Programming Example:
Filter Operations

Hannes Hofmann
19 – 29 March 2006

Outline

- Convolution
- Border conditions
- Partitioning
- 3x3 Convolution
  - Scalar
  - IBM's solution
- How to use IBM's implementation

Convolution

- Continuous

$$(f*g)(u,v)=\int_u \int_v f(x,y)\,g(u-x,v-y)\,dx\,dy$$

- Discrete case

$$c(x,y)=\sum_{i=0}^{2}\sum_{j=0}^{2} f(x+1-i,y+1-j)\,k(i,j)$$

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

Convolution Example

- What are filters used for?

Convolution Example (2)

- What are filters used for?

Gaussian Blur

Smoothing

1	2	1
2	4	2
1	2	1

Convolution Example

- What are filters used for?

First step: Apply contrast agent to improve results

Convolution Example

- What are filters used for?

Sobel horizontal

Edge detection

-1	-2	-1
0	0	0
1	2	1

Convolution Example

- What are filters used for?

Sobel horizontal

Sobel vertical

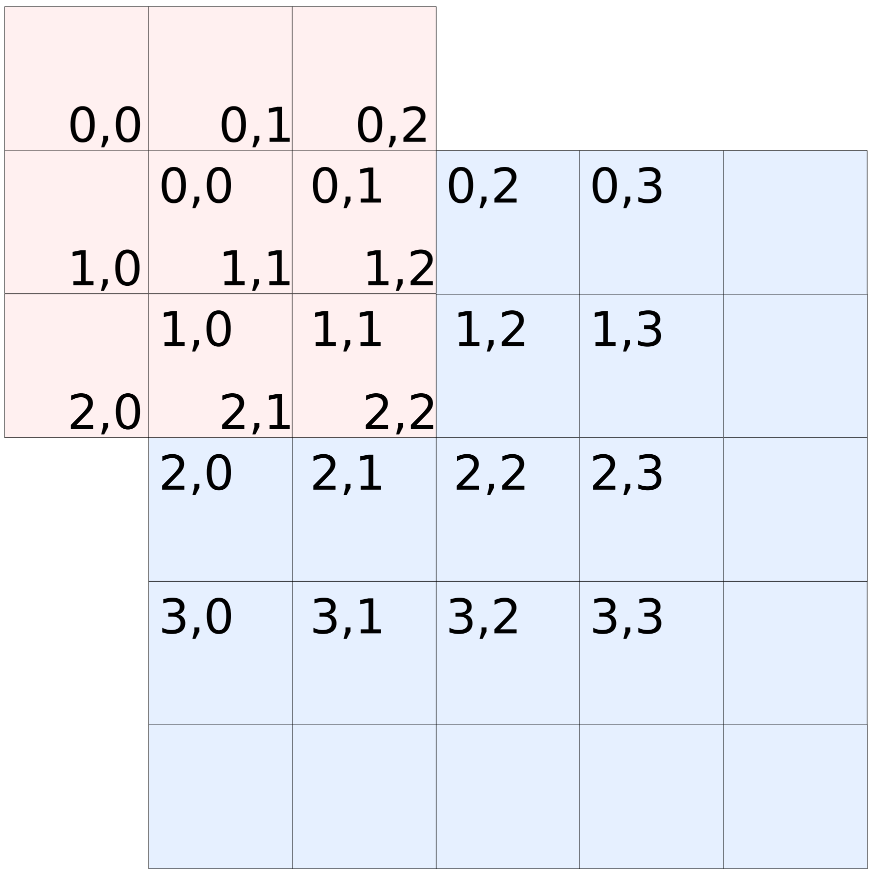
Edge detection

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Border Conditions

- How to compute border pixels?



Handling Border Conditions

- Clamp
  - The border pixels are repeated
- Wrap
  - The opposing border pixel is taken
- Zero
  - Every pixel outside the image is assumed to be 0
- ...

Partitioning

- 1 float = 32 bit
- 512^2 pixels = 1 MB
- Medical images: 1024^2 and more pixels

- Image data to big for Local Store
- Divide the problem into smaller ones that fit into LS

Partitioning

- Border Conditions for each tile
  - Input is bigger than output
  - Partitions need to overlap

Partitioning

- Imagine a multi-level filter that works on images that get smaller on each level
- Decreasing computation need
- What to do with idle SPEs?

