#define BDX 8

#define BDY 8

#define BDZ 8

#define REPEAT\_THREAD 4

// BDX\* BDY\* BDZ = 512

#define BTN (BDX\* BDY\* BDZ)

#define PIXELS\_SPAN\_BLOCK (BDX \* REPEAT\_THREAD)

#define MAX\_NB\_SIDE\_SPAN 16

#define GRAYBIN\_NUM 64

// (MAX\_NB\_SIDE\_SPAN + PIXELS\_SPAN\_BLOCK + MAX\_NB\_SIDE\_SPAN )

#define IMG\_SMEM\_SPAN 64

// IMG\_SMEM\_SPAN / 8

#define IMG\_SMEM\_SPAN\_lint 8

#define IMG\_POSI\_OFFSET neighborsSpan

#define GRAY\_RESULTION 4

\_\_kernel\_\_ clusterLocalGray(int workIndex )

{

Work\* protWork = &protWorks[workIndex];

unsiged char\* normalizedImg = protWork -> normalizedImage->imgData;

unsiged char\* testImage = protWork-> testImage->imgData;

unsiged char neighborsSideSpan = protWork -> locallyGrayClusterSideSpan; // < MAX\_NB\_SIDE\_SPAN

unsiged char neighborsSpan = neighborsSideSpan << 1 + 1;

unsigned short neighborsArea = neighborsSpan \* neighborsSpan;

unsigned char spanOfESMS = neighborsSideSpan << 1 + PIXELS\_SPAN\_BLOCK;

unsigned char spanOfESMS\_lint = spanOfESMS >> 3;

unsigned char sizeOfESMS\_lint = spanOfESMS\_lint \* spanOfESMS\_lint;

int tx = threadIdx.x;

int ty = threadIdx.y;

int tz = threadIdx.z;

int bx = blockIdx.x;

int by = blockIdx.y;

int imgWidth = protWork->imgWidth;

int imgHeight = protWork->imgHeight;

unsigned char hGrayPercentTh = protWork-> hGrayPercentThCLG;

unsigned char lGrayPercentTh = protWork-> lGrayPercentThCLG;

\_\_shared\_\_ unsigned short c[BDY][BDX][GRAYBIN\_NUM]; // 将图像灰度分成GRAYBIN\_NUM 个bin. i.e., gray value的量子化单位为256 / GRAYBIN\_NUM .

// 因为据说以8byte的形式读入global data最快, 故采用long long int形式的shared memory. ------应做别的形式的实验!

\_\_shared\_\_ long long int imgSharedMem[IMG\_SMEM\_SPAN\_lint][IMG\_SMEM\_SPAN\_lint];

// 下面的这个\_\_shared\_\_ 是通常的image buffer------应做相应的实验!

两种shared memory定义对应两种global memory loading法.

应比较loading速度.

// \_\_shared\_\_ unsiged char imgSharedMem[IMG\_SMEM\_SPAN][IMG\_SMEM\_SPAN];

unsigned char\* testImageB = ((unsigned char\*) imgSharedMem);

unsigned short efcImgW = imgWidth – neighborsSideSpan – 2;

unsigned short efcImgW\_lint = efcImgW >>3;

unsigned short efcImgH = imgHeight – neighborsSideSpan – 2;

unsigned char startOfESMS = MAX\_NB\_SIDE\_SPAN – neighborsSideSpan;

unsigned char repeatPerTn = (sizeOfESMS\_lint + BTN – 1) / BTN;

unsigned short gImgLoadStartX = PIXELS\_SPAN\_BLOCK \* bx;

unsigned short gImgLoadStartY = PIXELS\_SPAN\_BLOCK \* by;

unsigned short gImgStartX = gImgLoadStartX + IMG\_POSI\_OFFSET;

unsigned short gImgStartY = gImgLoadStartY + IMG\_POSI\_OFFSET;

unsigned short gImgLoadStartX\_lint = gImgLoadStartX >>3; // 1/8

unsigned short gImgLoadStartY\_lint = gImgLoadStartY >>3; // 1/8

unsigned short tn = BDX\*(BDY\*BDZ＋ty)＋tx; // thread index 🡪 thread number, 直列化, for having coalescence of accessing to global memory.

unsigned char repeatPerTz = (neighborsArea + BDZ – 1) / BDZ;

// int statisticCount = repeatPerTz \* BDZ ;

