parallels Cheat Sheet

by mtelias2 via cheatography.com/50550/cs/13933/

unroll

```
int H_out = H - K + 1;
int W_out = W - K + 1;
for (int b = 0; b < B; ++b)
for (int c = 0; c < C; ++c) { int
w_base = c (KK); for (int p = 0; p
< K; ++p)
for (int q = 0; q < K; ++q) {
for(inth=0;h< H_out;++h)
for (int w = 0; w < W_out; ++w) {
int w_unroll = w_base + p * K + q;
int h_unroll = h * W_out + w;
X_unroll[b, h_unroll, w_unroll] =
X[b, c, h + p, w + q];
```

replicate

Unroll size CKK x H_out*W_out

Input (/ for rep) C(H_out+K-1)(W_out+K-1)

convo forward

```
int m = blockIdx.x;
int h = blockIdx.y / W_grid +
threadIdx.y;
int w = blockIdx.y % W_grid +
threadIdx.x;
float acc = 0.;
for(intc=0; c<C;c++){
for(intp=0;p<K;p++) KxK filter for
(int q = 0; q < K; q++)acc += X[c,
h + p, w + q] * W[m, c, p, q];
}Y[m, h, w] = acc;
```

matrix

```
int X_tile_width = TILE_WIDTH + K-
1;
extern __shared__ float shmem[];
float* X_shared = &shmem[0];
float W_shared =
&shmem[X_tile_width X_tile_width];
m = blockIdx.x;
```

matrix (cont)

```
h_base = (blockIdx.z / W_grid)
TILE_SIZE; the block w_base =
(blockIdx.z % W_grid) TILE_SIZE; x
= threadIdx.x; ty = threadIdx.y; h
= h_base + tx; w = w_base + ty;
float acc = 0.;
for (c = 0; c < C; c++)
tx and ty used as shorthand for
threadIdx.x and threadIdx.y
if (( ty < K) && ( tx < K))
W_shared[ty, tx]= W [m, c, ty, tx];
__syncthreads();
for (int i = h; i < h_base +
X_tile_width; i += TILE_WIDTH) {
for (int j = w; j < w_base +
X_tile_width; j += TILE_WIDTH)
X_shared[i - h_base, j - w_base] =
X[n, c, i,
j]}__syncthreads();}Y[n, m, h, w] =
acc;
```

Histogram

```
__shared__ unsigned int
histo_private[256];
if (threadIdx.x < 256)
histo_private[threadidx.x] = 0;
__syncthreads();
int i = threadIdx.x + blockIdx.x *
blockDim.x;
int stride = blockDim.x *
gridDim.x; while (i < size) {
atomicAdd( &
(private_histo[buffer[i]), 1);
i += stride; }
__syncthreads();
if (threadIdx.x < 256)
atomicAdd( &(histo[threadIdx.x]),
private_histo[threadIdx.x] );
```

CSR ELL COO JDS JDST

```
int row = blockIdx.x * blockDim.x
+ threadIdx.x;
if (row < num_rows){
float dot = 0;
int row_start = row_ptr[row];
int row_end = row_ptr[row+1];
for (int elem = row_start; elem <
row_end; elem++) {
dot += data[elem] *
x[col_index[elem]]; }
y[row] = dot;
for(inti=0;i<num_elem;i++){
dot +=
data[row+inum_rows]x[col_index[ro
w+i*num_rows]]; y[row] = dot;
for (int i = 0; i < num_elem;
row++)
y[row_index[i]] += data[i] *
x[col_index[i]];
int row_start = jds_row_ptr[row];
int row_end = jds_row_ptr[row+1];
for (int elem = row_start; elem <
row_end; elem++) {
dot += data[elem] *
x[col_index[elem]];
y[jds_row_index[row]] = dot;
unsigned in sec = 0;
while (jds_t_col_ptr[sec+1]-
jds_t_col_ptr[sec] > row){ dot +=
data[jds_t_col_ptr[sec]+row] *
x[col_index[jds_t_col_ptr[sec]+row]
];
   sec++;
y[jds_row_index[row]] = dot;
}
```