Partitioning Strategies

- Static Partitioning
  - Divide your problem by the number of SPEs you want to use
  - Simple and unflexible
- Dynamic Partitioning
  - SPEs request new data when they are finished
- Microtask model
  - Scheduler running on PPU can hand data to SPEs or start new threads for other tasks
  - Flexibel, but has to be implemented by hand

3x3 Convolution, scalar

```
void conv3x3 (const float *in, float *out,
                   const float kern[9], int w_out, int h_out) {
    // assuming in is bigger than out
    int x, y, k;

    for (y=0; y<h_out; y++) {
        for (x=0; x<w_out; x++) {
            for (k=0; k<3; k++) {
                out[x][y] += in[x+k][y] * kern[k];
                out[x][y] += in[x+k][y+1] * kern[k+3];
                out[x][y] += in[x+k][y+2] * kern[k+6];
            }
        }
    }
}
```

9 mul and 9 add operations per pixel

3x3 Convolution, IBM

- Let's have a look at src/lib/image/conv3x3_1f.h
- conv3x3_1f computes one line

```
void conv3x3_1f (const float *in[3], float *out,
                     const vec_float4 m[9], int w);
```

- It takes three pointers to the lines

in[0]	4 36 1 1	5 7 9 11	4 7 1 1	6 9 9 8	1 3 3 7
in[1]	7 6 3 81	1 0 7 37	3 4 1 9	7 2 7 0	3 5 1 8
in[2]	2 5 27 7	6 25 9 3	2 6 9 7	2 5 1 9	5 1 8 2

3x3 Convolution, IBM (2)

```
void _conv3x3_1f (const float *in[3], float *out,
                  const vec_float4 m[9], int w) {

   // init local variables
   const vec_float4 *in0  = (const vec_float4 *)in[0];   // ... in2
   vec_float4 m00 = m[0];                                 // ... m08

   // pre-process
   //   init some pointers to handle left border (_CLAMP_CONV,
   //   _WRAP_CONV)

   // process the line

   // post-process
   //   right border
}
```

Process The Line

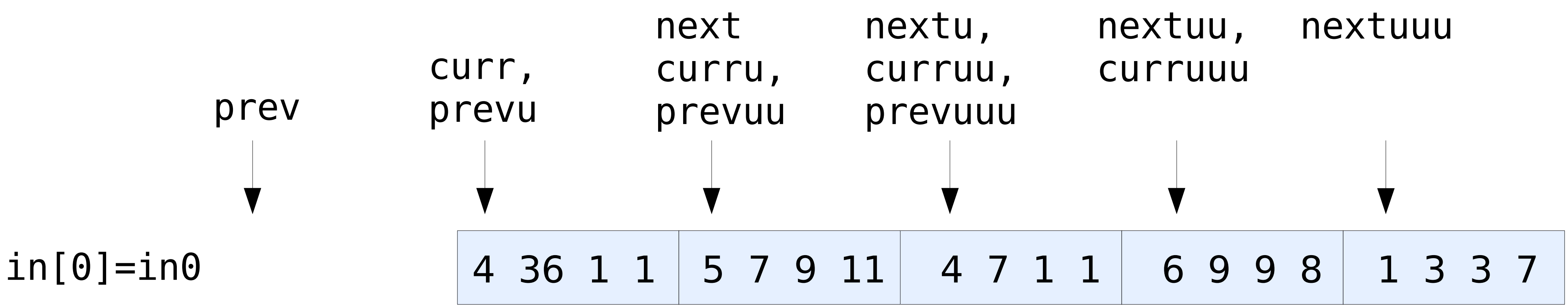
```
for (i0=0, i1=1, i2=2, i3=3, i4=4; i0<(w>>2)-4;
        i0+=4, i1+=4, i2+=4, i3+=4, i4+=4) {

    res = resu = resuu = resuuu = VEC_SPLAT_F32(0.0f);

    _GET_SCANLINE_x4(p0, in0[i0], in0[i1], in0[i2], in0[i3], in0[i4]);
    _CONV3_1f(m00, m01, m02);

    _GET_SCANLINE_x4(p1, in1[i0], in1[i1], in1[i2], in1[i3], in1[i4]);
    _CONV3_1f(m03, m04, m05);

    _GET_SCANLINE_x4(p2, in2[i0], in2[i1], in2[i2], in2[i3], in2[i4]);
    _CONV3_1f(m06, m07, m08);

    vout[i0] = res; vout[i1] = resu;
    vout[i2] = resuu; vout[i3] = resuuu;
} // process line
```

