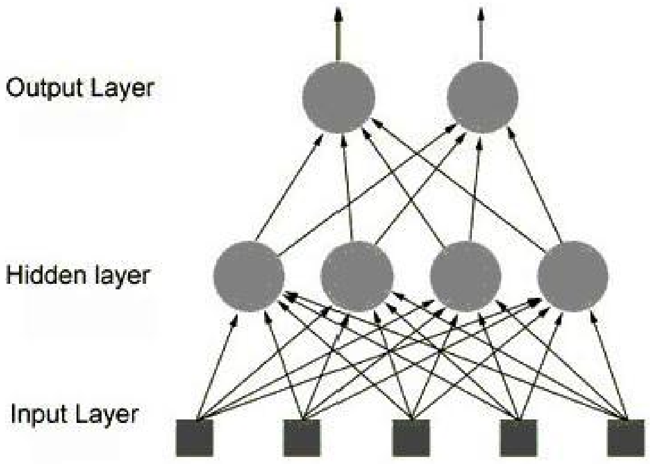
Jan.2015

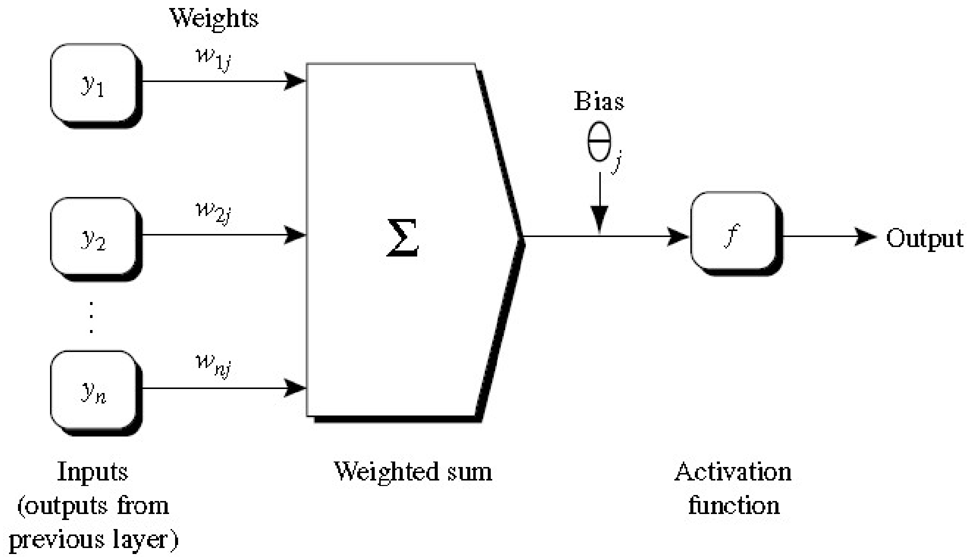
Very Simple digit recognition

- 3 layers: Simple BP network
  - 1 Input Layer, 1 hidden layer, 1 output layer
- Several Neurons
  - 784 (28*28) input, 100 hidden, 10 output
- Some configuration
  - Activation Function: Sigmoid function

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \quad (0 < f(x) < 1)$$

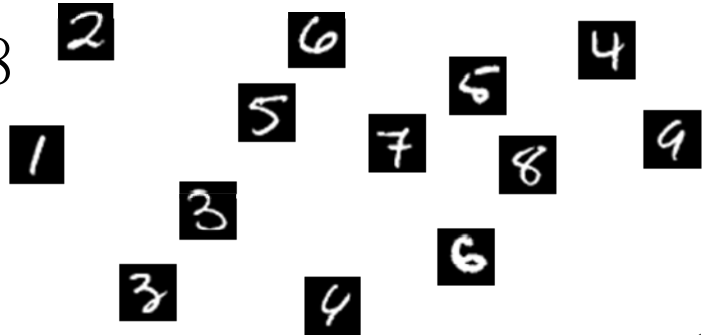
You' ve already known this very well





10 Digits to Recognize

- Training and testing samples are from
- MNIST , http://yann.lecun.com/exdb/mnist/
- Every pic is 28*28
- Totally 10000 pics



- We use 600 of each as training set and 200 as test set.

cess

Input & feature extraction??

