

Parallel Processing Lab 4:

Vector Dot Product

Sarah Peachey & Nathan Schomer

March 10, 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 to 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 $i = \text{number of strides}$ **do**

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

end

now the element wise multiplication is in shared memory;

perform reduction;

for $depth=1; depth \leq k; depth*=2$ **do**

 stride=k/(2*depth);

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

 synch;

end

Cd=C[0];

2 Discussion of speed up

Speed-up was calculated for the vector dot product GPU and CPU code and the speed up was calculated with, Equation 1. As seen by the speed-up ratios in Table 1, a trend of increased speed-up with increased matrix size was found. This is to be expected since an increased number of FLOPs on the GPU decreases the impact of kernel initialization and related GPGPU overhead.

$$s = \frac{t_{serial}}{t_{parallel}} \quad (1)$$

| Number of Elements | Speed-Up |
|--------------------|----------|
| 10^5 | 0.00 |
| 10^6 | 0.00 |
| 10^7 | 0.00 |

Table 1: Speed-Up calculated on Xunil-05