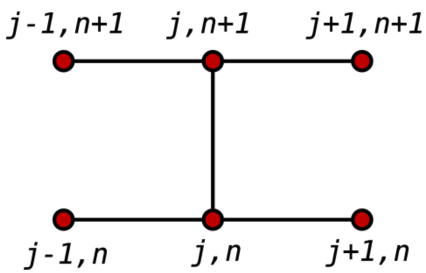
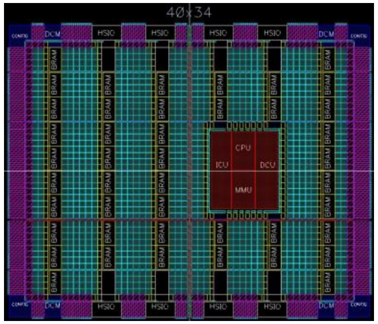
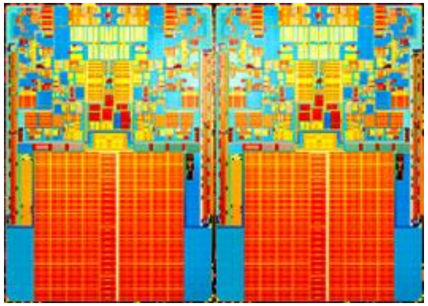
Synthesis of Data-Parallel GPU Software into FPGA Hardware

