

02/05/11
22:43:54

VecAdd/VecAdd.cu

1

```
1: #include <R.h>
2:
3: __global__ void VecAdd(float* a, float* b, float* c) {
4:     int k = blockIdx.x * 50 + threadIdx.x;
5:     c[k] = a[k] + b[k];
6: }
7:
8: extern "C" {
9:     void VecAddR(float* a, float* b, float* c) {
10:         float *a_GPU, *b_GPU, *c_GPU;
11:
12:         cudaMalloc(&a_GPU, 1000*sizeof(float));
13:         cudaMalloc(&b_GPU, 1000*sizeof(float));
14:         cudaMalloc(&c_GPU, 1000*sizeof(float));
15:
16:         cudaMemcpy(a_GPU, a, 1000*sizeof(float), cudaMemcpyHostToDevice);
17:         cudaMemcpy(b_GPU, b, 1000*sizeof(float), cudaMemcpyHostToDevice);
18:
19:         VecAdd<<<20,50>>>>(a_GPU, b_GPU, c_GPU);
20:         cudaThreadSynchronize();
21:
22:         cudaMemcpy(c, c_GPU, 1000*sizeof(float), cudaMemcpyDeviceToHost);
23:
24:         cudaFree(a_GPU);
25:         cudaFree(b_GPU);
26:         cudaFree(c_GPU);
27:     }
28: }
29:
```

02/05/11
14:48:57

VecAdd/VecAdd.R

1

```
1: dyn.load("VecAdd.so")
2:
3: a <- 1:1000
4: b <- 1000:1
5:
6: res <- .C("VecAddR", as.single(a), as.single(b), c=as.single(1:1000))
7:
8: res$c
```

02/06/11
19:32:31

Combn/combnGPU.cu

1

```
1: #include <R.h>
2:
3: __global__ void findSumGPU_7(float *probs, float thres, float *res2) {
4:     int i1 = blockIdx.x;
5:     int i2 = threadIdx.x;
6:     int i3 = threadIdx.y;
7:     float res = 0.0;
8:     float tot = 0.0;
9:     tot += probs[i1 + 0 * 17] + probs[i2 + 1 * 17] + probs[i3 + 2 * 17];
10:    for(int i4=0; i4<17; i4++) {
11:        tot += probs[i4 + 3 * 17];
12:        for(int i5=0; i5<17; i5++) {
13:            tot += probs[i5 + 4 * 17];
14:            for(int i6=0; i6<17; i6++) {
15:                tot += probs[i6 + 5 * 17];
16:                for(int i7=0; i7<17; i7++) {
17:                    if(tot + probs[i7 + 6 * 17] < thres) {
18:                        res += exp(tot + probs[i7 + 6 * 17]);
19:                    } else {
20:                        break;
21:                    }
22:                }
23:                tot -= probs[i6 + 5 * 17];
24:            }
25:            tot -= probs[i5 + 4 * 17];
26:        }
27:        tot -= probs[i4 + 3 * 17];
28:    }
29:    res2[i1 + i2*17 + i3*17*17] = res;
30: }
31:
32: extern "C" {
33:     // Function to compute the sum of all probabilities of permutations below a given threshold
34:     // probs (input)
35:     // 17 x 7 matrix of log(probabilities). Events in COLUMNS, outcomes in ROWS. Must be presorted with highest probabilities in lowest row index
36:     // thres (input)
37:     // log(probability) of upper limit
38:     // res (output)
39:     // sum of all permutations below given threshold
40:     void computeUnlikelySumGPU(float *probs, float *thres, double *res) {
41:         float res2[17*17*17];
42:         for(int i=0; i<17*17*17; i++) {
43:             res2[i] = 0.0;
44:         }
45:
46:         float *GPU_probs, *GPU_res2;
47:         cudaMalloc(&GPU_probs, 17 * 7 * sizeof(float));
48:         cudaMalloc(&GPU_res2, 17*17*17 * sizeof(float));
49:
50:         cudaMemcpy(GPU_probs, probs, 17 * 7 * sizeof(float), cudaMemcpyHostToDevice);
51:         cudaMemcpy(GPU_res2, res2, 17*17*17 * sizeof(float), cudaMemcpyHostToDevice);
52:     }
53:
54:     int numBlocks = 17;
55:     dim3 threadsPerBlock(17, 17);
56:     findSumGPU_7<<<numBlocks, threadsPerBlock>>>(GPU_probs, *thres, GPU_res2);
57:     cudaMemcpy(probs, GPU_probs, 17 * 7 * sizeof(float), cudaMemcpyDeviceToHost);
58:     cudaMemcpy(res2, GPU_res2, 17*17*17 * sizeof(float), cudaMemcpyDeviceToHost);
59:
60:     cudaFree(GPU_probs);
61:     cudaFree(GPU_res2);
62: }
```