- Raw Data?
- PCA for Images
- GPU??

Supervised Training

- BP with a lot of rounds
- Very Time Consuming!
- GPU ??

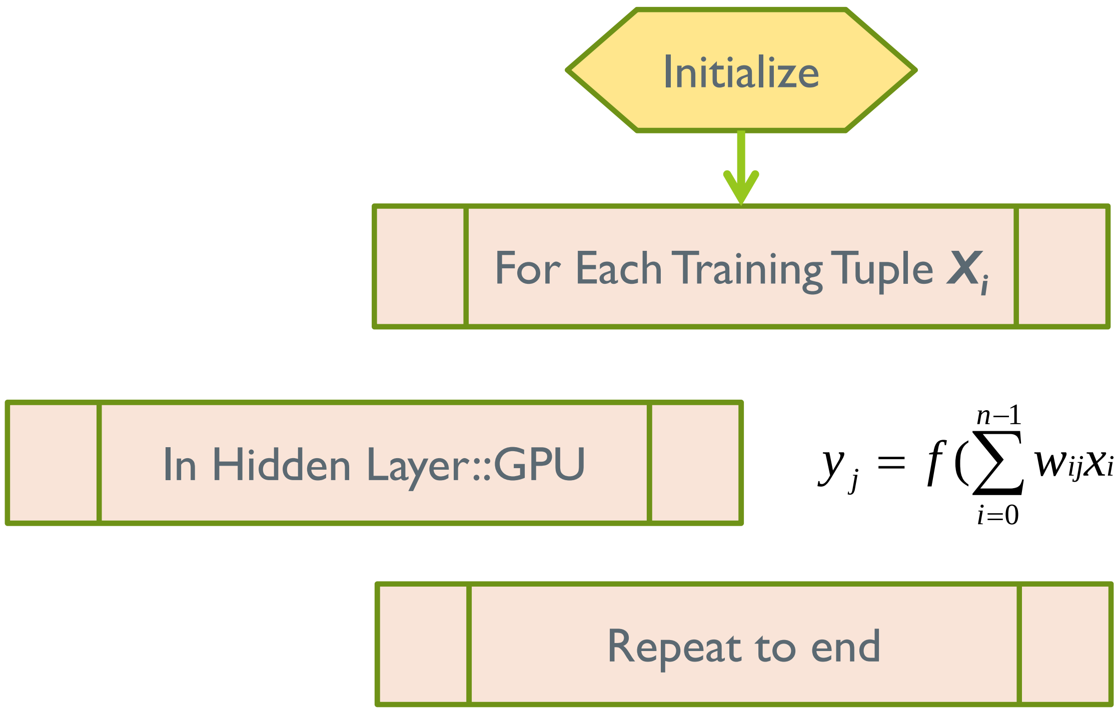
Classification

- Saved network for future usage
- GPU??

Training Process: GPU Accelerated

- Which Part?
- linear Algebra inside single iteration/Sample
- But not between iterations/Samples
- Dependency between iterations/Samples

Single Step Computation



$$y_j = f\left(\sum_{i=0}^{n-1} w_{ij} x_i - \theta_j\right) \qquad \Delta =$$

Algorithm View

Method:

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3) **for** each training tuple X in D {
(4) // Propagate the inputs forward:
(5) **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) **for** each unit j in the output layer
(12) $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
(13) **for** each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) **for** each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) **for** each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

GPU Implementation

- Initialize the network on GPU
  - Hidden Layer Nodes, Weight and Bias
  - Output Layer Nodes, Weight and Bias
  - Input dataset
- Prepare the data to GPU
  - Pack the batched images in CPU and then
  - Remember to do it all at once
- Then start the training for each sample

Parallelization Strategy

- Each thread is in charge of computing one output of the neuron
- Not limited by the thread number within a block
- Back propagation is also the same
- Very careful about the Memory Access Pattern!

