Parallel Processing Lab 4:

Vector Dot Product

Sarah Peachey & Nathan Schomer

March 11, 2018

Abstract: A vector dot product is the element wise multiplication and then summation of two vectors with n elements. The summation of all the elements is a form of reduction. The best form of reduction is one that tries reduce branch divergence by not dividing warps until only one warp is left.

1 Design

For a given input value, n elements are created. If n is greater than the possible number of threads, then a stride is needed to bring all the elements into shared memory. Each thread initializes a spot in the C_shared vector to zero then brings in the A and B value multiplies them and accumulates them the C_shared memory vector. Reduction is then performed on the C_shared vector by splitting the vector in half and using the first half of the threads to add the second half of the data to the first half of the data. The first half is then halved over and over again. Branch divergence does not occur until the size of the reduction is less than one warp. Then the final answer is stored in the zeroeth element of the C_shared vector, that is moved to global memory and copied back to the CPU.

Data: Ad, Bd, Cd

Result: kernel to calculate the vector dot product

calculate number of threads (k);

calculate the tid;

create shared C vector;

calculate number of strides;

initialization shared memory;

for *i = number of strides* **do**

    C[tid]+=Ad[tid+(k*i)]*Bd[tid+(k*i)];

end

now the element wise multiplication is in shared memory;

perform reduction;

for *stride=k; stride¿0; stride/=2* **do**

    if(tid¡stride) C[tide]+=C[tid+shared];

    synch;

end

Cd=C[0];

2 Discussion of speed up

Speed-up was calculated for the vector dot roduct GPU and CPU code and the speed up was calculated with, Equation 1. As seen by the speed-up ratios in Table 1, when 4096 threads are used there is definitive speed up but the epsilon values were in the range of 10-50. Whereas, where the code was ran with 1024 threads there was actually a slow down do to the overhead, but the epsilon values were zero. Furthermore, time was recorded for transfering the constant and without transfering the constant and one can see that transferring data is clearly the bottle neck.

$$s = \frac{t_{serial}}{t_{parallel}} \tag{1}$$

Number of threads and Elements	Speed-Up with constant	Speed-Up without
4096 10^5	1.278	1.545
4096 10^6	2.370	2.708
4096 10^7	2.473	2.503
1024 10^5	0.698	0.758
1024 10^6	0.747	0.780
1024 10^7	0.634	0.639

Table 1: Speed-Up calculated on Xunil-05