Lecture 17
Case Study: DianNao (Part 2)

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese566/

Class Project Teams

- Team 1
  - Andrew Ellison, Shixuan Zhang
- Team 2
  - Brett Gilpin, Matthew Wedrewer, Nestor Gonzalez
- Team 3
  - Weidong Cao, Liu Ke, Xinyao Li
- Team 4
  - Meizhi Wang, Longzhen Zhang, An Zou
- Team 5
  - Yuyang Li, Yu Liu, Qilan Ding
- Team 6
  - Chenxi Yin, Yuyao Hu
- Team 7
  - Wenmei Bo, Jizhou Huang, Bojun Li

DianNao (ASPLOS 2014)

- ASPLOS
  - ACM International Conference on Architectural Support for Programming Languages and Operating Systems
  - other examples: ISCA, MICRO, HPCA, ISSCC, JSSC, DAC
- Ubiquitous Machine Learning
  - artificial neural network (ANN)
  - convolutional neural network (CNN)
  - deep neural network (DNN)
- High-Throughput
  - large-scale neural network
  - impact of memory

Neural Network Hierarchy

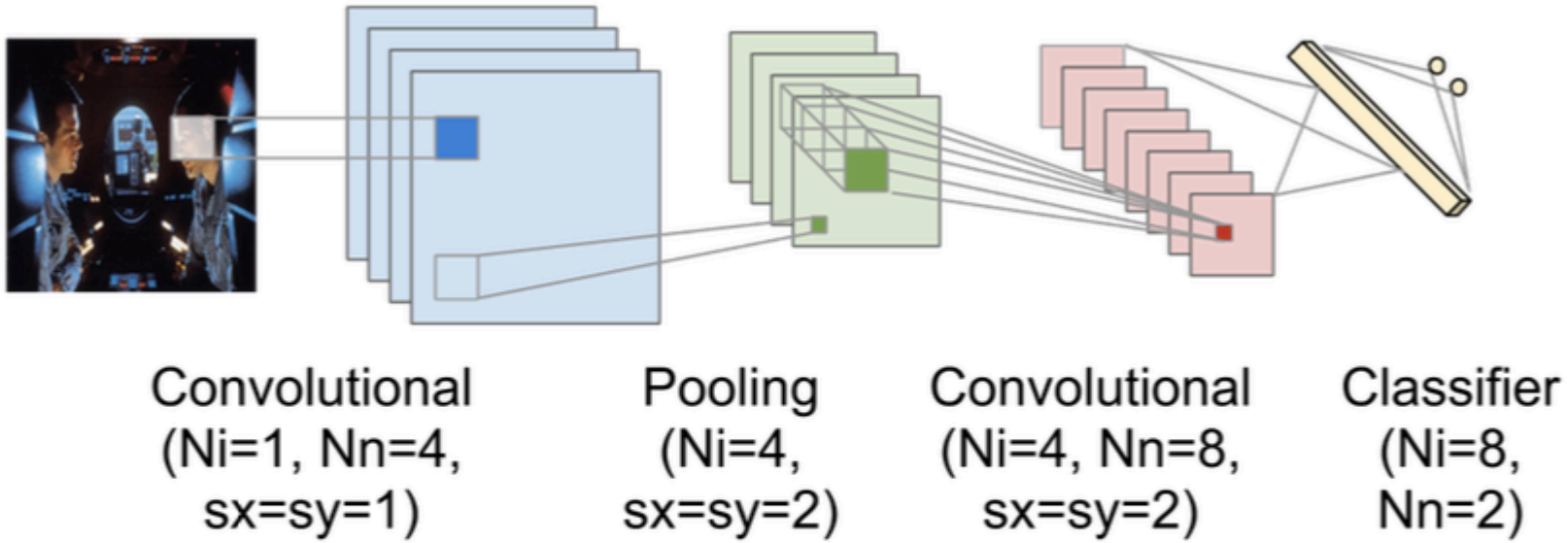

Figure 1. *Neural network hierarchy containing convolutional, pooling and classifier layers.*

```
for (int nnn = 0; nnn ¡ Nn; nnn += Tnn) { // tiling for output neurons;
  for (int iii = 0; iii ¡ Ni; iii += Tii) { // tiling for input neurons;
    for (int nn = nnn; nn ¡ nnn + Tnn; nn += Tn) {
      for (int n = nn; n ¡ nn + Tn; n++)
        sum[n] = 0;
      for (int ii = iii; ii ¡ iii + Tii; ii += Ti)
        // — Original code —
        for (int n = nn; n < nn + Tn; n++)
          for (int i = ii; i < ii + Ti; i++)
            sum[n] += synapse[n][i] * neuron[i];