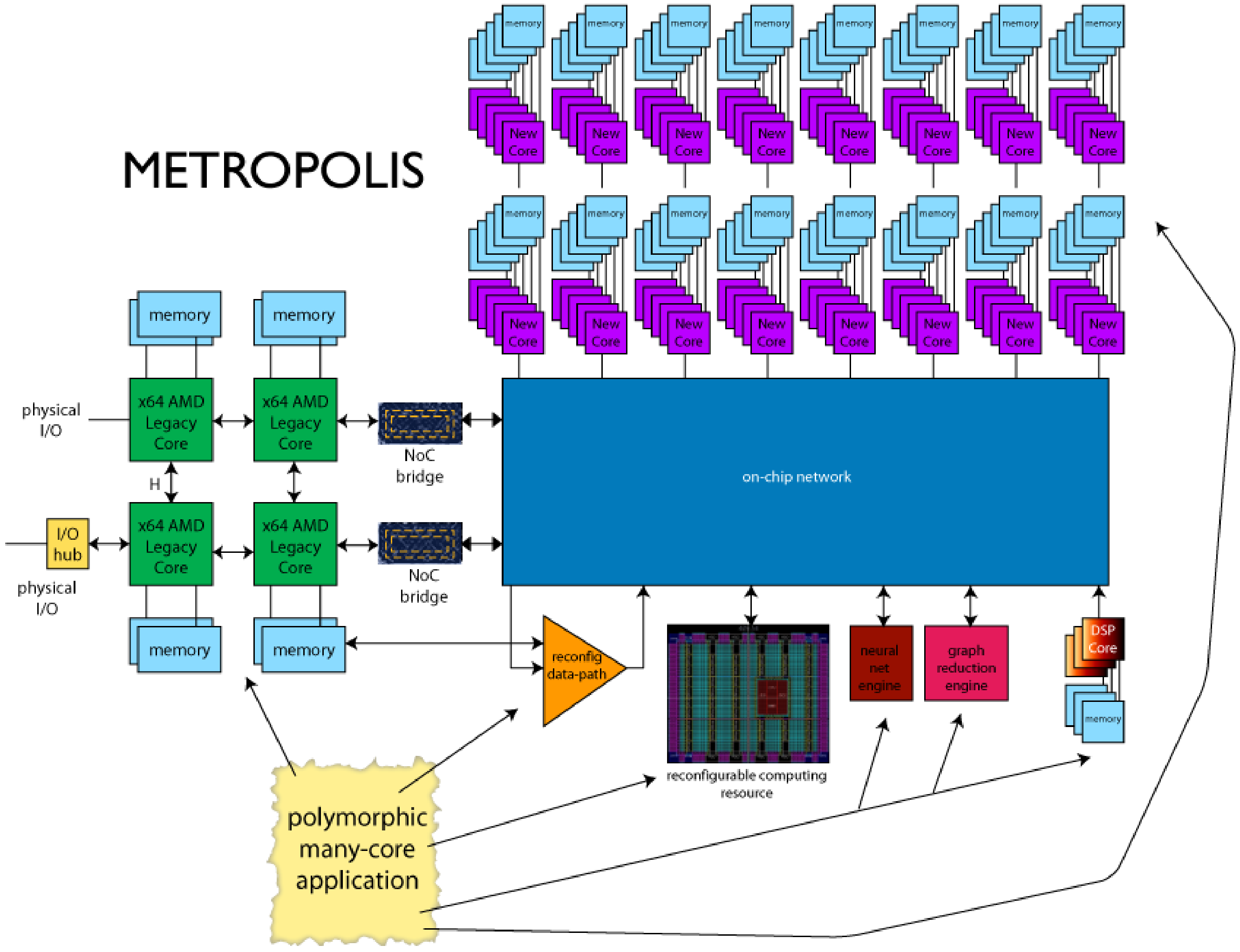






Satnam Singh
Microsoft Corporation

METROPOLIS

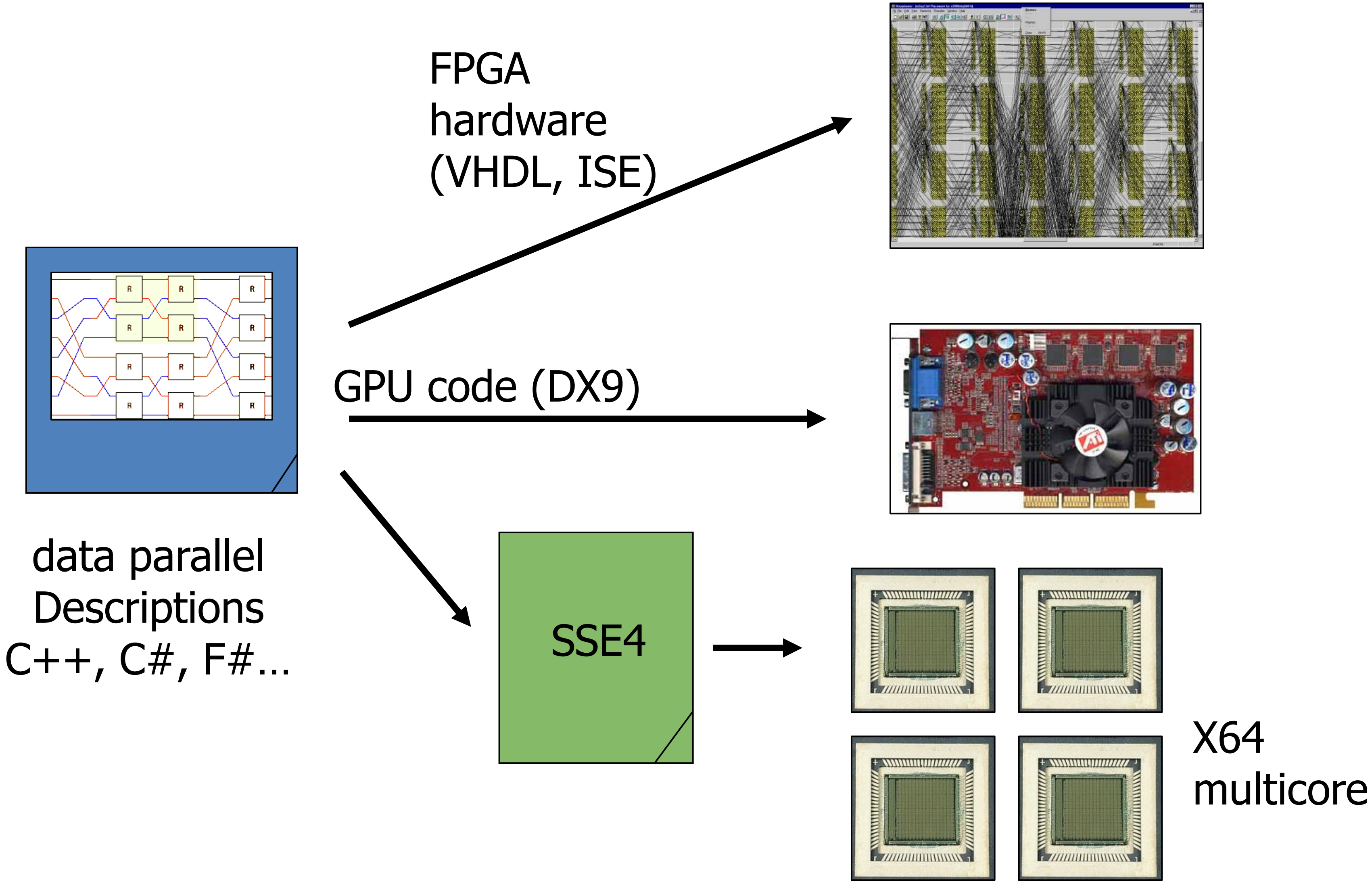

memory
memory
physical I/O
x64 AMD Legacy Core
x64 AMD Legacy Core
NoC bridge
H
I/O hub
x64 AMD Legacy Core
x64 AMD Legacy Core
NoC bridge
physical I/O
memory
memory
New Core
on-chip network
reconfig data-path
reconfigurable computing resource
