N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Code Optimization

For optimization simplification we have used our OneHiddenClassification.cu program. This code constructs a one hidden layer neural network (32754 input nodes 128 hidden layer nodes, 4 output nodes) for our text classification task. By using Tensorflow (A computation library) we have confirmed that our data can be correctly classified with high accuracy with only one hidden layer.

Our optimization steps of GPU code consist of following steps:

Combination of some sequential operations(functions) into one function to reduce data retrieval overhead. These consecutive functions use their input output sequentially. Two or more times the data is retrieved from global memory. Instead we can apply these operations in one retrieval.

Some kernel executions have dependencies but some don't. We can run non-dependent kernels at the same because they have no input output dependency to each other. By executing these kinds of kernels at the same time we can gain more parallelism on GPU and consequently speed gain.

Unrolling technique: In first implementation our kernels is responsible for only one data point. By introducing more data point calculation, we can increase the total operations on fly on GPU resulting in more optimization and speed gain.

Shared Memory usage with padding: In first implementation our kernels retrieve all data directly from global memory. Most of our kernels obtain data and perform one operation on it and save it back to global memory. The main consideration here is to make all these global data load and store operations aligned and coalesced. Every thread in one warp is executed at the same time. These 32 threads request data from the global memory. Ideally when the data reads are coalesced and aligned, this demand from one warp can be fulfilled with only one data transaction. If the data is not aligned the data retrieval operation is performed more than one transaction, which results in performance degrade. Most of our kernel performs coalesced and aligned data reading. But transpose and matrix multiplication operations include column read which results uncoalesced data retrieval transactions. One remedy to this problematic reading is to load data to first shared memory. Shared memory is a on chip memory which is much more faster than global memory in reading and storing operations. So, reading the data from global memory in coalesced and aligned manner and storing it to shared memory saves us from performing more data transactions than needed. We are eliminating uncoalesced and aligned data requests. Shared memory has a special data reading mechanism. The shared memory is divided into banks. If two requests correspond to the same bank, the back confliction occurs and shared memory services this two requests sequentially. When we upload the data to shared memory and perform transpose operation or matrix multiplication, here we face the block confliction problem. Because kernels will try to read or write column vectors which are mapped upon the same banks. Of course, bank confliction overhead is less than the global memory's unaligned and uncoalesced memory accesses. To fix this confliction problem a padding technique is used. This technique simply adds a blank column vector to original data to shift overlapping banks and any accesses to the same bank with padding will not cause bank confliction therefore the store or load operations will be performed in optimum manner.

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Texture memory cache: Texture memory has a cache optimized for 2D spatial data. CUDA framework allows programmers to use this cache while accessing the global data. Global data can be cached on this cache memory. We can use it like a L1 cache. Before going to directly to global memory the data load request will be directed to texture cache, it the requested data rest there, the data will be retrieved from here if not cache miss occurs and from global data the needed data is loaded to cache. By using this cache, the memory request number from global memory could be lessened. Because the global memory accesses are time consuming, we expect some gain from usage of this cache.

Batch Size and Thread Block number: Different thread block numbers effect the GPU utilization rate. Finding correct thread block number requires some trail and error process. Batch size of the training data defines sample number that is used in forward and back propagation. Sending more number of samples to training leads early finishing of whole data pass.

We have test these optimization techniques by applying every one of them to unoptimized code. Only one feature is implemented at a time.

Combination of some sequential operations

```
MatrixAdd<<<grid, block>>>(output_2, output_2, bias_result_2);
cudaDeviceSynchronize();
Exponential<<<grid, block>>>(output_2, output_2);
cudaDeviceSynchronize();
```

These two functions are fused into one function.

```
AddandExponential<<<grid, block>>>(output_2, output_2, bias_result_2);
cudaDeviceSynchronize();
```

Original forms:

```
__global__ void MatrixAdd(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{

    int tx = blockIdx.x*blockDim.x+ threadIdx.x;
    int ty = blockIdx.y*blockDim.y + threadIdx.y;
    int tid = ty*vec1->width+tx;

    if(tid < vec1->width*vec1->height)
    {
```

```
		result->data[tid] = vec1->data[tid] + vec2->data[tid];
	}


}

__global__ void Exponential(Vector2D * result, Vector2D * vec1)
{


	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = exp(vec1->data[tid]);
	}




}
```

Combined form:

```
__global__ void AddandExponential(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = exp(vec1->data[tid] + vec2->data[tid]);
	}

}

	MatrixPairwiseProduct<<<grid6, block>>>(layer_1_error, layer_1_error, output_1);
	cudaDeviceSynchronize();
	ScalarMinusVector2D<<<grid6, block>>>(scalar_minus, 1.0, output_1);
	cudaDeviceSynchronize();
	MatrixPairwiseProduct<<<grid6, block>>>(layer_1_error, layer_1_error, scalar_minus);
	cudaDeviceSynchronize();
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

These three functions are fused into one.

```
LayerErrorCalculate<<<grid6, block>>>(layer_1_error, layer_1_error, output_1);
```

Original forms:

```
__global__ void MatrixPairwiseProduct(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] * vec2->data[tid];
	}
}



__global__ void ScalarMinusVector2D(Vector2D * result, float value, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;


	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = 1-vec1->data[tid];
	}


}
```

Combined form:

```
//Combination of matrixpairwise-Scalarminus-matrixpairwise in backpropagte....
__global__ void LayerErrorCalculate(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{

	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid + < vec1->width*vec1->height)
```

```
        {
            result->data[tid] = vec1->data[tid] * vec2->data[tid]*(1-vec2->data[tid]) ;
        }

}


        dim3 grid10((OUTPUT_NODE_COUNT+block.x-1)/block.x,
        (HIDDEN_LAYER_NODE_COUNT+block.y-1)/block.y);
        ScalarMatrixProduct<<<grid10, block>>>(w2_update, learning_rate, w2_update);
        cudaDeviceSynchronize();
        //Apply w2 update
        MatrixAdd<<<grid10, block>>>(w2, w2, w2_update);
        cudaDeviceSynchronize();
```

These two functions are fused into one function.

```
        dim3 grid10((OUTPUT_NODE_COUNT+block.x-1)/block.x,
        (HIDDEN_LAYER_NODE_COUNT+block.y-1)/block.y);
        ApplyWeightChange<<<grid10, block>>>(w2, learning_rate, w2_update);
```

Original forms:

```
__global__ void ScalarMatrixProduct(Vector2D * result, float scalar, Vector2D * vec1)
{
        int tx = blockIdx.x*blockDim.x+ threadIdx.x;
        int ty = blockIdx.y*blockDim.y + threadIdx.y;
        int tid = ty*vec1->width+tx;

        if(tid < vec1->width*vec1->height)
        {
                result->data[tid] = scalar*vec1->data[tid];
        }
}

__global__ void MatrixAdd(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
        int tx = blockIdx.x*blockDim.x+ threadIdx.x;
        int ty = blockIdx.y*blockDim.y + threadIdx.y;
        int tid = ty*vec1->width+tx;

        if(tid < vec1->width*vec1->height)
        {
                result->data[tid] = vec1->data[tid] + vec2->data[tid];
        }
}
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Combined form:

```
__global__ void ApplyWeightChange(Vector2D * result, float learning_rate, Vector2D * source)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*source->width+tx;

	if(tid < source->width*source->height)
	{
		result->data[tid] += learning_rate*source->data[tid];	}}


			dim3 gridd((OUTPUT_NODE_COUNT+block.x-1)/block.x,
			(batch_size+block.y-1)/block.y);
			Log2D<<<gridd, block>>>(output_2, output_2);
			cudaDeviceSynchronize();
			MatrixPairwiseProduct<<<gridd, block>>>(layer_2_error, batch_label, o
			utput_2);
			cudaDeviceSynchronize();

		dim3 gridd((OUTPUT_NODE_COUNT+block.x-1)/block.x, (batch_size+block.y-1
		)/block.y);
		calculateCrossEntropyLoss<<<gridd, block>>>(layer_2_error, batch_label,
		output_2);
```

Original forms:

```
__global__ void Log2D(Vector2D * result, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	float val;
	if(tid < vec1->width*vec1->height)
	{	val = log(vec1->data[tid]);
		result->data[tid] = val;
	}
}


__global__ void MatrixPairwiseProduct(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] * vec2->data[tid];
```

```
	}
}
```

Combined form:

```
__global__ void calculateCrossEntropyLoss(Vector2D * __restrict__ result, Vector2D * __restrict__ vec1, Vector2D * __restrict__ vec2)
{	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
		if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] * log(vec2->data[tid]);
	}
}
```

Unrolling Technique

Original form:

```
__global__ void MatrixAdd(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] + vec2->data[tid];
	}
	}
```

Unrolled form:

```
__global__ void MatrixAdd(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid +blockDim.x*3< vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] + vec2->data[tid];
		result->data[tid+blockDim.x] = vec1->data[tid+blockDim.x] + vec2->data[tid+blockDim.x];
		result->data[tid+2*blockDim.x] = vec1->data[tid+2*blockDim.x] + vec2->data[tid+2*blockDim.x];
		result->data[tid+3*blockDim.x] = vec1->data[tid+3*blockDim.x] + vec2->data[tid+3*blockDim.x];
```

```
	}

}
```

Original form:

```
__global__ void MatrixSubtract(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] - vec2->data[tid];

	}
}
```