Initialize Local Variables

```
// call:  _GET_SCANLINE_x4(p0, in0[i0], in0[i1],
//                          in0[i2], in0[i3], in0[i4]);

#define _GET_SCANLINE_x4(_p, _a, _b, _c, _d, _e)       \
  prev = _p;                                           \
  curr = prevu = _a;                                   \
  next = curru = prevuu = _b;                          \
  nextu = curruu = prevuuu = _c;                       \
  nextuu = curruuu = _p = _d;                          \
  nextuuu = _e
```

Initialize Local Variables

```
// call:  _GET_SCANLINE_x4(p0, in0[i0], in0[i1],
//                          in0[i2], in0[i3], in0[i4]);

#define _GET_SCANLINE_x4(_p, _a, _b, _c, _d, _e)       \
  prev = _p;                                           \
  curr = prevu = _a;                                   \
  next = curru = prevuu = _b;                          \
  nextu = curruu = prevuuu = _c;                       \
  nextuu = curruuu = _p = _d;                          \
  nextuuu = _e
```



Permutation & Computation

```
// calls: _CONV3_1f(m00, m01, m02);
//        _CONV3_1f(m03, m04, m05);
//        _CONV3_1f(m06, m07, m08);

#define _CONV3_1f(_m0, _m1, _m2)            \
  _GET_x4(prev, curr, left, left_shuf);     \
  _GET_x4(curr, next, right, right_shuf);   \
  _CALC_PIXELS_1f_x4(left, _m0, res);       \
  _CALC_PIXELS_1f_x4(curr, _m1, res);       \
  _CALC_PIXELS_1f_x4(right, _m2, res)
```

Permutation

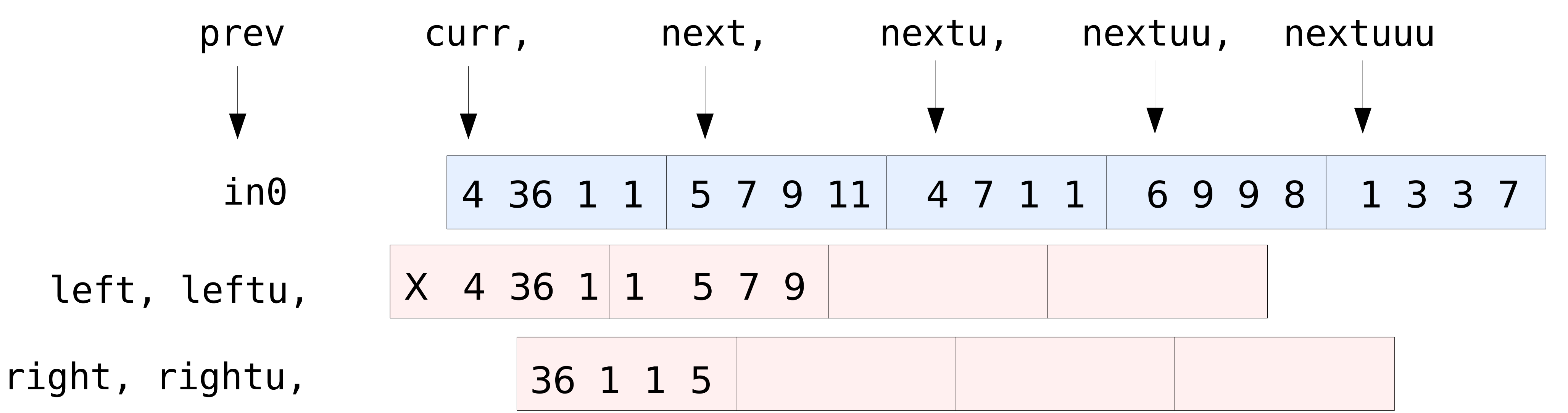
Load values shifted by one pixel left and right

```
// calls: _GET_x4(prev, curr, left, left_shuf);
//        _GET_x4(curr, next, right, right_shuf);

#define _GET_x4(_a, _b, _c, _shuf)                          \
  _c = vec_perm(_a, _b, _shuf);                             \
  _c##u = vec_perm(_a##u, _b##u, _shuf);                    \
  _c##uu = vec_perm(_a##uu, _b##uu, _shuf);                 \
  _c##uuu = vec_perm(_a##uuu, _b##uuu, _shuf)
```

