Optimized Matrix Multiplication on GPU

K. Matsumoto* N. Nakasato* T. Sakai* S.G. Sedukhin*

We present an implementation of dense matrix-matrix multiply-add (MMA) on GPU from AMD when the memory requirement is larger than a GPU's local memory. This MMA implementation utilizes our previous DGEMM (Double-precision General Matrix Multiply) kernel, which efficiently works on a Cypress GPU.

One of the drawbacks of GPU processing is the communication time between the host memory and GPU's local memory. The proposed implementation overlaps the communication and the computation and alleviate the communication latency. For our implementation on a Cypress GPU running at 850 MHz clock speed, we have selected the block size of 1792×1792 in the blocked MMA algorithm to hide the latency. In addition, the multiplication kernel of a transposed matrix and a non-transposed matrix (TN) in row-major order was discovered faster than that of non-transposed matrices (NN) in the DGEMM kernel; therefore we use the TN kernel for our implementation by copying matrix data into transposed form before transferring it to GPU. Fig. 1 shows the performance in Gflop/s of our NN implementation with and without the optimization on the GPU.

In Fig. 1, the performance in multiples of 1792 (the blocking factor) is only depicted. When a size is not the multiples, we insert the zero padding so that MMA can effectively be done on GPU. The performance is sensitive to the amount of padding since it increases the redundant computations on the padded portions. Thus, our MMA implementation chooses the appropriate size of blocking in multiples of 128. Fig. 2 shows the performance of NN implementation with the padding. Notice that the blocking factors used are up to 1792 because the GPU with 1 GB local memory cannot allocate the memory region for the bigger blocking factor.

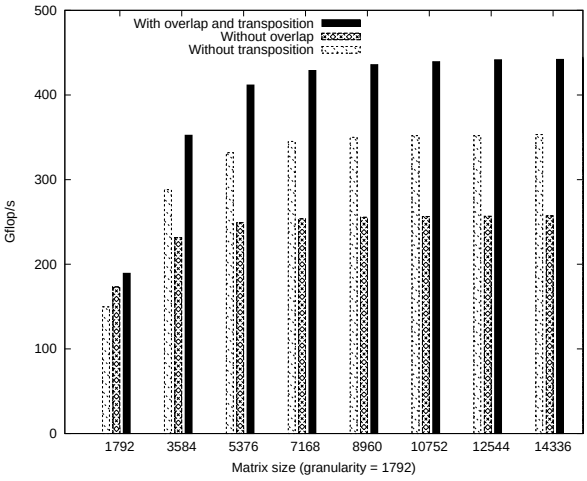

Figure 1: MMA performance with and without optimizations

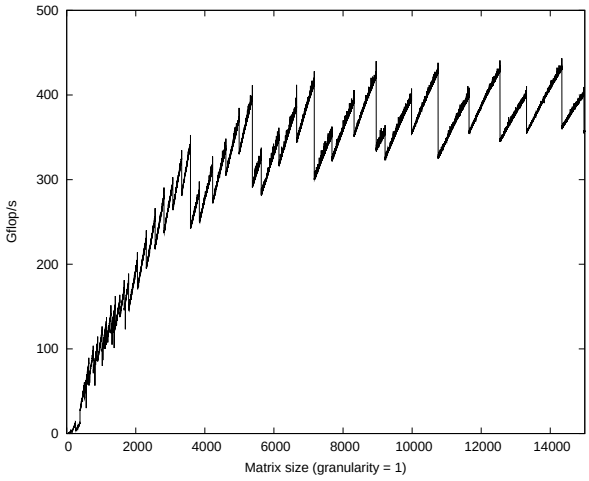

Figure 2: MMA performance with a padding

*The University of Aizu