Unrolled form:

```
__global__ void MatrixSubtract(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid +3*blockDim.x< vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] - vec2->data[tid];
		result->data[tid+blockDim.x] = vec1->data[tid] - vec2->data[tid+blockDim.x];
		result->data[tid+2*blockDim.x] = vec1->data[tid+2*blockDim.x] - vec2-
>data[tid+2*blockDim.x];
		result->data[tid+2*blockDim.x] = vec1->data[tid+3*blockDim.x] - vec2-
>data[tid+3*blockDim.x];
	}


}
```

Original form:

```
__global__ void TransposeVector2D(Vector2D * res, Vector2D * m1)
{
	int thx = blockIdx.x*blockDim.x+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;
	int tid = thx + thy*m1->width;
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

```
	if(tid < m1->width*m1->height)
	{
	res->data[thy+thx*m1->height] = m1->data[tid] ;
	}

}
```

Unrolled form:

```
__global__ void TransposeVector2DUnroll4(Vector2D * res, Vector2D * m1)
{

	int thx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;
	int tid = thx + thy*m1->width;

	if(tid +3*blockDim.x< m1->width*m1->height)
	{

	res->data[thy+thx*m1->height] = m1->data[tid] ;
	res->data[thy+(thx+blockDim.x)*m1->height] = m1->data[tid+blockDim.x] ;
	res->data[thy+(thx+2*blockDim.x)*m1->height] = m1->data[tid+2*blockDim.x] ;
	res->data[thy+(thx+3*blockDim.x)*m1->height] = m1->data[tid+3*blockDim.x] ;

	}

}
```

Original form:

```
__global__ void MatrixProduct(Vector2D * result, Vector2D * m1, Vector2D * m2)
{
	int thx = blockIdx.x*blockDim.x+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;

	if(thx < result->width && thy < result->height)
	{
		float toplam = 0;
		for(int h = 0; h < m1->width; h++)
		{
			toplam += m1->data[thy*m1->width+h] * m2->data[h*m2->width+thx];
		}
	result->data[thy*result->width + thx] = toplam;
	}

}
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Unrolled form:

```
__global__ void MatrixProduct(Vector2D * result, Vector2D * m1, Vector2D * m2)
{
	int thx = blockIdx.x*blockDim.x+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;

	if(thx < result->width && thy < result->height)
	{
		float toplam = 0;

		#pragma unroll 4
		for(int h = 0; h < m1->width; h++)
		{
			toplam += m1->data[thy*m1->width+h] * m2->data[h*m2->width+thx];

		}
	result->data[thy*result->width + thx] = toplam;

	}

}
```

Original form:

```
__global__ void ScalarMinusVector2D(Vector2D * result, float value, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = 1-vec1->data[tid];
	}

}
```

Unrolled form:

```
__global__ void ScalarMinusVector2D(Vector2D * result, float value, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;



	if(tid +3*blockDim.x< vec1->width*vec1->height)
```

```
	{
		result->data[tid] = 1-vec1->data[tid];
		result->data[tid+blockDim.x] = 1-vec1->data[tid+blockDim.x];
		result->data[tid+2*blockDim.x] = 1-vec1->data[tid+2*blockDim.x];
		result->data[tid+3*blockDim.x] = 1-vec1->data[tid+3*blockDim.x];
	}

}
```

Original form:

```
__global__ void ScalarMatrixProduct(Vector2D * result, float scalar, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = scalar*vec1->data[tid];
	}
}
```

Unrolled form:

```
__global__ void ScalarMatrixProduct(Vector2D * result, float scalar, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid +3*blockDim.x< vec1->width*vec1->height)
	{
		result->data[tid] = scalar*vec1->data[tid];
		result->data[tid+blockDim.x] = scalar*vec1->data[tid+blockDim.x];
		result->data[tid+2*blockDim.x] = scalar*vec1->data[tid+2*blockDim.x];
		result->data[tid+3*blockDim.x] = scalar*vec1->data[tid+3*blockDim.x];
	}
}
```

Original form:

```
__global__ void MatrixPairwiseProduct(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] * vec2->data[tid];
```

```
	}
}
```

Unrolled form:

```
__global__ void MatrixPairwiseProduct(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid +3*blockDim.x< vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] * vec2->data[tid];
		result->data[tid+blockDim.x] = vec1->data[tid+blockDim.x] * vec2-
>data[tid+blockDim.x];
		result->data[tid+2*blockDim.x] = vec1->data[tid+2*blockDim.x] * vec2-
>data[tid+2*blockDim.x];
		result->data[tid+3*blockDim.x] = vec1->data[tid+3*blockDim.x] * vec2-
>data[tid+3*blockDim.x];
	}


}
```

Original form:

```
__global__ void Softmax(Vector2D * result, Vector2D * vec1)
{
	int tid = blockIdx.y*blockDim.y + threadIdx.y;

	if(tid < vec1->height)
	{
		float toplam = 0;
		for(int a = 0; a < vec1->width;a++)
		{
			toplam += vec1->data[a+tid*vec1->width];

		}
		for(int a = 0; a < vec1->width;a++)
		{
			result->data[a+tid*vec1->width] = vec1->data[a+tid*vec1->width]/toplam;

		}
	}
}
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Unrolled form:

```
__global__ void Softmax(Vector2D * result, Vector2D * vec1)
{
	int tid = blockIdx.y*blockDim.y + threadIdx.y;

	if(tid < vec1->height)
	{
		float toplam = 0;
		#pragma unroll OUTPUT_NODE_COUNT
		for(int a = 0; a < vec1->width;a++)
		{
			toplam += vec1->data[a+tid*vec1->width];

		}
		#pragma unroll OUTPUT_NODE_COUNT
		for(int a = 0; a < vec1->width;a++)
		{
			result->data[a+tid*vec1->width] = vec1->data[a+tid*vec1->width]/toplam;

		}
	}
}
```

Original form:

```
__global__ void Sum2D(Vector2D * vec)
{
	int tid = threadIdx.y;
	float val = 0;
	int width = vec->width;
	for(int a = 0; a < width; a++)
	{
		val += vec->data[a+tid*width];
	}
	error_sum[tid] = val;
}
```

Unrolled form:

```
__global__ void Sum2D(Vector2D * vec)
{
	int tid = threadIdx.y;
	float val = 0;
	int width = vec->width;
	#pragma unroll 4
	for(int a = 0; a < width; a++)
	{
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

```
            val += vec->data[a+tid*width];
        }
        error_sum[tid] = val;
}
```

Original form:

```
__global__ void ArgMax2D(Vector2D * vec1)
{
        int tid = blockIdx.y*blockDim.y + threadIdx.y;
        if(tid < vec1->height)
        {
                float max = -100000;
                int max_index = 0;
                for(int a = 0; a < vec1->width;a++)
                {
                        if(vec1->data[tid*vec1->width+a]>max)
                        {
                                max = vec1->data[tid*vec1->width+a];
                                max_index = a;
                        }}
        arg_max_result[tid] = max_index;
        }
}
```

Unrolled form:

```
__global__ void ArgMax2D(Vector2D * vec1)
{
        int tid = blockIdx.y*blockDim.y + threadIdx.y;
        if(tid < vec1->height)
        {
                float max = -100000;
                int max_index = 0;
                #pragma unroll 4
                for(int a = 0; a < vec1->width;a++)
                {
                        if(vec1->data[tid*vec1->width+a]>max)
                        {
                                max = vec1->data[tid*vec1->width+a];
                                max_index = a;
                        }
                }
                arg_max_result[tid] = max_index;
        }
}
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Original form:

```
__global__ void Log2D(Vector2D * result, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = log(vec1->data[tid]);
	}
}
```

Unrolled form:

```
__global__ void Log2D(Vector2D * result, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid + 3*blockDim.x< vec1->width*vec1->height)
	{
		result->data[tid] = log(vec1->data[tid]);
		result->data[tid+blockDim.x] = log(vec1->data[tid+blockDim.x]);
		result->data[tid+2*blockDim.x] = log(vec1->data[tid+2*blockDim.x]);
		result->data[tid+3*blockDim.x] = log(vec1->data[tid+3*blockDim.x]);
	}
}
```

Original form:

```
__global__ void Exponential(Vector2D * result, Vector2D * vec1)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = exp(vec1->data[tid]);
	}
}
```

Unrolled form:

```
__global__ void Exponential(Vector2D * result, Vector2D * vec1)
{	int tx = blockIdx.x*blockDim.x*4*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
```

```
        int tid = ty*vec1->width+tx;
        if(tid +3*blockDim.x< vec1->width*vec1->height)
        {
                result->data[tid] = exp(vec1->data[tid]);
                result->data[tid+blockDim.x] = exp(vec1->data[tid+blockDim.x]);
                result->data[tid+2*blockDim.x] = exp(vec1->data[tid+2*blockDim.x]);
                result->data[tid+3*blockDim.x] = exp(vec1->data[tid+3*blockDim.x]);
        }

}
```

Original form:

```
__global__ void Sigmoid(Vector2D * result, Vector2D * vec1)
{
        int tx = blockIdx.x*blockDim.x+ threadIdx.x;
        int ty = blockIdx.y*blockDim.y + threadIdx.y;
        int tid = ty*vec1->width+tx;
        if(tid < vec1->width*vec1->height)
        {
                result->data[tid] = 1.0/(1.0 + exp(-(vec1->data[tid])));
        }
}
```

Unrolled form:

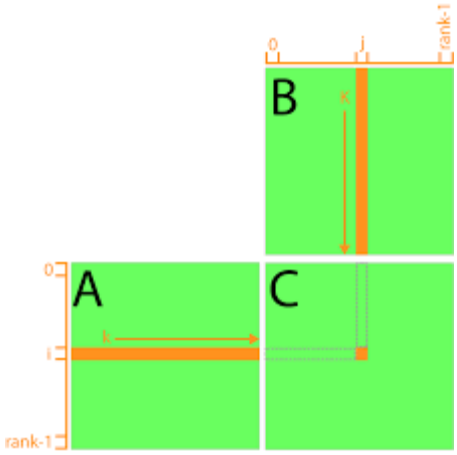
```
__global__ void Sigmoid(Vector2D * result, Vector2D * vec1)
{
        int tx = blockIdx.x*blockDim.x*4+ threadIdx.x;
        int ty = blockIdx.y*blockDim.y + threadIdx.y;
        int tid = ty*vec1->width+tx;
        if(tid +3*blockDim.x< vec1->width*vec1->height)
        {
                result->data[tid] = 1.0/(1.0 + exp(-(vec1->data[tid])));
                result->data[tid+blockDim.x] = 1.0/(1.0 + exp(-(vec1->data[tid+blockDim.x])));
                result->data[tid+2*blockDim.x] = 1.0/(1.0 + exp(-(vec1->data[tid+2*blockDim.x])));
                result->data[tid+3*blockDim.x] = 1.0/(1.0 + exp(-(vec1->data[tid+3*blockDim.x])));
        }
}
```

Shared Memory with Padding

Without shared memory:
Each thread of the kernel calculates the multiplication of one row vector of first matrix with one column vector of second matrix. Second matrix data reading from global memory is not coalesced. It brings more transaction for data retrieval overhead.

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL



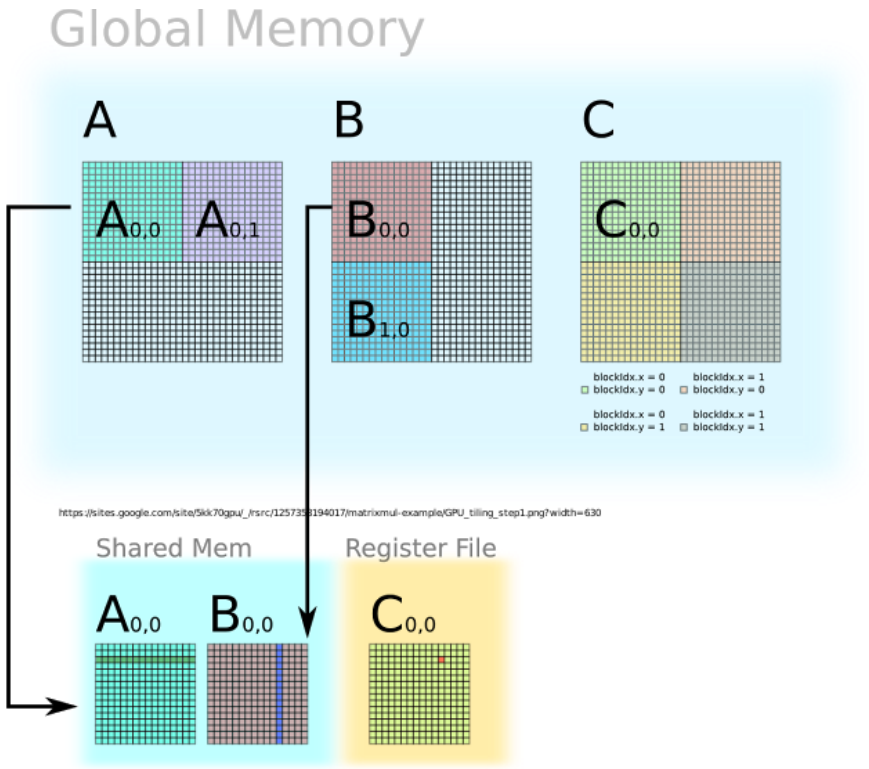
(https://www.3dgep.com/cuda-memory-model/)

```
__global__ void MatrixProduct(Vector2D * result, Vector2D * m1, Vector2D * m2)
{
	int thx = blockIdx.x*blockDim.x+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;

	if(thx < result->width && thy < result->height)
	{
		float toplam = 0;
		for(int h = 0; h < m1->width; h++)
		{
			toplam += m1->data[thy*m1->width+h] * m2->data[h*m2->width+thx];
		}
	result->data[thy*result->width + thx] = toplam;

	}
}
```

With shared memory:
Each matrix in matrix multiplication is divided into blocks(tiles). Thread blocks are responsible for each part. Each thread block uses shared memory with tile size. First each block reads matrix data from global to shared memory for each tile. Because the data transactions fall into the warp boundaries, of thread block the transactions of reading data from global to shared are aligned and coalesced. While reading the column vector of tile in B matrices normally bank conflictions occur. To eliminate that the required padding of shared memory is applied.

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL



(http://www.cstechera.com/2016/03/tiled-matrix-multiplication-using-shared-memory-in-cuda.html)

```
__global__ void MatrixProductShared(  Vector2D * result, Vector2D * m1, Vector2D * m2 )//float *A, float *B, float *C ) {
 {
  __shared__ float A_tile[TILE_HEIGHT][TILE_WIDTH];
   __shared__ float B_tile[TILE_HEIGHT][TILE_WIDTH+1];
   int numARows = m1->height, numAColumns= m1->width, numBRows = m2->height, numBColumns = m2->width, numCRows = result->height, numCColumns = m2->width;
  float * A = m1->data, * B = m2->data, * C = result->data;
  float sum = 0.0;
  // tx for thread_x or tile_x
  int tx = threadIdx.x; int ty = threadIdx.y;
  // cx for top left corner of tile in C
  int cx = blockIdx.x * blockDim.x; int cy = blockIdx.y * blockDim.y;
  // Cx for cell coordinates in C
  int Cx = cx + tx; int Cy = cy + ty;

  int total_tiles = (numAColumns + TILE_WIDTH - 1) / TILE_WIDTH;

  for (int tile_idx = 0; tile_idx < total_tiles; tile_idx++) {
    // the corresponding tiles' top left corners are:
    // for A: row = blockIdx.y * blockDim.y, col = tile_idx * TILE_WIDTH
    // for B: row = tile_idx * TILE_WIDTH, col = blockIdx.x * blockDim.x
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

```
    // loading tiles
   int Ax = tile_idx * TILE_WIDTH + tx; int Ay = cy + ty;
   int Bx = cx + tx; int By = tile_idx * TILE_WIDTH + ty;
  if (Ax < numAColumns && Ay < numARows) {
    A_tile[ty][tx] = A[Ay * numAColumns + Ax];
   }
   else {
    A_tile[ty][tx] = 0.0;
   }
   if (Bx < numBColumns && By < numBRows) {
    B_tile[ty][tx] = B[By * numBColumns + Bx];
   }
   else {
    B_tile[ty][tx] = 0.0;
   }
   __syncthreads();
   // multiplying tiles
   for (int i = 0; i < TILE_WIDTH; i++) {
    sum += A_tile[ty][i] * B_tile[i][tx];
   }
   __syncthreads();
  }
  // saving result (discarded if we're in the wrong thread)
  if (Cx < numCColumns && Cy < numCRows) {
   C[Cy * numCColumns + Cx] = sum;
  }
}
```

Without shared memory:
Kernel reads one data point from the source matrix and stores it to target matrix. The data readings are aligned and coalesced. But storing operation is performed in column vector manner. Therefore, more storing operations will be performed writing data rather than only one write transaction.

```
__global__ void TransposeVector2D(Vector2D * res, Vector2D * m1)
{
        int thx = blockIdx.x*blockDim.x+ threadIdx.x;
        int thy = blockIdx.y*blockDim.y+threadIdx.y;
        int tid = thx + thy*m1->width;
        if(tid < m1->width*m1->height)
        {
        res->data[thy+thx*m1->height] = m1->data[tid] ;
        }
}
```

With Shared Memory:

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

First matrix data is brought to shared memory. All readings are coalesced and aligned. The transpose operation is performed on shared memory. Each thread block brings it data to shared memory and writes to target location in shared memory. This writing operations cause bank confliction. To eliminate this flaw, on the target shared memory matrix padding technique is applied. After each block performs transpose operation, the storing this transposed data on shared memory to global memory is carried out in coalesced and aligned manner. So, we eliminate uncoalesced memory accesses by using shared memory.

```
__global__ void TransposeVector2DShared(Vector2D * res, Vector2D * m1)
{
	int thx = blockIdx.x*blockDim.x+ threadIdx.x;
	int thy = blockIdx.y*blockDim.y+threadIdx.y;

	int tid = thx + thy*m1->width;

	 __shared__ float ordered_data[BLOCK_Y][BLOCK_X+1];
	 __shared__ float transposed_data[BLOCK_Y][BLOCK_X+1];

	int j = threadIdx.x+blockDim.x*blockIdx.y;

	int k = threadIdx.y + blockDim.y*blockIdx.x;

	int target = j + res->width*k;

	if(tid < m1->width*m1->height)
	{
	//padded
	ordered_data[threadIdx.y][threadIdx.x] = m1->data[tid] ;

	}
	__syncthreads();

	if(thx < m1->width && thy< m1->height)
	{

	transposed_data[threadIdx.x][threadIdx.y]  = ordered_data[threadIdx.y][threadIdx.x];

	}
__syncthreads();

if(thx < m1->width && thy< m1->height)
	{
	res->data [target] = transposed_data[threadIdx.y][threadIdx.x] ;

	}}
```

Texture memory cache

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

No texture cache is used:

```
__global__ void MatrixAdd(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;

	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] + vec2->data[tid];
	}
}
```

Texture cache is used:

```
__global__ void MatrixAdd(Vector2D * __restrict__ result, Vector2D * __restrict__  vec1, Vector2D *
__restrict__  vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] + vec2->data[tid];
	}
}
```

Texture memory is not used:

```
__global__ void MatrixSubtract(Vector2D * result, Vector2D * vec1, Vector2D * vec2)
{	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
	{
		result->data[tid] = vec1->data[tid] - vec2->data[tid];
	}
}
```

Texture memory is used:

```
__global__ void MatrixSubtract(Vector2D * __restrict__ result, Vector2D * __restrict__  vec1,
Vector2D *  __restrict__  vec2)
{
	int tx = blockIdx.x*blockDim.x+ threadIdx.x;
	int ty = blockIdx.y*blockDim.y + threadIdx.y;
	int tid = ty*vec1->width+tx;
	if(tid < vec1->width*vec1->height)
```

```
{
    result->data[tid] = vec1->data[tid] – vec2→data[tid];}}
```

The remaining functions are in the same form.