Permutation

Load values shifted by one pixel left and right

```
// calls:  _GET_x4(prev, curr, left, left_shuf);
//         _GET_x4(curr, next, right, right_shuf);

#define _GET_x4(_a, _b, _c, _shuf)                        \
  _c = vec_perm(_a, _b, _shuf);                           \
  _c##u = vec_perm(_a##u, _b##u, _shuf);                  \
  _c##uu = vec_perm(_a##uu, _b##uu, _shuf);               \
  _c##uuu = vec_perm(_a##uuu, _b##uuu, _shuf)
```

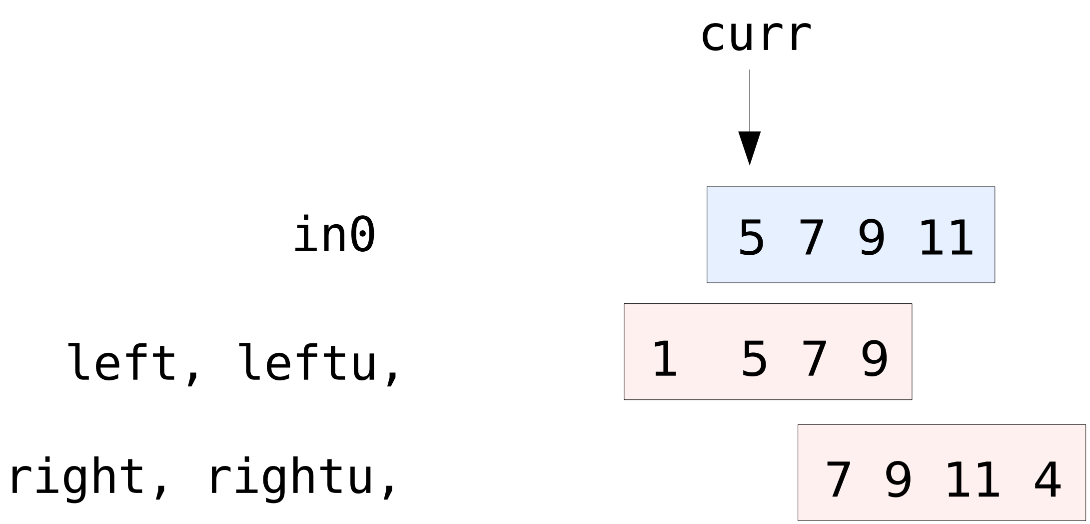


Computation

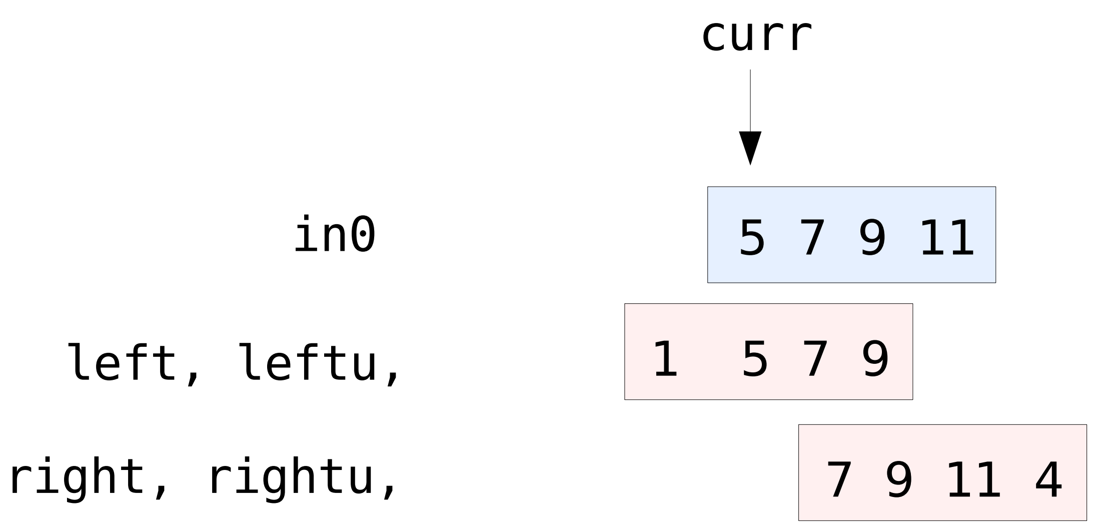
```
// calls: _CALC_PIXELS_1f_x4(left, _m0, res);
//        _CALC_PIXELS_1f_x4(curr, _m1, res);
//        _CALC_PIXELS_1f_x4(right, _m2, res)

#define _CALC_PIXELS_1f_x4(_a, _b, _c)                  \
  _c = vec_madd(_a, _b, _c);                            \
  _c##u = vec_madd(_a##u, _b, _c##u);                   \
  _c##uu = vec_madd(_a##uu, _b, _c##uu);                \
  _c##uuu = vec_madd(_a##uuu, _b, _c##uuu)
```