Close look at the code

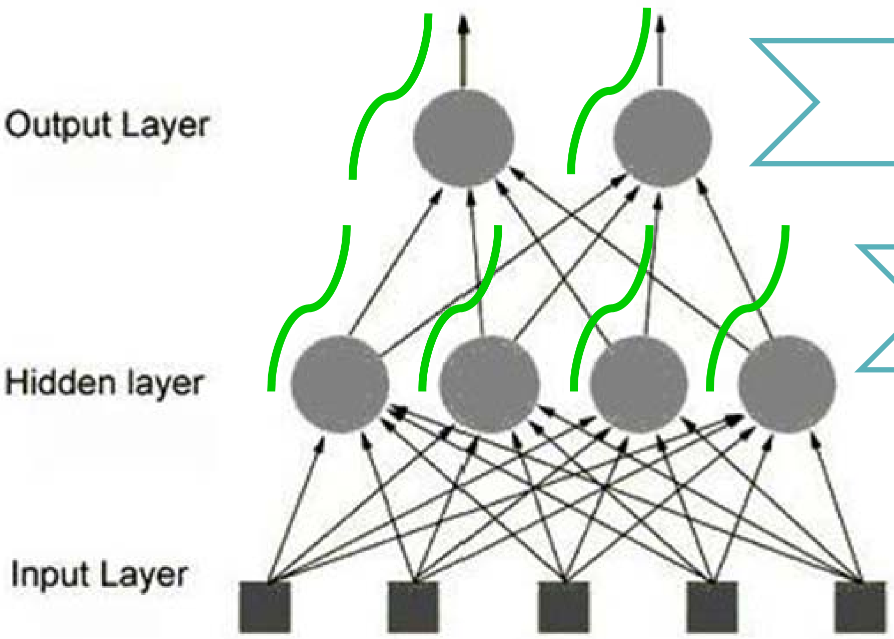
```
for(i=0;i<NON;i++)
        node0[i].Output=pic[i];

for(j=0;j<N1N;j++)
{
        node1[j].Input=node1[j].bias;
        for(i=0;i<NON;i++)
        {
                node1[j].Input+=w01[i][j]*node0[i].Output;
        }
        node1[j].Output=1.0/(1.0+exp(-node1[j].Input));
}
```

j is independent, which can be processed parallel

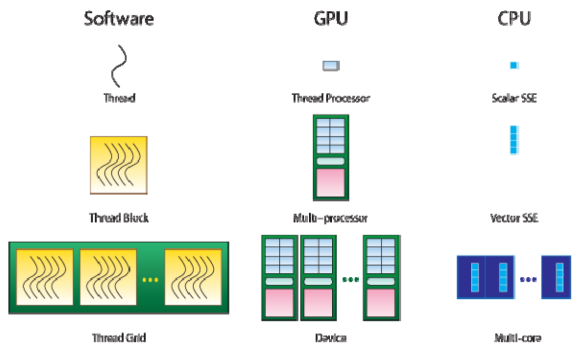
GPU Parallelization

Simple!



Layer 2 Every thread in charge of 1 neuron

Layer 1 Every thread in charge of 1 neuron



Close look at CUDA/GPU code

```
__global__ void kL0toL1(float *input, float *output, float *w, float *b)
{
        int nodeNum = threadIdx.x;
        int i = 0;
        float aTmp=0;
        if (nodeNum < N1N)
        {
                aTmp=b[nodeNum];
                for (i = 0; i< N0N; i++)
                        aTmp += *(w+i*100+nodeNum)*input[i];

                output[nodeNum] = 1.0/(1.0+exp(-aTmp));
        }
}
```

Every thread in charge of 1 neuron

Performance Consideration

- Memory Limited ? Instruction Limited?
- Memory Access Pattern?
  - Every thread will access w01[ ][ ] in a continuous way; Not so good.