Comparison Results

For comparison, we have tested every code with 1 iteration 9 batch training 10 batch testing. During this execution the metrics of nvprof have been obtained. To find out whether the technique improved execution time upon the unoptimized code, we have run all code with 100 iteration whole training data and whole testing data. By doing so we have been able to see the effect of change in long run.

Unoptimized OneHiddenClassification Program Results

First program execution time : 212.580296 second program execution time : 208.290466
Average execution time : 210,435381

Kernel: Log2D(Vector2D*, Vector2D*)		Min	Max	Avg
gld_transactions	Global Load Transactions	3394	3394	3394
gst_transactions	Global Store Transactions	128	128	128
gld_efficiency	Global Memory Load Efficiency	53.79%	53.79%	53.79%
gst_efficiency	Global Memory Store Efficiency	89.06%	89.06%	89.06%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	611	611	611
global_store_requests	Total number of global store requests from Multiprocessor	118	118	118
sm_efficiency	Multiprocessor Activity	7.85%	8.44%	8.04%
achieved_occupancy	Achieved Occupancy	0.485524	0.489766	0.487825
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixProduct(Vector2D*, Vector2D*, Vector2D*)				
gld_transactions	Global Load Transactions	1074	157221906	32781061
gst_transactions	Global Store Transactions	1	524064	39199
gld_efficiency	Global Memory Load Efficiency	22.16%	82.50%	48.11%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	76.86%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total of global load requests from Multiprocessor	232	37208995	8038451
global_store_requests	Total number of global store requests from Multiprocessor	1	524064	39199
sm_efficiency	Multiprocessor Activity	7.67%	99.89%	37.57%
achieved_occupancy	Achieved Occupancy	0.111670	0.904035	0.462840
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: ScalarMinusVector2D(Vector2D*, float, Vector2D*)				
gld_transactions	Global Load Transactions	20482	20482	20482
gst_transactions	Global Store Transactions	512	512	512
gld_efficiency	Global Memory Load Efficiency	56.94%	56.94%	56.94%

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	3072	3072	3072
global_store_requests	Total number of global store requests from Multiprocessor	512	512	512
sm_efficiency	Multiprocessor Activity	30.02%	35.13%	32.02%
achieved_occupancy	Achieved Occupancy	0.456350	0.475226	0.466024
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Sigmoid(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	14346	14346	14346
gst_transactions	Global Store Transactions	512	512	512
gld_efficiency	Global Memory Load Efficiency	59.34%	59.34%	59.34%
gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency	Shared Memory Efficiency	0.00%		
global_load_requests	Total number of global load requests from Multiprocessor	2561	2561	2561
global_store_requests	Total number of global store requests from Multiprocessor	512	512	512
sm_efficiency	Multiprocessor Activity	35.96%	41.40%	39.54%
achieved_occupancy	Achieved Occupancy	0.461936	0.476044	0.467086
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: ScalarMatrixProduct(Vector2D*, float, Vector2D*)

gld_transactions	Global Load Transactions	2582	20967042	5250117
gst_transactions	Global Store Transactions	1	524064	131160
gld_efficiency	Global Memory Load Efficiency	21.21%	56.94%	38.92%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	84.73%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total of global load requests from Multiprocessor	389	4193184	512019
global_store_requests	Total number of global store requests from Multiprocessor	1	524064	63962
sm_efficiency	Multiprocessor Activity	6.04%	99.73%	30.87%
achieved_occupancy	Achieved Occupancy	0.352943	0.795298	0.472345
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Sum2D(Vector2D*)

gld_transactions	Global Load Transactions	114	114	114
gst_transactions	Global Store Transactions	4	4	4
gld_efficiency	Global Memory Load Efficiency	24.81%	24.81%	24.81%
gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	24	24	24
global_store_requests	Total number of global store requests from Multiprocessor	4	4	4
sm_efficiency	Multiprocessor Activity	4.86%	7.36%	6.35%
achieved_occupancy	Achieved Occupancy	0.015616	0.015631	0.015622
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Kernel: Softmax(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	518	518	518
gst_transactions	Global Store Transactions	64	64	64
gld_efficiency	Global Memory Load Efficiency	24.56%	24.56%	24.56%
gst_efficiency	Global Memory Store Efficiency	25.00%	25.00%	25.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	89	89	89
global_store_requests	Total number of global store requests from Multiprocessor	16	16	16
sm_efficiency	Multiprocessor Activity	10.73%	12.00%	11.30%
achieved_occupancy	Achieved Occupancy	0.015623	0.015627	0.015624
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixSubtract(Vector2D*, Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	6338	6338	6338
gst_transactions	Global Store Transactions	128	128	128
gld_efficiency	Global Memory Load Efficiency	60.71%	60.71%	60.71%
gst_efficiency	Global Memory Store Efficiency	89.06%	89.06%	89.06%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	974	974	974
global_store_requests	Total number of global store requests from Multiprocessor	118	118	118
sm_efficiency	Multiprocessor Activity	4.21%	8.91%	7.76%
achieved_occupancy	Achieved Occupancy	0.468496	0.489397	0.483328
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixPairwiseProduct(Vector2D*, Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	6338	26626	19863
gst_transactions	Global Store Transactions	128	512	384
gld_efficiency	Global Memory Load Efficiency	60.71%	66.96%	64.88%
gst_efficiency	Global Memory Store Efficiency	89.06%	100.00%	96.35%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	974	4096	3055
global_store_requests	Total number of global store requests from Multiprocessor	118	512	380
sm_efficiency	Multiprocessor Activity	6.37%	38.30%	25.93%
achieved_occupancy	Achieved Occupancy	0.409192	0.479844	0.446007
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: TransposeVector2D(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	18234	4718574	1585080
gst_transactions	Global Store Transactions	3984	1048562	352214
gld_efficiency	Global Memory Load Efficiency	57.14%	62.50%	59.35%
gst_efficiency	Global Memory Store Efficiency	12.50%	12.50%	12.50%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total of global load requests from Multiprocessor	2530	655357	220149
global_store_requests	Total of global store requests from Multiprocessor	502	131071	44028
sm_efficiency	Multiprocessor Activity	28.80%	98.91%	53.97%
achieved_occupancy	Achieved Occupancy	0.382613	0.759363	0.521375

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: ArgMax2D(Vector2D*)

gld_transactions	Global Load Transactions	134	134	134
gst_transactions	Global Store Transactions	4	4	4
gld_efficiency	Global Memory Load Efficiency	24.81%	24.81%	24.81%
gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	25	25	25
global_store_requests	Total number of global store requests from Multiprocessor	4	4	4
sm_efficiency	Multiprocessor Activity	6.40%	8.40%	7.44%
achieved_occupancy	Achieved Occupancy	0.015621	0.015632	0.015626
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Exponential(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	3394	3394	3394
gst_transactions	Global Store Transactions	128	128	128
gld_efficiency	Global Memory Load Efficiency	53.79%	53.79%	53.79%
gst_efficiency	Global Memory Store Efficiency	89.06%	89.06%	89.06%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total of global load requests from Multiprocessor	611	611	611
global_store_requests	Total of global store requests from Multiprocessor	118	118	118
sm_efficiency	Multiprocessor Activity	6.43%	8.26%	7.50%
achieved_occupancy	Achieved Occupancy	0.481851	0.488086	0.484999
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

A long run nvprof output :

```
==10654== Profiling application: ./OneHiddenClassification
==10654== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   79.26%  137.571s    194772  706.32us  2.2070us  15.877ms
MatrixProduct(Vector2D*, Vector2D*, Vector2D*)
                  8.87%  15.4011s    129786  118.67us  1.5680us  1.1177ms  MatrixAdd(Vector2D*,
Vector2D*, Vector2D*)
                  8.09%  14.0482s     86400  162.59us  1.5680us  1.0158ms
ScalarMatrixProduct(Vector2D*, float, Vector2D*)
                  3.26%  5.66601s     64800  87.438us  2.3360us  620.51us  TransposeVector2D(Vector2D*,
Vector2D*)
                  0.14%  246.88ms        94  2.6264ms     512ns  167.73ms  [CUDA memcpy HtoD]
                  0.07%  128.28ms     64800  1.9790us  1.6000us  17.888us
MatrixPairwiseProduct(Vector2D*, Vector2D*, Vector2D*)
                  0.07%  114.24ms     21693  5.2660us  4.9600us  177.60us  Softmax(Vector2D*,
Vector2D*)
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