Computation

```
// calls: _CALC_PIXELS_1f_x4(left, _m0, res);
//        _CALC_PIXELS_1f_x4(curr, _m1, res);
//        _CALC_PIXELS_1f_x4(right, _m2, res)

#define _CALC_PIXELS_1f_x4(_a, _b, _c)                  \
  _c = vec_madd(_a, _b, _c);                            \
  _c##u = vec_madd(_a##u, _b, _c##u);                   \
  _c##uu = vec_madd(_a##uu, _b, _c##uu);                \
  _c##uuu = vec_madd(_a##uuu, _b, _c##uuu)
```



```
// calls:  _CALC_PIXELS_1f_x4(left, _m0, res);
//         _CALC_PIXELS_1f_x4(curr, _m1, res);
//         _CALC_PIXELS_1f_x4(right, _m2, res)

#define _CALC_PIXELS_1f_x4(_a, _b, _c)                  \
  _c = vec_madd(_a, _b, _c);                            \
  _c##u = vec_madd(_a##u, _b, _c##u);                   \
  _c##uu = vec_madd(_a##uu, _b, _c##uu);                \
  _c##uuu = vec_madd(_a##uuu, _b, _c##uuu)
```




```
// in scalar terms:
res += left * m0
       + curr * m1
       + right * m2
```

Rep.: Process The Line

```
for (i0=0, i1=1, i2=2, i3=3, i4=4; i0<(w>>2)-4;
        i0+=4, i1+=4, i2+=4, i3+=4, i4+=4) {

    res = resu = resuu = resuuu = VEC_SPLAT_F32(0.0f);

    _GET_SCANLINE_x4(p0, in0[i0], in0[i1], in0[i2], in0[i3], in0[i4]);
    _CONV3_1f(m00, m01, m02);

// We are here, having mult. 16 pixels with the 1st 3 kernel elements

    _GET_SCANLINE_x4(p1, in1[i0], in1[i1], in1[i2], in1[i3], in1[i4]);
    _CONV3_1f(m03, m04, m05);

    _GET_SCANLINE_x4(p2, in2[i0], in2[i1], in2[i2], in2[i3], in2[i4]);
    _CONV3_1f(m06, m07, m08);

    vout[i0] = res; vout[i1] = resu;
    vout[i2] = resuu; vout[i3] = resuuu;
} // process line
```

Optimizations

- Loop unrolling

```
for (i0=0, i1=1, i2=2, i3=3, i4=4; i0<(w>>2)-4;
       i0+=4, i1+=4, i2+=4, i3+=4, i4+=4) { ... }
```

- Those *u, *uu, *uuu variables

```
#define _GET_SCANLINE_x4(_p, _a, _b, _c, _d, _e)   \
    prev = _p;                                     \
    curr = prevu = _a;                             \
    ...
```

- Vectorization

```
#define _CALC_PIXELS_1f_x4(_a, _b, _c)             \
  _c = vec_madd(_a, _b, _c);                       \
  ...
```

Register Usage (approx.)

