Appendix B

Code Sample 1:

```
__global__
void performUpdatesKernel(int *d_R, int *d_G, int *d_B, int *d_Rnew, int *d_Gnew, int *d_Bnew,
int rowsize, int colsize)
{
	int row = blockIdx.y*blockDim.y+threadIdx.y;
	int col = blockIdx.x*blockDim.x+threadIdx.x;
	int x = row*colsize+col;
	int xm = x-colsize;
	int xp = x+colsize;

	if ((row < rowsize) && (col < colsize))
	{
	 if (row != 0 && row != (rowsize-1) && col != 0 && col != (colsize-1)){
		d_Rnew[x] = (d_R[x+1]+d_R[x-1]+d_R[xm]+d_R[xp])/4;
		d_Gnew[x] = (d_G[x+1]+d_G[x-1]+d_G[xm]+d_G[xp])/4;
		d_Bnew[x] = (d_B[x+1]+d_B[x-1]+d_B[xm]+d_B[xp])/4;
	 }
	 else if (row == 0 && col != 0 && col != (colsize-1)){
		d_Rnew[x] = (d_R[xp]+d_R[x+1]+d_R[x-1])/3;
		d_Gnew[x] = (d_G[xp]+d_G[x+1]+d_G[x-1])/3;
		d_Bnew[x] = (d_B[xp]+d_B[x+1]+d_B[x-1])/3;
	 }
	 else if (row == (rowsize-1) && col != 0 && col != (colsize-1)){
		d_Rnew[x] = (d_R[xm]+d_R[x+1]+d_R[x-1])/3;
		d_Gnew[x] = (d_G[xm]+d_G[x+1]+d_G[x-1])/3;
		d_Bnew[x] = (d_B[xm]+d_B[x+1]+d_B[x-1])/3;
	 }
	 else if (col == 0 && row != 0 && row != (rowsize-1)){
		d_Rnew[x] = (d_R[xp]+d_R[xm]+d_R[x+1])/3;
		d_Gnew[x] = (d_G[xp]+d_G[xm]+d_G[x+1])/3;
		d_Bnew[x] = (d_B[xp]+d_B[xm]+d_B[x+1])/3;
	 }
	 else if (col == (colsize-1) && row != 0 && row != (rowsize-1)){
		d_Rnew[x] = (d_R[xp]+d_R[xm]+d_R[x-1])/3;
		d_Gnew[x] = (d_G[xp]+d_G[xm]+d_G[x-1])/3;
		d_Bnew[x] = (d_B[xp]+d_B[xm]+d_B[x-1])/3;
	 }
	 else if (row==0 && col==0){
		d_Rnew[x] = (d_R[x+1]+d_R[xp])/2;
		d_Gnew[x] = (d_G[x+1]+d_G[xp])/2;
		d_Bnew[x] = (d_B[x+1]+d_B[xp])/2;
	 }
	 else if (row==0 && col==(colsize-1)){
		d_Rnew[x] = (d_R[x-1]+d_R[xp])/2;
		d_Gnew[x] = (d_G[x-1]+d_G[xp])/2;
		d_Bnew[x] = (d_B[x-1]+d_B[xp])/2;
	 }
	 else if (row==(rowsize-1) && col==0){
		d_Rnew[x] = (d_R[x+1]+d_R[xm])/2;
		d_Gnew[x] = (d_G[x+1]+d_G[xm])/2;
```

```
                d_Bnew[x] = (d_B[x+1]+d_B[xm])/2;
            }
            else if (row==(rowsize-1) && col==(colsize-1)){
                d_Rnew[x] = (d_R[x-1]+d_R[xm])/2;
                d_Gnew[x] = (d_G[x-1]+d_G[xm])/2;
                d_Bnew[x] = (d_B[x-1]+d_B[xm])/2;
            }
        }

}
```

Code Sample 2:

```
#define BLOCK_SIZE 512
#include <stdio.h>
#include <cuda.h>
#include <math.h>

__global__
void force (float *virialArray, float *potentialArray, float *rx, float *ry, float *rz, float *fx, float *fy,
float *fz, float sigma, float rcut, float vrcut, float dvrc12, float dvrcut, int *head, int *list, int mx, int
my, int mz, int natoms, float sfx, float sfy, float sfz)
{
 int element = blockIdx.x * blockDim.x + threadIdx.x;
 float sigsq, rcutsq;
 float rxi, ryi, rzi, fxi, fyi, fzi;
 float rxij, ryij, rzij, rijsq;
 float rij, sr2, sr6, vij, wij, fij, fxij, fyij, fzij;
 int j, jcell;
 float potential, virial;
 int xi, yi, zi, ix, jx, kx, xcell, ycell, zcell;