```
            0.04%  71.715ms     21693  3.3050us  2.8160us  18.144us  Sigmoid(Vector2D*,
Vector2D*)
            0.04%  61.940ms     43386  1.4270us  1.1830us  16.096us  PointerSet(Vector2D*,
Vector2D*, int, int)
            0.03%  53.689ms     21600  2.4850us  2.3360us  19.776us  Log2D(Vector2D*, Vector2D*)
            0.03%  50.599ms     21600  2.3420us  2.1760us  15.968us  MatrixSubtract(Vector2D*,
Vector2D*, Vector2D*)
            0.03%  47.755ms     21600  2.2100us  1.8880us  15.712us  Sum2D(Vector2D*)
            0.03%  46.123ms     21693  2.1260us  1.9840us  15.808us  Exponential(Vector2D*,
Vector2D*)
            0.03%  45.961ms     21600  2.1270us  2.0470us  20.544us
ScalarMinusVector2D(Vector2D*, float, Vector2D*)
            0.01%  17.283ms     21693     796ns     320ns  17.344us  [CUDA memcpy DtoH]
            0.00%  228.80us        93  2.4600us  2.2720us  3.2000us  ArgMax2D(Vector2D*)
      API calls:   96.81%  177.041s    735633  240.66us  1.6290us  15.902ms  cudaDeviceSynchronize
            2.21%  4.04183s    735516  5.4950us  4.2550us  45.583ms  cudaLaunch
            0.48%  880.29ms   2055069     428ns     122ns  544.72ms  cudaSetupArgument
            0.16%  295.20ms     21693  13.607us  8.9500us  90.177us  cudaMemcpyFromSymbol
            0.14%  248.01ms        94  2.6384ms  3.9640us  167.86ms  cudaMemcpy
            0.08%  149.70ms    735516     203ns     145ns  305.62us  cudaConfigureCall
            0.08%  144.57ms        56  2.5816ms  3.3160us  141.24ms  cudaMalloc
            0.04%  74.445ms         1  74.445ms  74.445ms  74.445ms  cudaDeviceReset
            0.00%  1.0087ms        86  11.729us     447ns  454.83us  cuDeviceGetAttribute
            0.00%  183.45us         1  183.45us  183.45us  183.45us  cuDeviceTotalMem
            0.00%  129.73us         1  129.73us  129.73us  129.73us  cuDeviceGetName
            0.00%  5.9250us         1  5.9250us  5.9250us  5.9250us  cuDeviceGetPCIBusId
            0.00%  4.4700us         3  1.4900us     428ns  3.3060us  cuDeviceGetCount
            0.00%  2.7730us         2  1.3860us     427ns  2.3460us  cuDeviceGet
            0.00%  2.0310us         1  2.0310us  2.0310us  2.0310us  cudaGetDeviceCount



79.26%  137.571s    194772  706.32us  2.2070us  15.877ms  MatrixProduct(Vector2D*, Vector2D*,
Vector2D*)
            8.87%  15.4011s    129786  118.67us  1.5680us  1.1177ms  MatrixAdd(Vector2D*,
Vector2D*, Vector2D*)
            8.09%  14.0482s     86400  162.59us  1.5680us  1.0158ms
ScalarMatrixProduct(Vector2D*, float, Vector2D*)
            3.26%  5.66601s     64800  87.438us  2.3360us  620.51us  TransposeVector2D(Vector2D*,
Vector2D*)
```

These three functions are the ones that must be optimized for performance gain.

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Function combination and independent kernel execution.

Several functions are fused into one. We don't put the metric results because the functions are different from the original one. Also, we have changed the execution order of independent kernels.

First program execution time : 163.206515 Second Program execution time : 170.467363
Average execution time : 166.836939

Long run nvprof output:
Accuracy : 1.444892
Tamam
Program execution time : 163.206515

```
==10774== Profiling application: ./OneHiddenClassificationFunctionCombination
==10774== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   82.90%  118.540s    194772  608.61us  2.1760us  4.3354ms
MatrixProduct(Vector2D*, Vector2D*, Vector2D*)
                  12.30%  17.5821s     86400  203.50us  1.5990us  1.2613ms
ApplyWeightChange(Vector2D*, float, Vector2D*)
                   4.28%  6.11896s     64800  94.428us  2.5280us  909.08us  TransposeVector2D(Vector2D*,
Vector2D*)
                   0.17%  240.27ms        94  2.5560ms     512ns  166.63ms  [CUDA memcpy HtoD]
                   0.10%  143.21ms     21693  6.6010us  5.4400us  215.13us  Softmax(Vector2D*,
Vector2D*)
                   0.05%  67.843ms     21693  3.1270us  2.4320us  23.296us  AddandSigmoid(Vector2D*,
Vector2D*, Vector2D*)
                   0.04%  64.029ms     43386  1.4750us  1.1830us  18.464us  PointerSet(Vector2D*,
Vector2D*, int, int)
                   0.04%  59.791ms     21600  2.7680us  2.3680us  20.896us
calculateCrossEntropyLoss(Vector2D*, Vector2D*, Vector2D*)
                   0.04%  55.511ms     21600  2.5690us  2.2080us  18.079us  Sum2D(Vector2D*)
                   0.04%  55.309ms     21600  2.5600us  2.2400us  17.952us  MatrixSubtract(Vector2D*,
Vector2D*, Vector2D*)
                   0.03%  45.331ms     21600  2.0980us  1.8880us  20.992us
LayerErrorCalculate(Vector2D*, Vector2D*, Vector2D*)
                   0.01%  18.707ms     21693     862ns     352ns  18.784us  [CUDA memcpy DtoH]
                   0.00%  301.63us        93  3.2430us  2.9760us  3.6480us  ArgMax2D(Vector2D*)
      API calls:   97.64%  144.716s    324768  445.60us  1.4850us  25.401ms  cudaDeviceSynchronize
                   1.77%  2.62084s    519237  5.0470us  3.6120us  4.0409ms  cudaLaunchKernel
                   0.20%  298.50ms     21693  13.760us  8.8550us  331.97us  cudaMemcpyFromSymbol
                   0.18%  260.94ms        56  4.6597ms  3.3870us  254.76ms  cudaMalloc
                   0.16%  241.31ms        94  2.5671ms  3.6260us  166.76ms  cudaMemcpy
                   0.05%  76.838ms         1  76.838ms  76.838ms  76.838ms  cudaDeviceReset
                   0.00%  740.16us        96  7.7100us     444ns  314.83us  cuDeviceGetAttribute
                   0.00%  208.83us         1  208.83us  208.83us  208.83us  cuDeviceTotalMem
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

0.00%	119.54us	1	119.54us	119.54us	119.54us	cuDeviceGetName
0.00%	4.7560us	3	1.5850us	413ns	3.5020us	cuDeviceGetCount
0.00%	2.5610us	1	2.5610us	2.5610us	2.5610us	cuDeviceGetPCIBusId
0.00%	2.4820us	2	1.2410us	445ns	2.0370us	cuDeviceGet
0.00%	2.3500us	1	2.3500us	2.3500us	2.3500us	cudaGetDeviceCount

So, before function combination original code has run in average 210,435381 seconds. Function combined version runs 166.836939 seconds. %20 speed gain has been obtained.

Unrolling technique

First program execution time : 177.111507 Second program execution time : 177.466072
Average : 177,2887895

Kernel: Log2D(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	2794	2794	2794
gst_transactions	Global Store Transactions	80	80	80
gld_efficiency	Global Memory Load Efficiency	46.05%	46.05%	46.05%
gst_efficiency	Global Memory Store Efficiency	90.00%	90.00%	90.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total number of global load requests from Multiprocessor	521	521	521
global_store_requests	Total number of global store requests from Multiprocessor	88	88	88
sm_efficiency	Multiprocessor Activity	10.32%	11.54%	10.66%
achieved_occupancy	Achieved Occupancy	0.223332	0.248030	0.234191
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixProduct(Vector2D*, Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	1074	157221906	32780924
gst_transactions	Global Store Transactions	1	524064	39199
gld_efficiency	Global Memory Load Efficiency	22.16%	82.50%	48.11%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	76.86%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total of global load requests from Multiprocessor	232	37208995	8038374
global_store_requests	Total number of global store requests from Multiprocessor	1	524064	39199
sm_efficiency	Multiprocessor Activity	8.21%	99.89%	36.90%
achieved_occupancy	Achieved Occupancy	0.123302	0.941063	0.462160
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: ScalarMinusVector2D(Vector2D*, float, Vector2D*)

gld_transactions	Global Load Transactions	12802	12802	12802
gst_transactions	Global Store Transactions	512	512	512
gld_efficiency	Global Memory Load Efficiency	68.98%	68.98%	68.98%
gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

global_load_requests	Total	of global load requests from Multiprocessor	1920	1920	1920
global_store_requests	Total	of global store requests from Multiprocessor	512	512	512
sm_efficiency		Multiprocessor Activity	9.22%	10.96%	10.37%
achieved_occupancy		Achieved Occupancy	0.437226	0.487806	0.470328
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Sigmoid(Vector2D*, Vector2D*)

