```cpp
#include "../common/common.h"
#include <cuda_runtime.h>
#include <stdio.h>

/*
 * simpleDivergence demonstrates divergent code on the GPU and its impact on
 * performance and CUDA metrics.
 */

__global__ void mathKernel1(float *c)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    float ia, ib;
    ia = ib = 0.0f;

    if (tid % 2 == 0)
    {
        ia = 100.0f;
    }
    else
    {
        ib = 200.0f;
    }

    c[tid] = ia + ib;
}

__global__ void mathKernel2(float *c)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    float ia, ib;
    ia = ib = 0.0f;

    if ((tid / warpSize) % 2 == 0)
    {
```

```
36          ia = 100.0f;
37      }
38      else
39      {
40          ib = 200.0f;
41      }
42
43      c[tid] = ia + ib;
44  }
45
46  __global__ void mathKernel3(float *c)
47  {
48      int tid = blockIdx.x * blockDim.x + threadIdx.x;
49      float ia, ib;
50      ia = ib = 0.0f;
51
52      bool ipred = (tid % 2 == 0);
53
54      if (ipred)
55      {
56          ia = 100.0f;
57      }
58
59      if (!ipred)
60      {
61          ib = 200.0f;
62      }
63
64      c[tid] = ia + ib;
65  }
66
67  __global__ void mathKernel4(float *c)
68  {
69      int tid = blockIdx.x * blockDim.x + threadIdx.x;
70      float ia, ib;
```

```
71      ia = ib = 0.0f;
72
73      int itid = tid >> 5;
74
75      if (itid & 0x01 == 0)
76      {
77          ia = 100.0f;
78      }
79      else
80      {
81          ib = 200.0f;
82      }
83
84      c[tid] = ia + ib;
85  }
86
87  __global__ void warmingup(float *c)
88  {
89      int tid = blockIdx.x * blockDim.x + threadIdx.x;
90      float ia, ib;
91      ia = ib = 0.0f;
92
93      if ((tid / warpSize) % 2 == 0)
94      {
95          ia = 100.0f;
96      }
97      else
98      {
99          ib = 200.0f;
100     }
101
102     c[tid] = ia + ib;
103 }
104
105
```

```
106 int main(int argc, char **argv)
107 {
108     // set up device
109     int dev = 0;
110     cudaDeviceProp deviceProp;
111     CHECK(cudaGetDeviceProperties(&deviceProp, dev));
112     printf("%s using Device %d: %s\n", argv[0], dev, deviceProp.name);
113 
114     // set up data size
115     int size = 64;
116     int blocksize = 64;
117 
118     if(argc > 1) blocksize = atoi(argv[1]);
119 
120     if(argc > 2) size      = atoi(argv[2]);
121 
122     printf("Data size %d ", size);
123 
124     // set up execution configuration
125     dim3 block (blocksize, 1);
126     dim3 grid  ((size + block.x - 1) / block.x, 1);
127     printf("Execution Configure (block %d grid %d)\n", block.x, grid.x);
128 
129     // allocate gpu memory
130     float *d_C;
131     size_t nBytes = size * sizeof(float);
132     CHECK(cudaMalloc((float**)&d_C, nBytes));
133 
134     // run a warmup kernel to remove overhead
135     size_t iStart, iElaps;
136     CHECK(cudaDeviceSynchronize());
137     iStart = seconds();
138     warmingup<<<grid, block>>>(d_C);
139     CHECK(cudaDeviceSynchronize());
140     iElaps = seconds() - iStart;
```

```
141     printf("warmup      <<< %4d %4d >>> elapsed %d sec \n", grid.x, block.x,
142            iElaps );
143     CHECK(cudaGetLastError());
144
145     // run kernel 1
146     iStart = seconds();
147     mathKernel1<<<grid, block>>>(d_C);
148     CHECK(cudaDeviceSynchronize());
149     iElaps = seconds() - iStart;
150     printf("mathKernel1 <<< %4d %4d >>> elapsed %d sec \n", grid.x, block.x,
151            iElaps );
152     CHECK(cudaGetLastError());
153
154     // run kernel 3
155     iStart = seconds();
156     mathKernel2<<<grid, block>>>(d_C);
157     CHECK(cudaDeviceSynchronize());
158     iElaps = seconds() - iStart;
159     printf("mathKernel2 <<< %4d %4d >>> elapsed %d sec \n", grid.x, block.x,
160            iElaps );
161     CHECK(cudaGetLastError());
162
163     // run kernel 3
164     iStart = seconds();
165     mathKernel3<<<grid, block>>>(d_C);
166     CHECK(cudaDeviceSynchronize());
167     iElaps = seconds() - iStart;
168     printf("mathKernel3 <<< %4d %4d >>> elapsed %d sec \n", grid.x, block.x,
169            iElaps);
170     CHECK(cudaGetLastError());
171
172     // run kernel 4
173     iStart = seconds();
174     mathKernel4<<<grid, block>>>(d_C);
175     CHECK(cudaDeviceSynchronize());
```

```
176     iElaps = seconds() - iStart;
177     printf("mathKernel4 <<< %4d %4d >>> elapsed %d sec \n", grid.x, block.x,
178            iElaps);
179     CHECK(cudaGetLastError());
180
181     // free gpu memory and reset divece
182     CHECK(cudaFree(d_C));
183     CHECK(cudaDeviceReset());
184     return EXIT_SUCCESS;
185 }
186
```