- 3x3 Convolution:
  - $6_F+2_S+9_K+6_{SL}+2*4_P+4_R = 35$
- 9x9 Convolution uses only
  - $6_F+6_S+9_K+6_{SL}+6*4_P+4_R = 55$
  - The whole kernel would also fit into registers

F: for loop, S: permutation selectors, K: kernel elements, SL: scanline, P: permuted vectors, R: result

Performance

- 16 pixels multiplied with one kernel row
  - 3*4=12 vec_madd ops (in _CALC_PIXELS)
  - 2*4=8 vec_perm ops (in _GET_x4)

Performance

- 16 pixels multiplied with one kernel row
  - 3*4=12 vec_madd ops (in _CALC_PIXELS)
  - 2*4=8 vec_perm ops (in _GET_x4)
- Assumption: Compiler can interleave madd and perm operations, 36 cycles are needed for the whole 3x3 kernel

Performance

- 36 cycles needed for 3x3 kernel
- Cost on scalar CPU:
  - 16 * 9 (MUL + ADD)
  - 144 Cycles if parallel MUL and ADD
- 144/36 = 4
- Full speedup only if permutes are for free

Static Partitioning

PPU

```
int main(int argc, char *argv[]) {
    float *in, *out;
    int h;
    int part_h = h/NUM_SPES;

    for (i=0; i<NUM_SPES; i++) {
        cmd.in = in+i*part_h*w;
        cmd.h = part_h;
        cmd.w = w;
        cmd.out = out+i*part_h*w;
        ids[i] = spe_create_thread(0, &spu_conv, &cmd, NULL, -1, 0);
    }
    for (i=0; i<NUM_SPES; i++) {
        spe_wait(ids[i], &status, 0);
    }
    return (0);
}
```

SPE Initialization

SPU (or PPU)

```
int main(unsigned long long speid, unsigned long long argv) {
   volatile cmd_t cmd;
   volatile float in[3][MAX_LINE_W];
   float out[MAX_LINE_W];
   const float *ptrs[3];
   float *kern = {1, 2, 1, 0, 0, 0, -1, -2, -1};
   vec_float4 mask[9];
   int next_tag, tag = 0;

   spu_writech(MFC_WrTagMask, 1 << 0);
   spu_mfcdma32((void *)(&cmd), (unsigned int)argv, sizeof(cmd_t),
                0, MFC_GET_CMD);
   spu_mfcstat(2);

   for (j = 0; j < 9; j++)
      mask[j] = splat_float(kern[j]);

   // ...
```

Prefetch For Double Buffering

SPU (or PPU)

```
// continued

// prefetch 3 lines
spu_mfcdma32((void *)(in[0]), (unsigned int)(cmd.in),
                 cmd.w*sizeof(float), tag, MFC_GET_CMD);
// clamping
spu_mfcdma32((void *)(in[1]), (unsigned int)(cmd.in),
                 cmd.w*sizeof(float), tag, MFC_GET_CMD);
cmd.in += cmd.w;
spu_mfcdma32((void *)(in[2]), (unsigned int)(cmd.in),
                 cmd.w*sizeof(float), tag, MFC_GET_CMD);
cmd.in += cmd.w;

ptrs[0] = (const float *)(in[0]);
ptrs[1] = (const float *)(in[1]);
ptrs[2] = (const float *)(in[2]);

// ...
```

Double Buffering

SPU (or PPU)

```
// continued

for (y=3; y<cmd.h+1; y++) {
    next_tag = tag^1;

    // prefetch next line
    ++inBuf;
    if (inBuf >= 4) {
        inBuf = 0;
    }
    spu_mfcdma32((void *)(in[inBuf]), (unsigned int)(cmd.in),
                    cmd.w*sizeof(float), next_tag, MFC_GETB_CMD);
    cmd.in += cmd.w;

    // wait for previous get (and put)
    spu_writech(MFC_WrTagMask, 1 << tag);
    (void)spu_mfcstat(2);

    // ...
```

Compute And Store

SPU (or PPU)

```
        // continued

        // process line
        conv3x3_1f(ptrs, out[tag], mask, cmd.w);

        // write result back
        spu_mfcdma32((void *)(out[tag]), (unsigned int)(cmd.out),
                                    cmd.w*sizeof(float), tag, MFC_PUT_CMD);
        cmd.out += cmd.w;

        // next line
        ptrs[0] = ptrs[1];
        ptrs[1] = ptrs[2];
        ptrs[2] = (const float *) in[inBuf];
        tag = next_tag;
    }
    // process last line
    return (0);
} /* main */
```

Thanks

Thank you for attending.

What questions do you have?

References

- IBM Cell SDK Library Samples (cell-sdk-lib-samples-1.0.1.tar.bz2)