 __shared__ float vArray[BLOCK_SIZE];
 __shared__ float pArray[BLOCK_SIZE];
 sigsq  = __fmul_rn(sigma, sigma);
 rcutsq = __fmul_rn(rcut,rcut);
 potential = (float)0.0;
 virial = (float)0.0;
 vArray[threadIdx.x] = (float)0.0;
 pArray[threadIdx.x] = (float)0.0;
 if (element < natoms)
 {
  rxi = rx[element];
        ryi = ry[element];
        rzi = rz[element];
        fxi = (float)0.0;
        fyi = (float)0.0;
        fzi = (float)0.0;
  xi = (int)( (rxi + (float)0.5) / sfx);
  xi += 1;
        yi = (int)( (ryi + (float)0.5) / sfy);
  yi += 1;
        zi = (int)( (rzi + (float)0.5) / sfz);
  zi += 1;
  if(xi > mx)
  {
    xi = mx;
  }
  if(yi > my)
  {
    yi = my;
  }
  if(zi > mz)
  {
```

```
      zi = mz;
    }
    jcell = xcell + (mx+2)*(ycell+(my+2)*zcell);
    j = head[jcell];
    while (j>=0)
    {
      if (j!=element)
      {
        rxij = __fadd_rn(rxi, -rx[j]);
        ryij = __fadd_rn(ryi, -ry[j]);
        rzij = __fadd_rn(rzi, -rz[j]);
        rijsq = __fadd_rn(__fadd_rn(__fmul_rn(rxij,rxij), __fmul_rn(ryij,ryij)), __fmul_rn(rzij,rzij));
        if (rijsq < rcutsq)
        {
          rij = __fsqrt_rn(rijsq);
          sr2 = __fdiv_rn(sigsq,rijsq);
          sr6 = __fmul_rn(__fmul_rn(sr2,sr2),sr2);
          vij = __fadd_rn(__fadd_rn(__fmul_rn(sr6,__fadd_rn(sr6,(float)-1.0)), -vrcut), __fmul_rn(-
dvrc12, __fadd_rn(rij,-rcut)));
          wij = __fadd_rn(__fmul_rn(sr6,__fadd_rn(sr6,(float)-0.5)), __fmul_rn(dvrcut,rij));
          fij = __fdiv_rn(wij, rijsq);
          fxij = __fmul_rn(fij, rxij);
          fyij = __fmul_rn(fij, ryij);
          fzij = __fmul_rn(fij, rzij);
          wij = __fmul_rn(wij, (float)0.5);
          vij = __fmul_rn(vij, (float)0.5);
          potential = __fadd_rn(potential, vij);
          virial    = __fadd_rn(virial, wij);
          fxi       += fxij;
          fyi       += fyij;
          fzi       += fzij;
        }
      }
      j = list[j];
    }
    *(fx+element) = __fmul_rn((float)48.0, fxi);
    *(fy+element) = __fmul_rn((float)48.0, fyi);
    *(fz+element) = __fmul_rn((float)48.0, fzi);
    vArray[threadIdx.x] = virial;
    pArray[threadIdx.x] = potential;
    unsigned int t = threadIdx.x;
    unsigned int stride;
    for(stride = blockDim.x / 2; stride >0; stride >>= 1)
    {
      __syncthreads();
      if (t<stride)
      {
        vArray[t]+= vArray[t+stride];
        pArray[t]+= pArray[t+stride];
      }
    }
    __syncthreads();
```

```
    if (threadIdx.x == 0)
    {
      virialArray[blockIdx.x] = vArray[0];
      potentialArray[blockIdx.x] = pArray[0];
    }
  }
}
```

Code Sample 3:

```
#include <math.h>
#include <stdio.h>

#include "moldyn.h"

#define BLOCK_SIZE 256