Training Perf	i5 2.0G CPU 1 core	Kepler GPU 1 SM
1 image	57ms	1ms

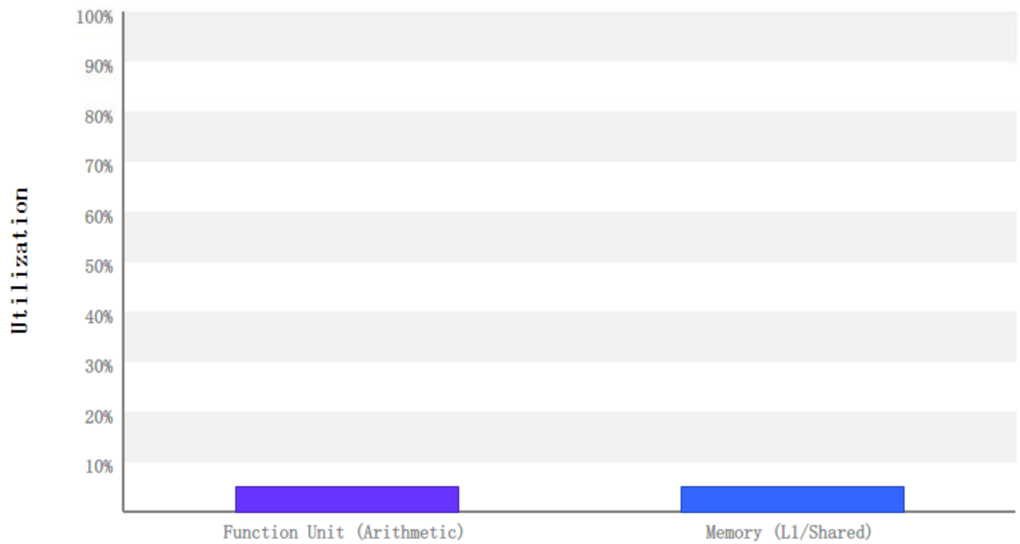
How to get a Better Solution?

- Memory Access Pattern is the first thing to deal with
- Put W01 into shared memory is a simple try
- Redesign the Memory Storage structure
- Or redesign the Algorithm to avoid the F function

More Detailed Analysis

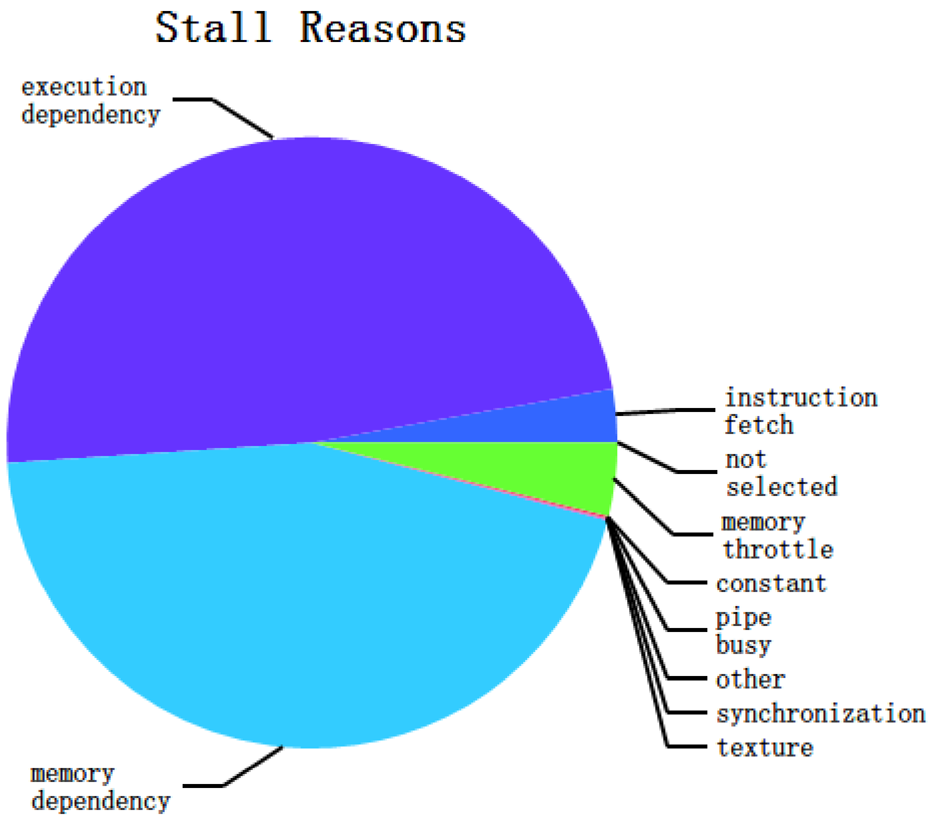
Performance is bounded by both Arithmetic and Memory latency. Too bad.

