1、

I implement the CPU version just by by using two for loops to achieve the matrix transpose, for the naive CUDA version I transpose the matrix by the global memory in the CUDA, and for the share memory I firstly put the data from the main memory to the share memory in the same block and then write it back to the main memory, for the bank non-conflict version, add an unused column of the share memory matrix. And for the unroll version, I make each thread do four times data moving and loop unrolling to get full use of the wrap and the thread. For the unroll version, I firstly change the size of the matrix and then use the same number of the total threads in one grid. Secondly, I test another version by use streams to launch the multi-kernel function, and also use the same number of the total threads to get the better running time.

The problem I met is that my GPU in the computer did not support streams, so the second version of unrolling loops did not test fully

2、

I test the execute time of the CUDA by use the `cudaEvent_t, cudaEventCreate, cudaEventRecord, cudaEventElapsedTime` functions

3、

http://baike.baidu.com/link?url=zz1Tt_erqQLf7BrsUQ3y0lsjGnuseBAUnhIvJBTv8xoE8_MHtA4fQXmldYVq9JjmwjdFQnp0laz2ULlxE8rs-a

This is a NVIDIA GPU with 384 cores and 850MHz drawing frequency and at most 64GB/sec, which support PCI Express2.0 and PCI Express 3.0, the biggest

VGA is at most 2048*1536 and support the 3D Blu-Ray

4、

test result (1024*1024)

	Time(ms)	bindwith	Step speed up	speed up
CPU	21.474(1k*1k)	0.976		
Naïve	1.3965(1k*1k)	15.017	153.8627	15.38
Coalesce	1.2637(1k*1k)	16.595	1.105	17.00
Bank	1.2047(1k*1k)	17.408	1.0489	17.834
Loop	4.3478(2k*2k)	19.293	1.108	19.76

The result shows that the CUDA performance much better than CPU and the unroll loop have a better speed up than others