      for (int n = nn; n < nn + Tn; n++)
        neuron[n] = sigmoid(sum[n]);
} } }
```

Figure 5. *Pseudo-code for a classifier (here, ı (original loop nest + locality optimization).*

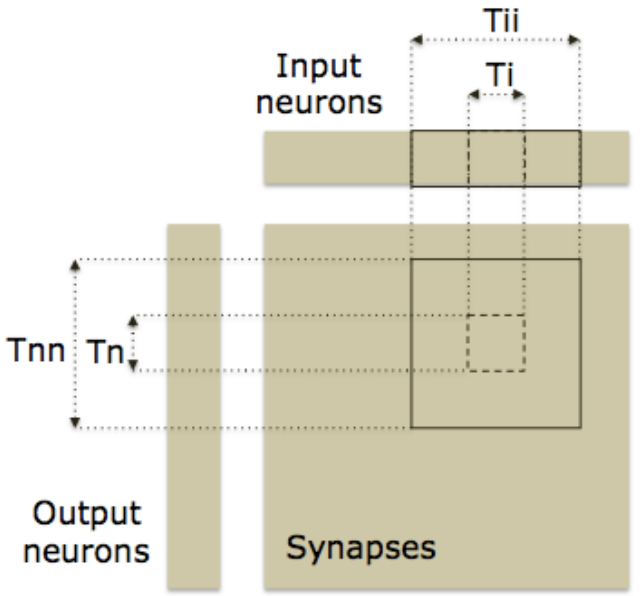

Figure 2. *Classifier layer tiling.*

```
for (int yy = 0; yy ¡ Nyin; yy += Ty)  {
  for (int xx = 0; xx ¡ Nxin; xx += Tx)  {
    for (int nnn = 0; nnn ¡ Nn; nnn += Tnn)  {
      // — Original code — (excluding nn, ii loops)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {  // tiling for y;
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {  // tiling for x;
          for (int nn = nnn; nn < nnn + Tnn; nn += Tn) {
            for (int n = nn; n < nn + Tn; n++)
              sum[n] = 0;
            sliding window;
             (int ky = 0; ky < Ky; ky++)
            or (int kx = 0; kx < Kx; kx++)
             for (int ii = 0; ii < Ni; ii += Ti)
               for (int n = nn; n < nn + Tn; n++)
                 for (int i = ii; i < ii + Ti; i++)
                   // version with shared kernels
                   sum[n] += synapse[ky][kx][n][i]
                             * neuron[ky + y][kx + x][i];
                   // version with private kernels
                   sum[n] += synapse[yout][xout][ky][kx][n][i]}
                             * neuron[ky + y][kx + x][i];
          for (int n = nn; n < nn + Tn; n++)
            neuron[yout][xout][n] = non_linear_transform(sum[n]);
        } xout++; } yout++;
}}}}
```