neural net engine
graph reduction engine
DSP Core
polymorphic many-core application

Alchemy Project

Kiwi: concurrent C# programs for **control-oriented** applications [Univ. Cambridge]

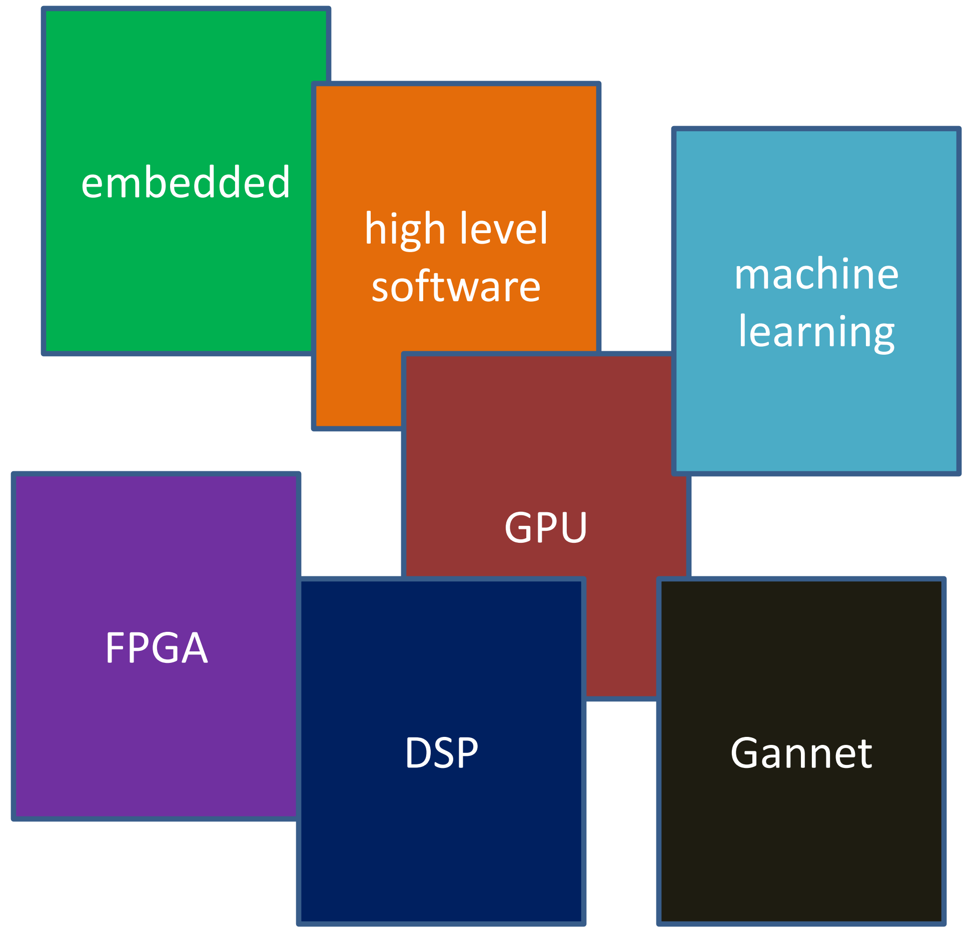
shape analysis: synthesis of **dynamic data structures** (C) [MPI and CMU]

Accelerator/FPGA: synthesis of **data parallel** programs in C++ [MSR Redmond]