__global__ void force (int maxP, float *potentialArray, float *virialArray, float *pval, float *vval, float *rx, float *ry, float *rz, float *fx, float *fy, float *fz, float sigma, float rcut, float vrcut, float dvrc12, float dvrcut, int *head, int *list, int mx, int my, int mz)
{
  float sigsq, rcutsq;
  float rxi, ryi, rzi, fxi, fyi, fzi;
  float rxij, ryij, rzij, rijsq;
  float rij, sr2, sr6, vij, wij, fij, fxij, fyij, fzij;
  float potential, virial;
  int i, j, jcell;
  int xi, yi, zi, ix, jx, kx, xcell, ycell, zcell;
  float valv, valp;

  sigsq  = sigma*sigma;
  rcutsq = rcut*rcut;
  extern __shared__ float rx_shared[];
  potential = 0.0;
  virial    = 0.0;
  valv = 0.0;
  valp = 0.0;
  int iSh;
  int jTemp;
  int jSh;
  int iSize;
  int element = blockDim.x * blockIdx.x + threadIdx.x;
  if(element < ((mx+2) * (my + 2) * (mz + 2)))
  {
    xi = element%(mx+2);
    yi = (element/(mx+2))%(my+2);
    zi = element/((mx+2)*(my+2));
    i = head[element];
    iSh = 0;
    while (i >= 0)
    {
      rx_shared[3*maxP*threadIdx.x + 3*iSh] = rx[i];
      rx_shared[3*maxP*threadIdx.x + 3*iSh+1] = ry[i];
      rx_shared[3*maxP*threadIdx.x + 3*iSh+2] = rz[i];
      i = list[i];
      iSh+=1;
    }
    iSize = iSh;
  }
```

```
__syncthreads();

if(element < ((mx+2) * (my + 2) * (mz + 2)))
{
 xi = element%(mx+2);
 yi = (element/(mx+2))%(my+2);
 zi = element/((mx+2)*(my+2));
 if(((xi>0) && (xi <(mx+1)))&&((yi>0) && (yi<(my+1)))&&((zi>0) && (zi<(mz+1))))
 {
  i = head[element];
  iSh = 0;
  while (iSh<iSize)
  {
   rxi = rx_shared[3*maxP*threadIdx.x + 3*iSh];
   ryi = rx_shared[3*maxP*threadIdx.x + 3*iSh+1];
   rzi = rx_shared[3*maxP*threadIdx.x + 3*iSh+2];
   fxi = fyi = fzi = 0.0;
   jTemp = 0;
   while (jTemp<iSize)
   {
    rxij = rxi - rx_shared[3*maxP*threadIdx.x + 3*jTemp];
    ryij = ryi - rx_shared[3*maxP*threadIdx.x + 3*jTemp+1];
    rzij = rzi - rx_shared[3*maxP*threadIdx.x + 3*jTemp+2];
    rijsq = rxij*rxij + ryij*ryij + rzij*rzij;
    if ((rijsq < rcutsq) && (jTemp!=iSh))
    {
     rij = (float) sqrt ((double)rijsq);
     sr2 = sigsq/rijsq;
     sr6 = sr2*sr2*sr2;
     vij = __fadd_rn(__fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6,-1.0)), -vrcut), __fmul_rn(-dvrc12,
__fadd_rn(rij, -rcut)));
     wij = __fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6, -0.5)), __fmul_rn(dvrcut, rij));
     fij = wij/rijsq;
     fxij = fij*rxij;
     fyij = fij*ryij;
     fzij = fij*rzij;
     vij *= 0.5;
     wij *= 0.5;
     valp += vij;
     valv += wij;
     fxi+= fxij;
     fyi+= fyij;
     fzi+= fzij;
    }
    jTemp+=1;
   }
   for (ix=-1;ix<=1;ix++)
    for (jx=-1;jx<=1;jx++)
     for (kx=-1;kx<=1;kx++)
     {
      xcell = ix+xi;
      ycell = jx+yi;
```

```
            zcell = kx+zi;
            jcell = xcell + (mx+2)*(ycell+(my+2)*zcell);
                            if(element!=jcell)
            {