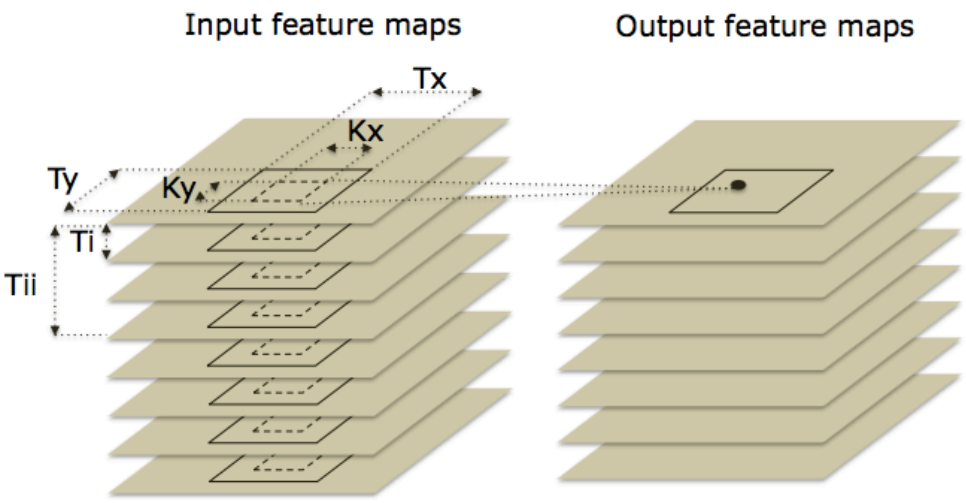

Figure 4. *Pooling layer tiling.*

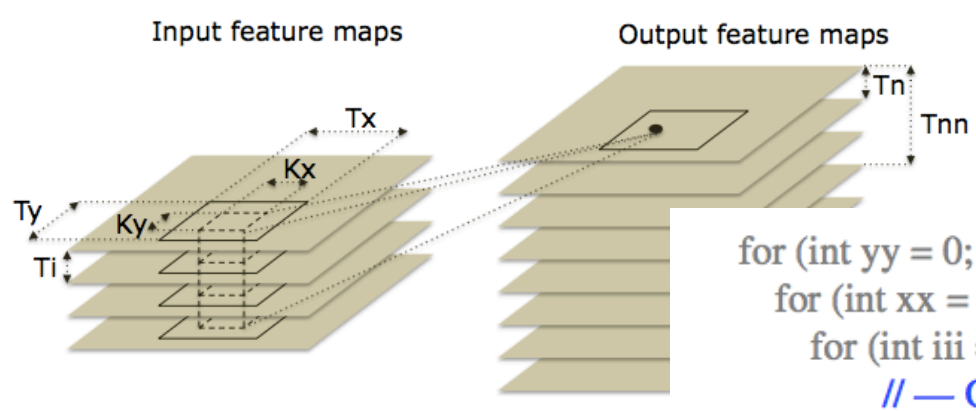

Figure 3. *Convolutional layer tiling.*

```
for (int yy = 0; yy ¡ Nyin; yy += Ty)  {
  for (int xx = 0; xx ¡ Nxin; xx += Tx)  {
    for (int iii = 0; iii ¡ Ni; iii += Tii)
      // — Original code — (excluding ii loop)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {
          for (int ii = iii; ii < iii + Tii; ii += Ti)
            for (int i = ii; i < ii + Ti; i++)
              value[i] = 0;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int i = ii; i < ii + Ti; i++)
                  // version with average pooling;
                  value[i] += neuron[ky + y][kx + x][i];
                  // version with max pooling;
                  value[i] = max(value[i], neuron[ky + y][kx + x][i]);
        } } } }
        // for average pooling;
        neuron[xout][yout][i] = value[i] / (Kx * Ky);
        xout++; } yout++;
  } } }
```

Figure 8. *Pseudo-code for pooling layer (original loop nest + locality optimization).*