gld_transactions		Global Load Transactions	11274	11274	11274
gst_transactions		Global Store Transactions	512	512	512
gld_efficiency		Global Memory Load Efficiency	70.14%	70.14%	70.14%
gst_efficiency		Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	1793	1793	1793
global_store_requests	Total	of global store requests from Multiprocessor	512	512	512
sm_efficiency		Multiprocessor Activity	12.53%	13.23%	13.02%
achieved_occupancy		Achieved Occupancy	0.454561	0.472193	0.465716
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: ScalarMatrixProduct(Vector2D*, float, Vector2D*)

gld_transactions		Global Load Transactions	2562	13102722	3287722
gst_transactions		Global Store Transactions	0	524064	131472
gld_efficiency		Global Memory Load Efficiency	20.83%	68.98%	46.04%
gst_efficiency		Global Memory Store Efficiency	0.00%	100.00%	72.23%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	384	1965408	493158
global_store_requests	Total	of global store requests from Multiprocessor	0	524064	131426
sm_efficiency		Multiprocessor Activity	5.78%	99.50%	38.57%
achieved_occupancy		Achieved Occupancy	0.231258	0.909328	0.515578
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixAdd(Vector2D*, Vector2D*, Vector2D*)

gld_transactions		Global Load Transactions	2562	19391490	2372921
gst_transactions		Global Store Transactions	0	524064	64111
gld_efficiency		Global Memory Load Efficiency	20.83%	75.80%	58.07%
gst_efficiency		Global Memory Store Efficiency	0.00%	100.00%	83.92%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	384	3013536	368756
global_store_requests	Total	of global store requests from Multiprocessor	0	524064	64091
sm_efficiency		Multiprocessor Activity	5.76%	99.63%	23.72%
achieved_occupancy		Achieved Occupancy	0.236834	0.926100	0.453854
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Sum2D(Vector2D*)

gld_transactions		Global Load Transactions	114	114	114
gst_transactions		Global Store Transactions	4	4	4

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

gld_efficiency		Global Memory Load Efficiency	24.81%	24.81%	24.81%
gst_efficiency		Global Memory Store Efficiency	100.00%	100.00%	100.00%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	24	24	24
global_store_requests	Total	of global store requests from Multiprocessor	4	4	4
sm_efficiency		Multiprocessor Activity	5.54%	7.87%	6.65%
achieved_occupancy		Achieved Occupancy	0.015616	0.015631	0.015624
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: Softmax(Vector2D*, Vector2D*)

gld_transactions		Global Load Transactions	518	518	518
gst_transactions		Global Store Transactions	64	64	64
gld_efficiency		Global Memory Load Efficiency	24.56%	24.56%	24.56%
gst_efficiency		Global Memory Store Efficiency	25.00%	25.00%	25.00%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	89	89	89
global_store_requests	Total	of global store requests from Multiprocessor	16	16	16
sm_efficiency		Multiprocessor Activity	11.18%	12.10%	11.65%
achieved_occupancy		Achieved Occupancy	0.015623	0.015627	0.015626
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixSubtract(Vector2D*, Vector2D*, Vector2D*)

gld_transactions		Global Load Transactions	5378	5378	5378
gst_transactions		Global Store Transactions	80	80	80
gld_efficiency		Global Memory Load Efficiency	53.41%	53.41%	53.41%
gst_efficiency		Global Memory Store Efficiency	90.00%	90.00%	90.00%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	824	824	824
global_store_requests	Total	of global store requests from Multiprocessor	88	88	88
sm_efficiency		Multiprocessor Activity	9.57%	10.45%	10.02%
achieved_occupancy		Achieved Occupancy	0.253425	0.293739	0.274066
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

Kernel: MatrixPairwiseProduct(Vector2D*, Vector2D*, Vector2D*)

gld_transactions		Global Load Transactions	5378	18946	14423
gst_transactions		Global Store Transactions	80	512	368
gld_efficiency		Global Memory Load Efficiency	53.41%	75.80%	68.33%
gst_efficiency		Global Memory Store Efficiency	90.00%	100.00%	96.67%
shared_efficiency		Shared Memory Efficiency	0.00%	0.00%	0.00%
global_load_requests	Total	of global load requests from Multiprocessor	824	2944	2237
global_store_requests	Total	of global store requests from Multiprocessor	88	512	370
sm_efficiency		Multiprocessor Activity	8.63%	11.51%	10.50%
achieved_occupancy		Achieved Occupancy	0.249967	0.489317	0.408073
shared_utilization		Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Kernel: TransposeVector2D(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	13826	3538854	1199408	
gst_transactions	Global Store Transactions	4096	1048520	355160	
gld_efficiency	Global Memory Load Efficiency	58.98%	65.09%	61.30%	
gst_efficiency	Global Memory Store Efficiency	12.50%	12.50%	12.50%	
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%	
global_load_requests	Total of global load requests from Multiprocessor	2176	557041	188777	
global_store_requests	Total of global store requests from Multiprocessor	512	131068	44401	
sm_efficiency	Multiprocessor Activity	11.95%	98.40%	51.14%	
achieved_occupancy	Achieved Occupancy	0.413317	0.870657	0.576137	
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)	

Kernel: ArgMax2D(Vector2D*)

gld_transactions	Global Load Transactions	134	134	134	
gst_transactions	Global Store Transactions	4	4	4	
gld_efficiency	Global Memory Load Efficiency	24.81%	24.81%	24.81%	
gst_efficiency	Global Memory Store Efficiency	100.00%	100.00%	100.00%	
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%	
global_load_requests	Total of global load requests from Multiprocessor	25	25	25	
global_store_requests	Total of global store requests from Multiprocessor	4	4	4	
sm_efficiency	Multiprocessor Activity	6.34%	8.54%	7.08%	
achieved_occupancy	Achieved Occupancy	0.015618	0.015632	0.015626	
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)	

Kernel: Exponential(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	2794	2794	2794	
gst_transactions	Global Store Transactions	80	80	80	
gld_efficiency	Global Memory Load Efficiency	46.05%	46.05%	46.05%	
gst_efficiency	Global Memory Store Efficiency	90.00%	90.00%	90.00%	
shared_efficiency	Shared Memory Efficiency	0.00%	0.00%	0.00%	
global_load_requests	Total of global load requests from Multiprocessor	521	521	521	
global_store_requests	Total of global store requests from Multiprocessor	88	88	88	
sm_efficiency	Multiprocessor Activity	6.37%	9.97%	9.34%	
achieved_occupancy	Achieved Occupancy	0.257498	0.283186	0.267128	
shared_utilization	Shared Memory Utilization	Idle (0)	Idle (0)	Idle (0)	

Long run nvprof output :
Program execution time : 177.466072

==17408== Profiling application: ./OneHiddenClassificationUnroll
==17408== Profiling result:

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   80.64%  112.595s    194772  578.09us  2.1440us  3.8781ms
MatrixProduct(Vector2D*, Vector2D*, Vector2D*)
                    8.31%  11.6006s    129786  89.382us  1.2790us  949.91us  MatrixAdd(Vector2D*,
Vector2D*, Vector2D*)
                    6.37%  8.89828s     86400  102.99us  1.2790us  872.60us
ScalarMatrixProduct(Vector2D*, float, Vector2D*)
                    3.83%  5.34408s     64800  82.470us  6.3040us  625.37us  TransposeVector2D(Vector2D*,
Vector2D*)
                    0.22%  307.21ms        94  3.2682ms     512ns  220.42ms  [CUDA memcpy HtoD]
                    0.14%  195.23ms     64800  3.0120us  2.4640us  171.49us
MatrixPairwiseProduct(Vector2D*, Vector2D*, Vector2D*)
                    0.11%  155.41ms     21693  7.1630us  6.9440us  236.89us  Sigmoid(Vector2D*,
Vector2D*)
                    0.08%  112.62ms     21693  5.1910us  4.9280us  172.25us  Softmax(Vector2D*,
Vector2D*)
                    0.07%  92.583ms     21600  4.2860us  4.0320us  324.35us  Log2D(Vector2D*, Vector2D*)
                    0.05%  73.142ms     21600  3.3860us  3.1680us  222.53us  MatrixSubtract(Vector2D*,
Vector2D*, Vector2D*)
                    0.05%  65.097ms     21600  3.0130us  2.8480us  16.928us
ScalarMinusVector2D(Vector2D*, float, Vector2D*)
                    0.05%  65.017ms     21693  2.9970us  2.7840us  18.496us  Exponential(Vector2D*,
Vector2D*)
                    0.04%  60.867ms     43386  1.4020us  1.1520us  18.112us  PointerSet(Vector2D*,
Vector2D*, int, int)
                    0.03%  41.195ms     21600  1.9070us  1.7280us  18.432us  Sum2D(Vector2D*)
                    0.01%  16.281ms     21693     750ns     639ns  15.200us  [CUDA memcpy DtoH]
                    0.00%  227.26us        93  2.4430us  2.2400us  10.080us  ArgMax2D(Vector2D*)
      API calls:   96.37%  141.745s    735633  192.68us  1.6030us  3.8846ms  cudaDeviceSynchronize
                    2.72%  3.99738s    735516  5.4340us  4.2290us  4.1265ms  cudaLaunch
                    0.24%  358.64ms        94  3.8153ms  4.1040us  220.54ms  cudaMemcpy
                    0.21%  310.34ms   2055069     151ns     122ns  395.86us  cudaSetupArgument
                    0.21%  304.90ms     21693  14.055us  9.3090us  227.66us  cudaMemcpyFromSymbol
                    0.10%  147.97ms        56  2.6423ms  3.7540us  144.14ms  cudaMalloc
                    0.10%  139.94ms    735516     190ns     138ns  432.63us  cudaConfigureCall
                    0.05%  80.881ms         1  80.881ms  80.881ms  80.881ms  cudaDeviceReset
                    0.00%  998.84us        86  11.614us     674ns  429.04us  cuDeviceGetAttribute
                    0.00%  251.23us         1  251.23us  251.23us  251.23us  cuDeviceTotalMem
                    0.00%  149.55us         1  149.55us  149.55us  149.55us  cuDeviceGetName
                    0.00%  7.3260us         3  2.4420us     857ns  4.5690us  cuDeviceGetCount
                    0.00%  4.3150us         1  4.3150us  4.3150us  4.3150us  cuDeviceGetPCIBusId
                    0.00%  3.8930us         2  1.9460us  1.0890us  2.8040us  cuDeviceGet
                    0.00%  3.0110us         1  3.0110us  3.0110us  3.0110us  cudaGetDeviceCount
```

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

If we focus on four bottleneck functions MatrixProduct, MatrixAdd, ScalarMatrixProduct, TransposeVector2D,

Matrix product unrolled:
gld_requested_throughput Requested Global Load Throughput 129.48MB/s 207.33GB/s 152.35GB/s
gst_requested_throughput Requested Global Store Throughput 1.9325MB/s 6.2180GB/s 1.4865GB/s

gld_throughput Global Load Throughput 525.63MB/s 259.60GB/s 186.69GB/s
gst_throughput Global Store Throughput 3.8649MB/s 6.2180GB/s 1.4869GB/s

Matrix product rolled:
gld_requested_throughput Requested Global Load Throughput 203.49MB/s 232.36GB/s 180.50GB/s
gst_requested_throughput Requested Global Store Throughput 3.0372MB/s 6.9685GB/s 1.7613GB/s

gld_throughput Global Load Throughput 826.11MB/s 290.94GB/s 221.20GB/s
gst_throughput Global Store Throughput 4.9869MB/s 6.9685GB/s 1.7617GB/s

MatrixAdd unrolled:
gld_requested_throughput Requested Global Load Throughput 177.00MB/s 55.968GB/s 44.923GB/s
gst_requested_throughput Requested Global Store Throughput 4.0690MB/s 23.879GB/s 19.165GB/s

gld_throughput Global Load Throughput 821.94MB/s 83.585GB/s 67.109GB/s
gst_throughput Global Store Throughput 8.1380MB/s 23.879GB/s 19.168GB/s

MatrixAdd rolled:
gld_requested_throughput Requested Global Load Throughput 232.96MB/s 47.692GB/s 46.075GB/s
gst_requested_throughput Requested Global Store Throughput 0.00000B/s 21.420GB/s 20.691GB/s

gld_throughput Global Load Throughput 908.26MB/s 62.922GB/s 60.821GB/s
gst_throughput Global Store Throughput 0.00000B/s 21.420GB/s 20.699GB/s

ScalarMatrixProduct unrolled:
gld_requested_throughput Requested Global Load Throughput 169.45MB/s 34.955GB/s 30.203GB/s
gst_requested_throughput Requested Global Store Throughput 4.0346MB/s 27.281GB/s 23.568GB/s
gld_throughput Global Load Throughput 798.85MB/s 61.390GB/s 53.060GB/s
gst_throughput Global Store Throughput 8.0692MB/s 27.281GB/s 23.571GB/s

ScalarMatrixProduct rolled:
gld_requested_throughput Requested Global Load Throughput 232.96MB/s 47.692GB/s 46.075GB/s
gst_requested_throughput Requested Global Store Throughput 0.00000B/s 21.420GB/s 20.691GB/s
gld_throughput Global Load Throughput 908.26MB/s 62.922GB/s 60.821GB/s
gst_throughput Global Store Throughput 0.00000B/s 21.420GB/s 20.699GB/s

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

TransposeVector2D unrolled:

gld_requested_throughput	Requested Global Load Throughput	3.0488GB/s	19.200GB/s	18.534GB/s
gst_requested_throughput	Requested Global Store Throughput	2.4391GB/s	15.360GB/s	14.827GB/s
gld_throughput	Global Load Throughput	4.8781GB/s	33.601GB/s	32.422GB/s
gst_throughput	Global Store Throughput	19.513GB/s	122.88GB/s	118.62GB/s

TransposeVector2D rolled:

gld_requested_throughput	Requested Global Load Throughput	232.96MB/s	47.692GB/s	46.075GB/s
gst_requested_throughput	Requested Global Store Throughput	0.00000B/s	21.420GB/s	20.691GB/s
gld_throughput	Global Load Throughput	908.26MB/s	62.922GB/s	60.821GB/s
gst_throughput	Global Store Throughput	0.00000B/s	21.420GB/s	20.699GB/s

As we can observe from metrics above by unrolling more data requests and store operations are being demanded. When the number of instructions increases GPU has a better ability to handle instruction optimization, fullfilling the requested demands.

So before function combination original code has run in average 210,435381 seconds. Unrolled technique combined version runs 177,2887895 seconds. %15,7 speed gain has been obtained.

Shared Memory with Padding

There are two functions where shared memory is used. MatrixProduct and TransposeVector.

First program execution time : 140.887181 Second program execution time : 132.795527
Average : 136,841354

Kernel: MatrixProductShared(Vector2D*, Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	5766	27262082	2701366
gst_transactions	Global Store Transactions	1	524064	39199
gld_efficiency	Global Memory Load Efficiency	18.87%	91.37%	47.60%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	76.86%
shared_efficiency	Shared Memory Efficiency	70.00%	70.00%	70.00%
global_load_requests	Total of global load requests from Multiprocessor	929	4718368	518055
global_store_requests	Total of global store requests from Multiprocessor	1	524064	39199
sm_efficiency	Multiprocessor Activity	4.48%	99.69%	35.17%
achieved_occupancy	Achieved Occupancy	0.490692	0.942265	0.527091
shared_utilization	Shared Memory Utilization	Low (1)	High (7)	Low (2)

Kernel: TransposeVector2DShared(Vector2D*, Vector2D*)

gld_transactions	Global Load Transactions	18322	6290942	2111283
gst_transactions	Global Store Transactions	128	131040	43893
gld_efficiency	Global Memory Load Efficiency	42.16%	44.78%	43.16%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	83.33%
shared_efficiency	Shared Memory Efficiency	33.69%	100.00%	77.89%

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

global_load_requests Total of global load requests from Multiprocessor 3437 1179551 395865
global_store_requests Total of global store requests from Multiprocessor 128 131040 43893
sm_efficiency Multiprocessor Activity 9.07% 98.50% 53.66%
achieved_occupancy Achieved Occupancy 0.485462 0.926145 0.632801
shared_utilization Shared Memory Utilization Low (1) Low (1) Low (1)

Long term execution in nvprof:
Program execution time : 140.887181

==12736== Profiling application: ./OneHiddenClassificationSharedMem
==12736== Profiling result:

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	62.82%	59.8691s	194772	307.38us	2.5920us	2.4894ms	MatrixProductShared(Vector2D*, Vector2D*, Vector2D*)
	16.60%	15.8225s	129786	121.91us	1.5360us	1.2451ms	MatrixAdd(Vector2D*, Vector2D*, Vector2D*)
	14.97%	14.2692s	86400	165.15us	1.5360us	1.2639ms	ScalarMatrixProduct(Vector2D*, float, Vector2D*)
	4.64%	4.41929s	64800	68.198us	2.1440us	676.76us	TransposeVector2DShared(Vector2D*, Vector2D*)
	0.28%	267.90ms	94	2.8499ms	512ns	184.99ms	[CUDA memcpy HtoD]