02/06/11
19:32:31

Combn/combnGPU.cu

2

```
63:     *res = 0.0;
64:     for(int i=0; i<17*17*17; i++) {
65:         *res += (double) res2[i];
66:     }
67: }
68: }
```

02/07/11
11:57:50

Combn/combnGPU_shared.cu

1

```
1: #include <R.h>
2:
3: __global__ void findSumGPU_7(float *probs, float thres, float *res2) {
4:     int i1 = blockIdx.x;
5:     int i2 = threadIdx.x;
6:     int i3 = threadIdx.y;
7:     float res = 0.0;
8:     float tot = 0.0;
9:     tot += probs[i1 + 0 * 17] + probs[i2 + 1 * 17] + probs[i3 + 2 * 17];
10:
11:     __shared__ float r4[17];
12:     __shared__ float r5[17];
13:     __shared__ float r6[17];
14:     __shared__ float r7[17];
15:
16:     if(i2 == 0) {
17:         r4[i3] = probs[i3 + 3 * 17];
18:     }
19:     if(i2 == 1) {
20:         r5[i3] = probs[i3 + 4 * 17];
21:     }
22:     if(i2 == 2) {
23:         r6[i3] = probs[i3 + 5 * 17];
24:     }
25:     if(i2 == 3) {
26:         r7[i3] = probs[i3 + 6 * 17];
27:     }
28:
29:     for(int i4=0; i4<17; i4++) {
30:         tot += r4[i4];
31:         for(int i5=0; i5<17; i5++) {
32:             tot += r5[i5];
33:             for(int i6=0; i6<17; i6++) {
34:                 tot += r6[i6];
35:                 for(int i7=0; i7<17; i7++) {
36:                     if(tot + r7[i7] < thres) {
37:                         res += exp(tot + r7[i7]);
38:                     } else {
39:                         break;
40:                     }
41:                 }
42:                 tot += r6[i6];
43:             }
44:             tot += r5[i5];
45:         }
46:         tot += r4[i4];
47:     }
48:     res2[i3 + i2*17 + i1*17*17] = res;
49: }
50:
51: extern "C" {
52:     // Function to compute the sum of all probabilities of permutations below a given threshold
53:     // probs (input)
54:     // 17 x 7 matrix of log(probabilities). Events in COLUMNS, outcomes in ROWS. Must be presorted with highest probabilities in lowest row index
55:     // thres (input)
56:     // log(probability) of upper limit
57:     // res (output)
58:     // sum of all permutations below given threshold
59:     void computeUnlikelySumGPU_shared(float *probs, float *thres, double *res) {
60:         float res2[17*17*17];
61:         for(int i=0; i<17*17*17; i++) {
62:             res2[i] = 0.0;
63:         }
64:
65:         float *GPU_probs, *GPU_res2;
66:         cudaMalloc(&GPU_probs, 17 * 7 * sizeof(float));
```

02/07/11
11:57:50

Combn/combnGPU_shared.cu

2

```
67:         cudaMalloc(&GPU_res2, 17*17*17 * sizeof(float));
68:
69:         cudaMemcpy(GPU_probs, probs, 17 * 7 * sizeof(float), cudaMemcpyHostToDevice);
70:         cudaMemcpy(GPU_res2, res2, 17*17*17 * sizeof(float), cudaMemcpyHostToDevice);
71:
72:         int numBlocks = 17;
73:         dim3 threadsPerBlock(17, 17);
74:         findSumGPU_7(<<numBlocks, threadsPerBlock>>)(GPU_probs, *thres, GPU_res2);
75:
76:         cudaMemcpy(probs, GPU_probs, 17 * 7 * sizeof(float), cudaMemcpyDeviceToHost);
77:         cudaMemcpy(res2, GPU_res2, 17*17*17 * sizeof(float), cudaMemcpyDeviceToHost);
78:
79:         cudaFree(GPU_probs);
80:         cudaFree(GPU_res2);
81:
82:         *res = 0.0;
83:         for(int i=0; i<17*17*17; i++) {
84:             *res += (double) res2[i];
85:         }
86:     }
87: }
```