// 下记处理的实际执行次数为: repeatPerTn\*BTN

// 由于shared memory dimension存在冗余行, 故不会有问题.

for ( int tnt = 0; tnt < repeatPerTn; tnt++) { // ★ BDZ个thread 并行 // 此load的逻辑是否正确? 对global memory的读取是否存在非coalescing现象 ?

unsigned short n = BTN\*tnt + tn;

unsigned short tny = n / spanOfESMS\_lint;

if (tny > spanOfESMS) break; // 多余的一行load, for冗余

unsigned short tnx = n % spanOfESMS\_lint;

imgSharedMem[startOfESMS + tny][ startOfESMS\_lint + tnx] = (long long int ) normalizedImg[min( gImgLoadStartY + tny, efcImgH][min( gImgLoadStartX\_lint + tnx, efcImgW\_lint)];

}

\_\_syncthreads( ); // thread同期 in a block

unsiged short\* cp = &c[ty][tx];

unsiged char dx, dy, xc, yc;

#plagma unroll

for ( int tyt = 0; tyt< REPEAT\_THREAD; tyt++ ) {

dy = tyt\*BDY + ty;

int gImgY = gImgStartY + dy + neighborsSideSpan;

if (gImgY >= gImgEndY) break;

#plagma unroll

for ( int txt = 0; txt< REPEAT\_THREAD ; txt++ ) {

int zn;

#plagma unroll // 这种unroll 是否真的实现了?

for ( int tzt = 0; (zn = tzt + tz) < GRAYBIN\_NUM; tzt += BDZ ) { // 是否会造成shared memory bank conflicts?

cp[zn] = 0;

}

\_\_syncthreads( ); // thread同期 in a block

// The relative position of a pixel in a tile a partial of test image.

dx = txt\*BDX + tx;

int gImgX = gImgStartX + dx + neighborsSideSpan;

if ( gImgX >= gImgEndX) break;

// SMEM中的绝对位置

xc = startOfESMS + dx;

yc = startOfESMS + dy;

for ( int tzt = 0; (zn = tzt + tz) < neighborsArea; tzt += BDZ ) { // ★ BDZ个thread 并行

int q = zn / neighborsSpan;

// makes gray histogram

atomicAdd(&cp[testImageB[yc + q ][ xc + zn – q\* neighborsSpan] >>2 ], 1); // >>2 : GRAY\_RESULTION