Hardware Implementation for Small NN

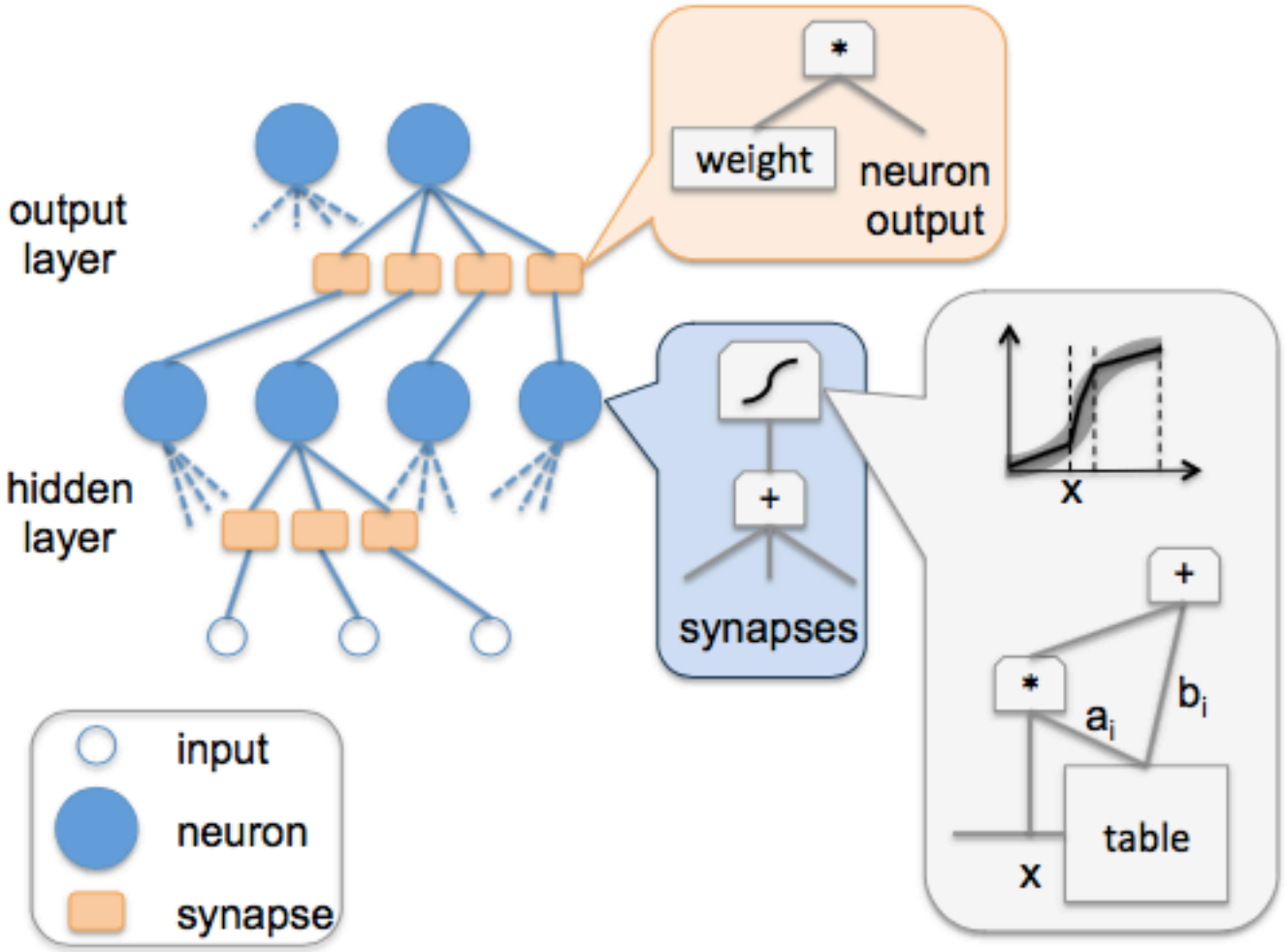

Figure 9. *Full hardware implementation of neural networks.*

Neural Functional Unit (NFU)