	0.14%	132.01ms	64800	2.0370us	1.6630us	20.928us	MatrixPairwiseProduct(Vector2D*, Vector2D*, Vector2D*)
	0.12%	114.77ms	21693	5.2900us	4.8960us	286.17us	Softmax(Vector2D*, Vector2D*)
	0.08%	73.697ms	21693	3.3970us	2.9760us	284.32us	Sigmoid(Vector2D*, Vector2D*)
	0.07%	65.025ms	43386	1.4980us	1.1520us	23.072us	PointerSet(Vector2D*, Vector2D*, int, int)
	0.06%	53.529ms	21600	2.4780us	2.3040us	18.272us	Log2D(Vector2D*, Vector2D*)
	0.05%	50.352ms	21600	2.3310us	2.1760us	16.640us	MatrixSubtract(Vector2D*, Vector2D*, Vector2D*)
	0.05%	47.695ms	21600	2.2080us	1.8880us	22.848us	Sum2D(Vector2D*)
	0.05%	46.108ms	21693	2.1250us	1.9520us	17.759us	Exponential(Vector2D*, Vector2D*)
	0.05%	45.188ms	21600	2.0920us	1.8880us	18.048us	ScalarMinusVector2D(Vector2D*, float, Vector2D*)
	0.02%	19.372ms	21693	892ns	320ns	19.296us	[CUDA memcpy DtoH]
	0.00%	207.49us	93	2.2310us	1.9830us	5.6960us	ArgMax2D(Vector2D*)
API calls:	94.95%	104.730s	735633	142.37us	1.6200us	7.2172ms	cudaDeviceSynchronize
	3.79%	4.17586s	735516	5.6770us	4.1260us	4.0064ms	cudaLaunch
	0.34%	372.96ms	21693	17.192us	9.3710us	160.71us	cudaMemcpyFromSymbol
	0.31%	342.73ms	2055069	166ns	126ns	426.67us	cudaSetupArgument

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

0.24%	269.58ms	94	2.8679ms	4.0420us	185.11ms	cudaMemcpy
0.16%	178.02ms	735516	242ns	156ns	399.27us	cudaConfigureCall
0.14%	150.39ms	56	2.6856ms	3.5320us	146.60ms	cudaMalloc
0.07%	76.008ms	1	76.008ms	76.008ms	76.008ms	cudaDeviceReset
0.00%	817.88us	86	9.5100us	443ns	359.22us	cuDeviceGetAttribute
0.00%	180.99us	1	180.99us	180.99us	180.99us	cuDeviceTotalMem
0.00%	111.38us	1	111.38us	111.38us	111.38us	cuDeviceGetName
0.00%	4.7350us	1	4.7350us	4.7350us	4.7350us	cuDeviceGetPCIBusId
0.00%	4.5910us	3	1.5300us	427ns	3.4610us	cuDeviceGetCount
0.00%	2.8570us	2	1.4280us	508ns	2.3490us	cuDeviceGet
0.00%	1.9700us	1	1.9700us	1.9700us	1.9700us	cudaGetDeviceCount

MatrixProduct without shared memory:

gld_transactions	Global Load Transactions	1074	157221906	32781061
gst_transactions	Global Store Transactions	1	524064	39199
gld_throughput	Global Load Throughput	525.63MB/s	259.60GB/s	186.69GB/s
gst_throughput	Global Store Throughput	3.8649MB/s	6.2180GB/s	1.4869GB/s
global_load_requests	Total of global load requests from Multiprocessor	232	37208995	8038451
global_store_requests	Total of global store requests from Multiprocessor	1	524064	39199
gld_efficiency	Global Memory Load Efficiency	22.16%	82.50%	48.11%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	76.86%

MatrixProduct with shared memory:

gld_transactions	Global Load Transactions	5766	27262082	2701366
gst_transactions	Global Store Transactions	1	524064	39199
gld_throughput	Global Load Throughput	1.6716GB/s	58.979GB/s	29.479GB/s
gst_throughput	Global Store Throughput	6.6343MB/s	15.723GB/s	3.5334GB/s
global_load_requests	Total of global load requests from Multiprocessor	929	4718368	518055
global_store_requests	Total of global store requests from Multiprocessor	1	524064	39199
gld_efficiency	Global Memory Load Efficiency	21.21%	56.94%	38.92%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	84.73%

TransposeVector2D without shared memory:

gld_transactions	Global Load Transactions	18234	4718574	1585080
gst_transactions	Global Store Transactions	3984	1048562	352214
gld_throughput	Global Load Throughput	4.8781GB/s	33.601GB/s	32.422GB/s
gst_throughput	Global Store Throughput	19.513GB/s	122.88GB/s	118.62GB/s
gld_throughput	Global Load Throughput	4.8781GB/s	33.601GB/s	32.422GB/s

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

gst_throughput	Global Store Throughput	19.513GB/s	122.88GB/s	118.62GB/s
gld_efficiency	Global Memory Load Efficiency	57.14%	62.50%	59.35%
gst_efficiency	Global Memory Store Efficiency	12.50%	12.50%	12.50%

TransposeVector2D with shared memory:

gld_transactions	Global Load Transactions	18322	6290942	2111283
gst_transactions	Global Store Transactions	128	131040	43893
gld_throughput	Global Load Throughput	10.722GB/s	67.067GB/s	64.456GB/s
gst_throughput	Global Store Throughput	927.63MB/s	21.036GB/s	20.165GB/s
global_load_requests	Total of global load requests from Multiprocessor	3437	1179551	395865
global_store_requests	Total of global store requests from Multiprocessor	128	131040	43893
gld_efficiency	Global Memory Load Efficiency	42.16%	44.78%	43.16%
gst_efficiency	Global Memory Store Efficiency	50.00%	100.00%	83.33%

From above, we can deduce that shared memory usage leads to coalesced and aligned reading and storing operations. In unoptimized form some readings and storings were unaligned. GPU performs more memory transaction to fulfill this demand. On the other hand, the reading is performed in coalesced and aligned manner GPU needed less memory transactions to fulfill this demand.
Shared memory usage lessened total transactions to global memory.

So before shared memory usage original code has run in average 210,435381 seconds. Shared memory with padding runs 136,841354 seconds. %35 speed gain has been obtained.

Texture Cache

First program execution time : 208.765870 Second program execution time : 207.500291
Average : 208,1330805

Speed gain : %1,1 The effect of it is so small, we just show the most bottleneck function.

Texture not used:
Kernel: MatrixProduct(Vector2D*, Vector2D*, Vector2D*)

gst_transactions	Global Store Transactions	1	524064	39199
l2_write_transactions	L2 Write Transactions	14	524097	39213
gst_requested_throughput	Requested Global Store Throughput	1.9325MB/s	6.2180GB/s	1.4865GB/s
tex_cache_transactions	Unified Cache Transactions	268	39305476	8195230
global_store_requests	Total of global store requests from Multiprocessor	1	524064	39199

Texture used:
Kernel: MatrixProduct(Vector2D*, Vector2D*, Vector2D*)

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

gst_transactions	Global Store Transactions	1	524064	10952	
l2_write_transactions	L2 Write Transactions	14	524077	10966	
gst_requested_throughput	Requested Global Store Throughput	2.5718MB/s	6.2308GB/s	383.33MB/s	
tex_cache_transactions	Unified Cache Transactions	269	39305477	8336842	
global_store_requests	Total of global store requests from Multiprocessor	1	524064	10952	

Usage of texture cache lessened write transactions and performed the same work in less number of store transactions.

Batch Size and Thread Block number:

We have tested several block sizes and batch sizes.

BATCH_SIZE	block_x	block_y	feed_block_x	feed_block_y	back_block_x	back_block_y	error	time	accuracy
32	32	32					1792	66.366664	76.713711
32	64	16							
32	16	64							
32	16	16					0	51.657997	1.444892
32	32	32	32	32	32	32	1792	66.366664	76.713711
32	64	16	32	32	32	32	0	67.016877	76.713711
32	16	64	32	32	32	32	503	66.146933	76.713711
32	16	16	32	32	32	32	461	66.406175	76.680106
32	32	32	32	32	16	16	3105	64.312346	52.016127
32	64	16	32	32	16	16	0	63.489224	51.276881
32	16	64	32	32	16	16	839	63.825981	52.184141
32	16	16	32	32	16	16	804	65.789716	52.284944
32	32	32	16	16	32	32	0	51.348242	1.444892
32	64	16	16	16	32	32	0	53.553986	1.444892
32	16	64	16	16	32	32	0	53.570938	1.444892
32	16	16	16	16	32	32	0	53.886724	1.444892
64	32	32	32	32	32	32	0	44.531389	1.42663
64	64	16	32	32	32	32	0	44.431587	1.42663
64	16	64	32	32	32	32	0	44.511612	1.42663
64	16	16	32	32	32	32	0	44.443287	1.42663
64	32	32	32	32	16	16	0	43.043412	1.42663
64	64	16	32	32	16	16			
64	16	64	32	32	16	16			
64	16	16	32	32	16	16			
64	32	32	16	16	32	32		40.474114	
64	64	16	16	16	32	32			
64	16	64	16	16	32	32			
64	16	16	16	16	32	32			
128	32	32	32	32	32	32		37.01504	

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

128	64	16	32	32	32	32	
128	16	64	32	32	32	32	
128	16	16	32	32	32	32	
128	32	32	32	32	16	16	
128	64	16	32	32	16	16	
128	16	64	32	32	16	16	
128	16	16	32	32	16	16	
128	32	32	16	16	32	32	
128	64	16	16	16	32	32	
128	16	64	16	16	32	32	
128	16	16	16	16	32	32	
256							35.088176
512							30.903113
1024							25.800068
2048							22.170835

We have used batch size of 1024 and thread block as 32 x 32.

Serial Code Comparisons

We have tested our serial, parallel and GPU code with the same amount of data and the same task. Each code is given 1600 data point and perform operation on that data.
Serial code run 1: 233.059089 run 2: 232.267144 Average: 232.6631165
Parallel(16 thread) code run 1: 103.712619 run 2: 111.021819 Average: 107.367219
Parallel(8 thread) code run 1: 103.926238 run 2: 103.134728 Average: 103.530483
Unoptimized code run 1: 0.635689 run 2: 0.633862 Average : 0.6347755
Tensorflow code run1: 0.404778003693 Run2: 0.412333011627 Average : 0.40855550766
Optimized code run 1: 0.430240 Run2: 0.441005 Average : 0.4356225

Unoptimized whole data run : (191.457468 + 190.580037)/2 = 191.0187525

Optimized whole data run: (38.038570+38.054162)/2 = 38.046366

Comparison with Other Project

https://github.com/xqding/NeuralNetwork_GPU-CUDA-_MNIST
Other project code execution times first run Program execution time : 10.119466 second Program execution time : 10.064995 Average : 10.0922305
Our unoptimized code execution time – First run Program execution time : 76.402052 Program execution time : 76.213952 Average: 76.308002

N12165419 Mustafa KÖSE

O18148203 Saim SUNEL

Optimized(32 batch) code run 1: 43.530097 run 2: 43.498238 Average : 43.5141675