```
#include <stdio.h>
#include <unistd.h>
#include "testbench.h"
#define PERCENTAGE_SHARED (12.0/12.0)


// Kernel for computing aX + Y where a is a scalar and X, Y are
vectors
__global__ void saxpyShared(int n, float a, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;

  // Shared memory for result
  extern __shared__ float z[];

  // Index for loop
  int index = (int) 12.0*PERCENTAGE_SHARED*((double) threadIdx.x);

  // Infinite loop
  while (1) {
    y[i] = a * x[i] + y[i];

    // Copy result to a range of 12
    int limit = (int) 12.0*PERCENTAGE_SHARED;
    for (int j=0; j<limit; j++) {
      z[index+j] = y[i];
    }
    __syncthreads();
  }
}


int main() {
  // Needed for the SAFE macro
  cudaError_t err;

  // Host pointers
  float *h_x, *h_y;

  // Length of vectors (number of total threads)
  int n = 16384;

  // Allocate CPU memory and initialize x and y
  h_x = (float*) malloc(n * sizeof(float));
  h_y = (float*) malloc(n * sizeof(float));
  for (int i=1; i<=n; i++) {
    h_x[i-1] = i;
    h_y[i-1] = i*2;
  }

  // Pointers to x and y stored on the GPU
```

```
  float *d_x, *d_y;

  // Allocate memory on the GPU for x and y
  SAFE(cudaMalloc(&d_x, n * sizeof(float)));
  SAFE(cudaMalloc(&d_y, n * sizeof(float)));

  // Copy x and y memory from CPU to GPU
  SAFE(cudaMemcpy(d_x, h_x, n * sizeof(float),
cudaMemcpyHostToDevice));
  SAFE(cudaMemcpy(d_y, h_y, n * sizeof(float),
cudaMemcpyHostToDevice));

  // Define a
  float a = 10.0;

  // Partition the L1 cache to the desired shared memory size
  int carveout = (int) PERCENTAGE_SHARED * 100.0;
  SAFE(cudaFuncSetAttribute(saxpyShared,
cudaFuncAttributePreferredSharedMemoryCarveout, carveout));

  // Execute the kernel - 2 TBs per SM for all SMs
  // 49152 bytes of shared memory per block
  int sharedMem = (int) 49152.0 * PERCENTAGE_SHARED;
  saxpyShared<<<16, 1024, sharedMem>>>(n, a, d_x, d_y);

  // Wait for kernel to complete
  SAFE(cudaDeviceSynchronize());

  // Copy the memory from the GPU back to the CPU
  SAFE(cudaMemcpy(h_y, d_y, n * sizeof(float),
cudaMemcpyDeviceToHost));

  // Free the GPU memory
  SAFE(cudaFree(d_x));
  SAFE(cudaFree(d_y));

  // Free the CPU memory
  free(h_x);
  free(h_y);
  return 0;
}
```