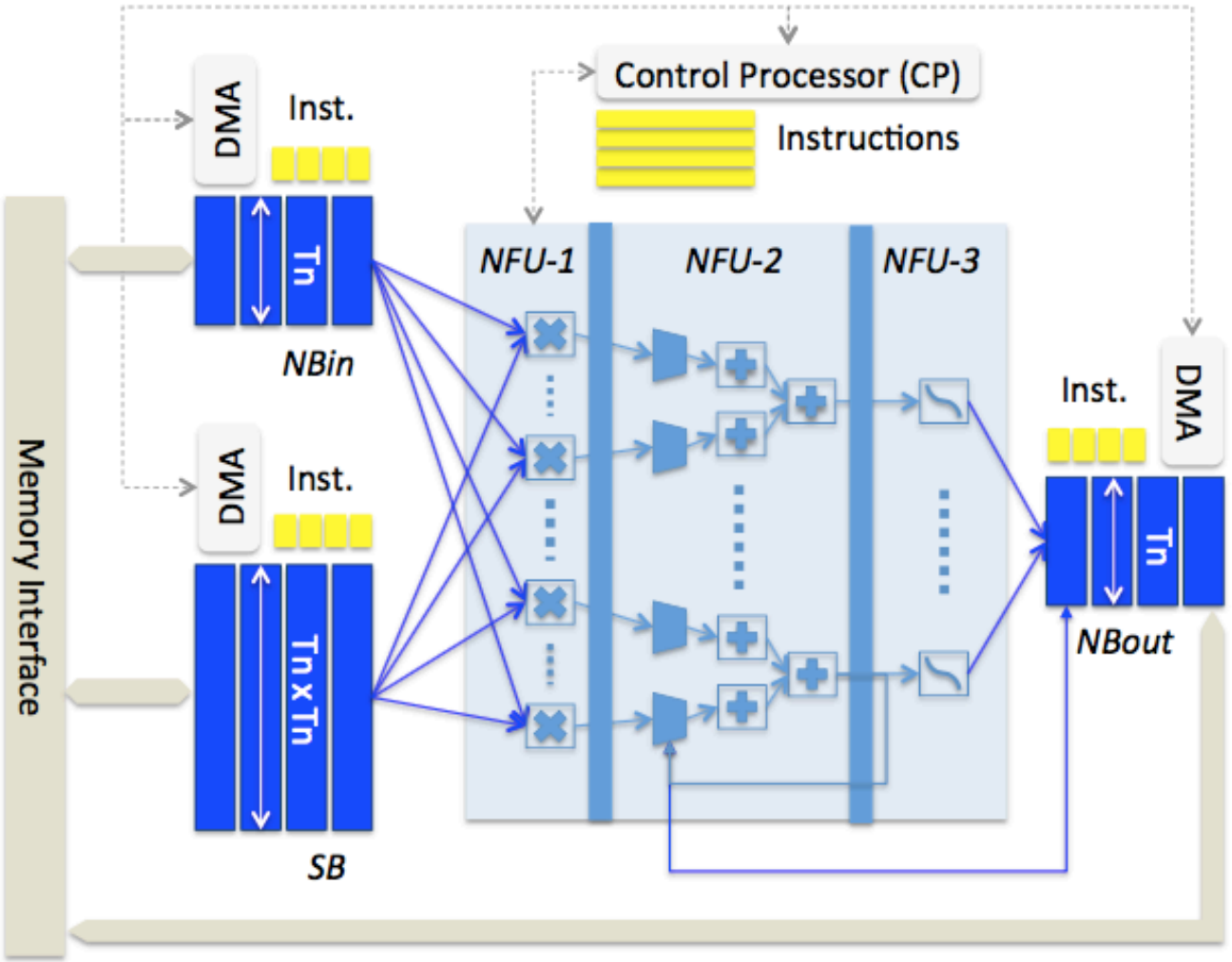

Figure 11. *Accelerator.*

Local Transpose in NBin

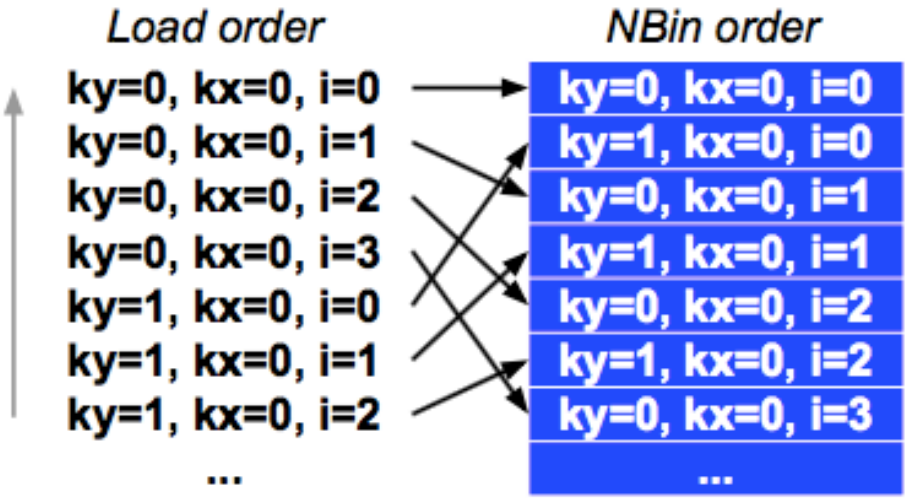

Figure 14. *Local transpose* ($K_y = 2, K_x = 1, N_i = 4$).

Layer Code

CP	SB				NBin							NBout				NFU						
END	READ OP	REUSE	ADDRESS	SIZE	READ OP	REUSE	STRIDE	STRIDE BEGIN	STRIDE END	ADDRESS	SIZE	READ OP	WRITE OP	ADDRESS	SIZE	NFU-1 OP	NFU-2 OP	NFU-2 IN	NFU-2 OUT	NFU-3 OP	OUTPUT BEGIN	OUTPUT END

Table 3. *Control instruction format.*

CP	SB				NBin							NBout				NFU						
NOP	LOAD	0	0	32768	LOAD	1	0	0	0	4194304	2048	NOP	WRITE	0	0	MULT	ADD	RESET	NBOUT	SIGMOID	1	0
NOP	LOAD	0	32768	32768	READ	1	0	0	0	0	0	NOP	WRITE	0	0	MULT	ADD	RESET	NBOUT	SIGMOID	0	0
.....																						
NOP	LOAD	0	7864320	32768	LOAD	1	0	0	0	4225024	2048	READ	STORE	8388608	512	MULT	ADD	NBOUT	NFU3	SIGMOID	1	0
.....																						

Table 4. *Subset of classifier/perceptron code ($N_i = 8192$, $N_o = 256$, $T_n = 16$, 64-entry buffers).*

Area and Power Breakdown

Component or Block	Area in μm^2	(%)	Power in mW	(%)	Critical path in ns
ACCELERATOR	3,023,077		485		1.02
Combinational	608,842	(20.14%)	89	(18.41%)	
Memory	1,158,000	(38.31%)	177	(36.59%)	
Registers	375,882	(12.43%)	86	(17.84%)	
Clock network	68,721	(2.27%)	132	(27.16%)	
Filler cell	811,632	(26.85%)			
SB	1,153,814	(38.17%)	105	(22.65%)	
NBin	427,992	(14.16%)	91	(19.76%)	
NBout	433,906	(14.35%)	92	(19.97%)	
NFU	846,563	(28.00%)	132	(27.22%)	
CP	141,809	(5.69%)	31	(6.39%)	
AXIMUX	9,767	(0.32%)	8	(2.65%)	
Other	9,226	(0.31%)	26	(5.36%)	

Table 6. *Characteristics of accelerator and breakdown by component type (first 5 lines), and functional block (last 7 lines).*

Questions?

Comments?

Discussion?