}

\_\_syncthreads( ); // thread同期 in a block

clusterPixelGray(testImageB[ yc + neighborsSideSpan][ xc + neighborsSideSpan],

cp,

neighborsArea,

lGrayPercentTh,

hGrayPercentTh,

&testImage[gImgY][ gImgX]); // test image中的绝对位置

} // #plagma unroll

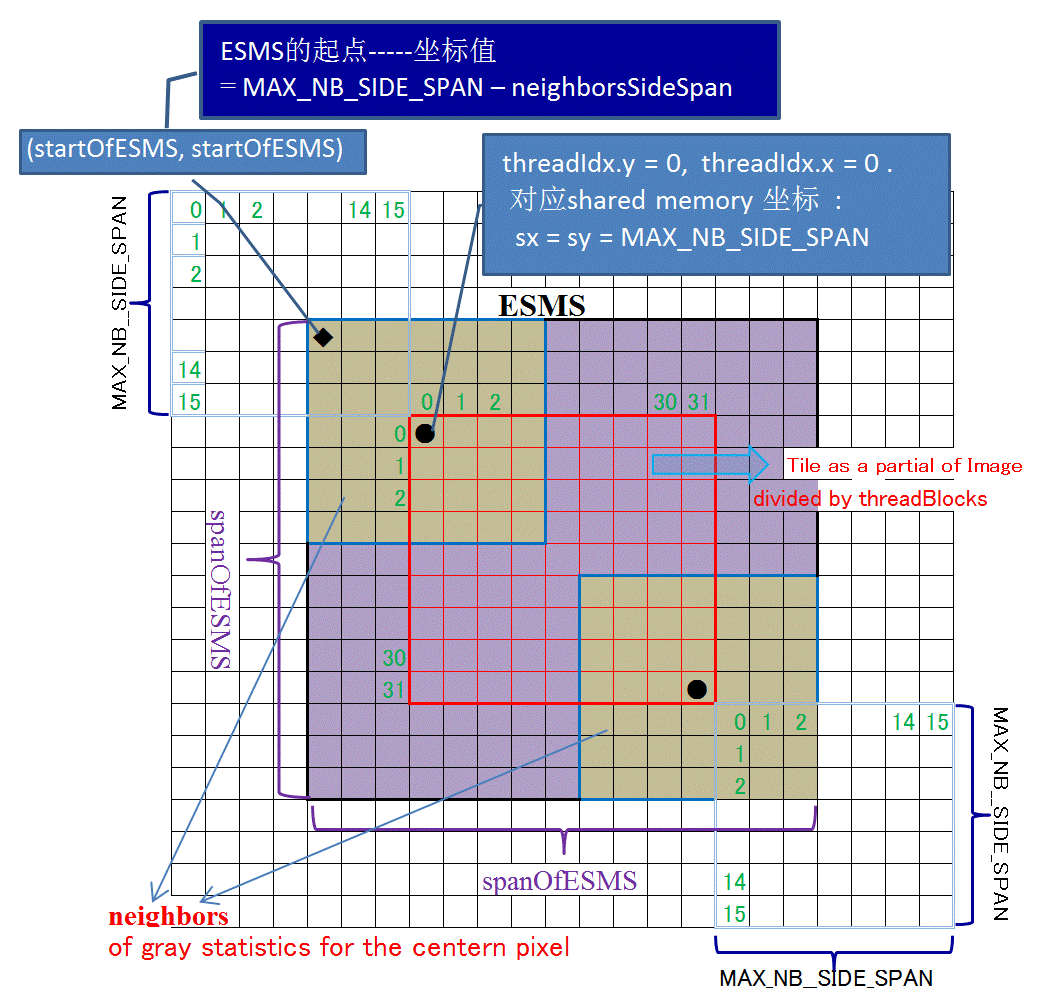
} // #plagma unroll

}

**下图为image shared memory (imgSharedMem)的dimension =** IMG\_SMEM\_SPAN \* IMG\_SMEM\_SPAN.

红色区域为1block应处理的pixel范围; **紫色**区域为红色区域中各个pixel所对应的square(local statistics area)之并集----

ESMS : Effective Shared Memory Square for one threaBlock.



**image shared memory (imgSharedMem)**

/////// \_\_forceinline\_\_ / \_\_device\_\_ ------哪个更合适, 更有利于提高速度?

\_\_forceinline\_\_ / \_\_device\_\_ clusterPixelGray(unsigned char gray, // original gray of the center pixel wicth will be regularized.

unsigned short\* c,

unsigned short neighborsArea,

unsigned char hGrayPercentTh,

unsigned char lGrayPercentTh,

unsigned char\* pixel)

{

int m;

int gSum;

m=0;

gSum=0;

int lNumTh = lGrayPercentTh\* neighborsArea/100;

for ( int n=0; m < lNumTh && n < GRAYBIN\_NUM; n++) {

m += c[n];

gSum += n\*c[n]; // 灰度累积;

}

unsigned char aveLg = (unsigned char)(( GRAY\_RESULTION \*gSum/m) + GRAY\_RESULTION>>1 + 0.5f); // + GRAY\_RESULTION>>1: debias ---- to adjust the middle value of each gray bin.

m=0;

gSum=0;

int hNumTh = hGrayPercentTh\* neighborsArea/100;

for ( int n= GRAYBIN\_NUM-1; m < hNumTh && n> –1; n--) {

m += c[n];

gSum += n\*c[n]; (unsigned char)(( GRAY\_RESULTION \*gSum/m) + GRAY\_RESULTION>>1); // + GRAY\_RESULTION>>1: debias ---- to adjust the middle value of each gray bin.

}

unsigned char aveHg = (unsigned char)( GRAY\_RESULTION \* gSum/m + GRAY\_RESULTION>>1 + 0.5f);

unsigned char gc = \*pixel; // 当前pixel的gray值.

unsigned char newGray = (aveHg + aveLg) >> 1; // 更新平庸pixel值

if ( aveHg – aveLg > garyGapTh) {

if ( gc > aveHg ) newGray = (unsigned char)(2-gc/255.0f )\*gc; // Enhancing high gray.

else if ( gc < aveLg ) newGray = (unsigned char)gc\*0.8f; // Depressing low gray.

}

\*pixel = newGray;

}