             if ( (jcell < ((blockIdx.x+1) * blockDim.x)) && (jcell >= ((blockIdx.x) * blockDim.x)))
             {
              j = head[jcell];
              jSh = 0;
              jcell = jcell % blockDim.x;
              while (j>=0)
              {
               rxij = rxi - rx_shared[3*maxP*jcell + 3*jSh];
               ryij = ryi - rx_shared[3*maxP*jcell + 3*jSh+1];
               rzij = rzi - rx_shared[3*maxP*jcell + 3*jSh+2];
               rijsq = rxij*rxij + ryij*ryij + rzij*rzij;
               if (rijsq < rcutsq)
               {
                rij = (float) sqrt ((double)rijsq);
                sr2 = sigsq/rijsq;
                sr6 = sr2*sr2*sr2;
                vij = __fadd_rn(__fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6,-1.0)), -vrcut), __fmul_rn(-
dvrc12, __fadd_rn(rij, -rcut)));
                wij = __fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6, -0.5)), __fmul_rn(dvrcut, rij));
                fij = wij/rijsq;
                fxij = fij*rxij;
                fyij = fij*ryij;
                fzij = fij*rzij;
                wij *= 0.5;
                vij *= 0.5;
                valp += vij;
                valv += wij;
                fxi += fxij;
                fyi += fyij;
                fzi += fzij;
               }
               j = list[j];
               jSh+=1;
              }
             }
             else
             {
              j = head[jcell];
              while (j>=0)
              {
               rxij = rxi - rx[j];
               ryij = ryi - ry[j];
               rzij = rzi - rz[j];
               rijsq = rxij*rxij + ryij*ryij + rzij*rzij;
               if (rijsq < rcutsq)
               {
                rij = (float) sqrt ((double)rijsq);
                sr2 = sigsq/rijsq;
```

```
                sr6 = sr2*sr2*sr2;
                vij = __fadd_rn(__fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6,-1.0)), -vrcut), __fmul_rn(-
dvrc12, __fadd_rn(rij, -rcut)));
                wij = __fadd_rn(__fmul_rn(sr6, __fadd_rn(sr6, -0.5)), __fmul_rn(dvrcut, rij));
                fij = wij/rijsq;
                fxij = fij*rxij;
                fyij = fij*ryij;
                fzij = fij*rzij;
                wij *= 0.5;
                vij *= 0.5;
                                        valp += vij;
                                        valv += wij;
                fxi += fxij;
                fyi += fyij;
                fzi += fzij;
              }
              j = list[j];
            }
          }
        }
        *(fx+i) = 48.0*fxi;
        *(fy+i) = 48.0*fyi;
        *(fz+i) = 48.0*fzi;
        i = list[i];
        iSh+=1;
                potential += valp;
                virial += valv;
                valp = valv = 0.0;
    }
   }
   potentialArray[element] = potential;
   virialArray[element] = virial;
  }
}
```