We have only 1 block, far away from filling the SM.



Kernel Latency

- Grid Size is too small to hide the latency



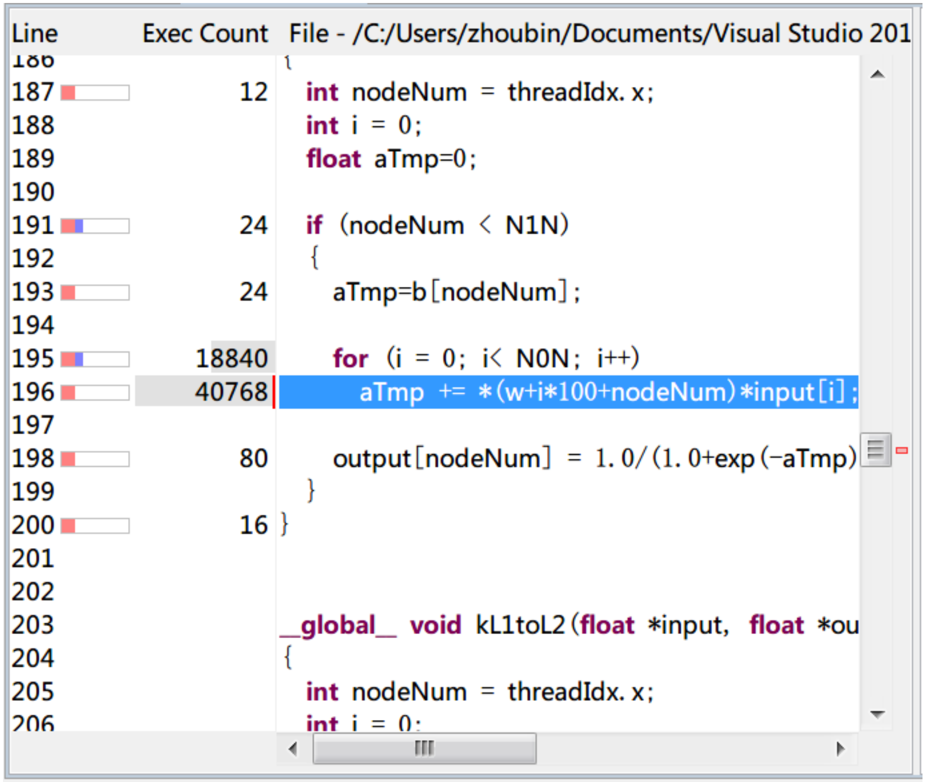
Register Analysis

Variable	Achieved	Theoretical	Device Limit	Grid Size: [1,1,1] (1 block)Block Size: [100,1,1] (100 threads)
Occupancy Per SM				
Active Blocks		10	16	
Active Warps	3.93	40	64	
Active Threads		1280	2048	
Occupancy	6.1%	62.5%	100%	
Warps				
Threads/Block		100	1024	
Warps/Block		4	32	
Block Limit		16	16	
Registers				
Registers/Thread		42	255	
Registers/Block		6144	65536	
Block Limit		10	16	
Shared Memory				
Shared Memory/Block		0	49152	
Block Limit			16	

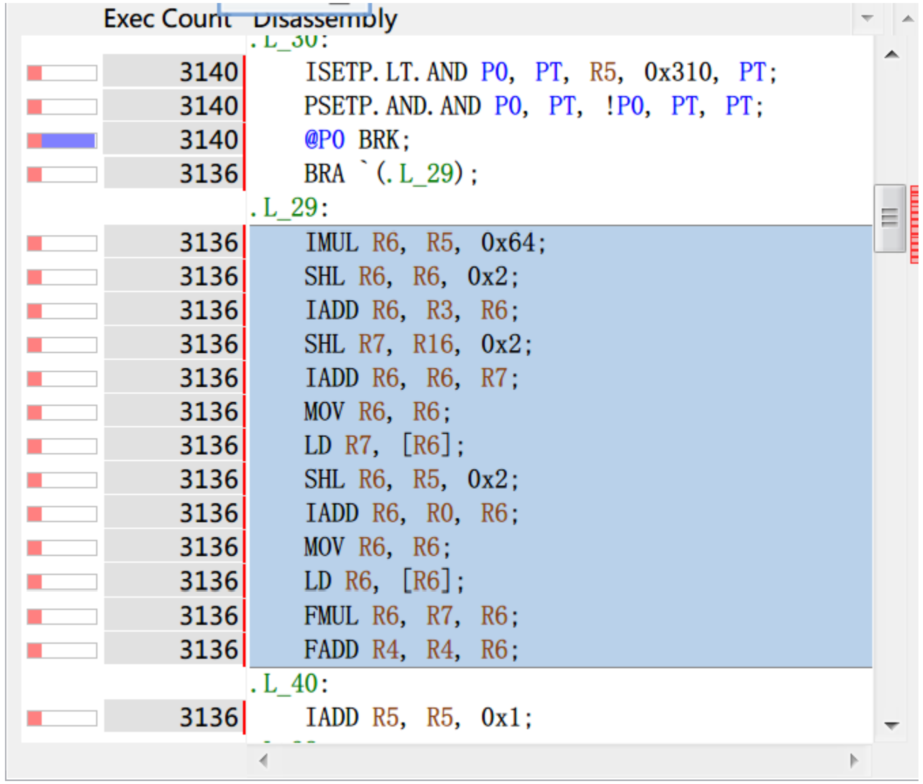
Kernel Memory

	Transactions	Bandwidth	Utilization
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	8334	857.238 MB/s	
Global Stores	4	767.657 kB/s	
Atomic	0	0 B/s	
L1/Shared Total	8338	858.005 MB/s	Idle Low Medium High Max
L2 Cache			
L1 Reads	14517	857.238 MB/s	
L1 Writes	13	767.657 kB/s	
Texture Reads	0	0 B/s	
Atomic	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Total	14530	858.005 MB/s	Idle Low Medium High Max
Texture Cache			
Reads	0	0 B/s	Idle Low Medium High Max
Device Memory			
Reads	8930	527.322 MB/s	
Writes	14	826.708 kB/s	
Total	8944	528.149 MB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen2 x4, 5 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	1	59.05 kB/s	Idle Low Medium High Max

Target



Line Exec Count File - /C:/Users/zhoubin/Documents/Visual Studio 201
187 12 int nodeNum = threadIdx.x;
188 int i = 0;
189 float aTmp=0;
191 24 if (nodeNum < N1N)
193 24 aTmp=b[nodeNum];
195 18840 for (i = 0; i< N0N; i++)
196 40768 aTmp += *(w+i*100+nodeNum)*input[i];
198 80 output[nodeNum] = 1.0/(1.0+exp(-aTmp));
200 16 }
203 __global__ void kL1toL2(float *input, float *ou
205 int nodeNum = threadIdx.x;



Exec Count Disassembly
3140 ISETP.LT.AND P0, PT, R5, 0x310, PT;
3140 PSETP.AND.AND P0, PT, !P0, PT, PT;
3140 @P0 BRK;
3136 BRA `(.L_29);
.L_29:
3136 IMUL R6, R5, 0x64;
3136 SHL R6, R6, 0x2;
3136 IADD R6, R3, R6;
3136 SHL R7, R16, 0x2;
3136 IADD R6, R6, R7;
3136 MOV R6, R6;
3136 LD R7, [R6];
3136 SHL R6, R5, 0x2;
3136 IADD R6, R0, R6;
3136 MOV R6, R6;
3136 LD R6, [R6];
3136 FMUL R6, R7, R6;
3136 FADD R4, R4, R6;
.L_40:
3136 IADD R5, R5, 0x1;