FPGA
hardware
(VHDL, ISE)
GPU code (DX9)
data parallel
Descriptions
C++, C#, F#…
SSE4
X64
multicore

grand unification theory



polygots

Effort vs. Reward

CUDA
OpenCL
HLSL
DirectCompute

Accelerator

low effort	medium effort	high effort
low reward	medium reward	high reward

```
using System;
using Microsoft.ParallelArrays;

namespace AddArraysPointwise
{
    class AddArraysPointwiseDX9
    {
        static void Main(string[] args)
        {
            var x = new FloatParallelArray (new[] {1.0F, 2, 3, 4, 5});
            var y = new FloatParallelArray (new[] {6.0F, 7, 8, 9, 10});
            var dx9Target = new DX9Target();
            var z = x + y;
            foreach (var i in dx9Target.ToArray1D (z))
                Console.Write(i + " ");
            Console.WriteLine();
        }
    }
}
```

```
using System;
using Microsoft.ParallelArrays;

namespace AddArraysPointwiseMulticore
{
    class AddArraysPointwiseMulticore
    {
        static void Main(string[] args)
        {
            var x = new FloatParallelArray (new[] {1.0F, 2, 3, 4, 5});
            var y = new FloatParallelArray (new[] {6.0F, 7, 8, 9, 10});
            var multicoreTarget = new X64MulticoreTarget();
            var z = x + y;
            foreach (var i in multicoreTarget.ToArray1D (z))
                Console.Write(i + " ");
            Console.WriteLine();
        }
    }
}
```

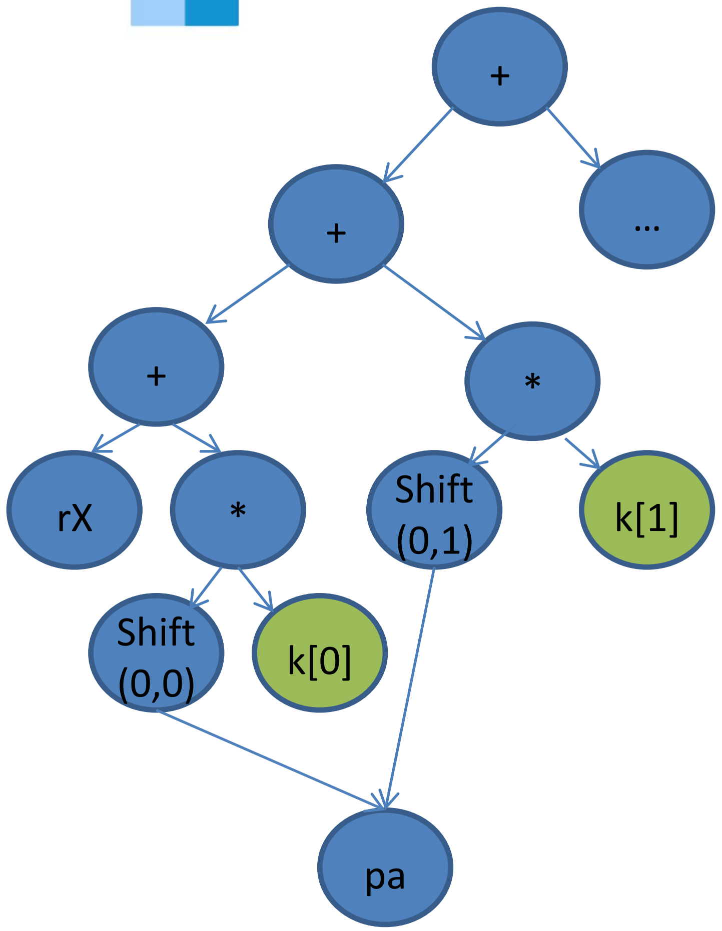
```
using System;
using Microsoft.ParallelArrays;

namespace AddArraysPointwiseFPGA
{
    class AddArraysPointwiseMulticore
    {
        static void Main(string[] args)
        {
            var x = new FloatParallelArray (new[] {1.0F, 2, 3, 4, 5});
            var y = new FloatParallelArray (new[] {6.0F, 7, 8, 9, 10});
            var fpgaTarget = new FPGATarget();
            var z = x + y;
            fpgaTarget.ToArray1D (z) ;
        }
    }
}
```

Microsoft Research

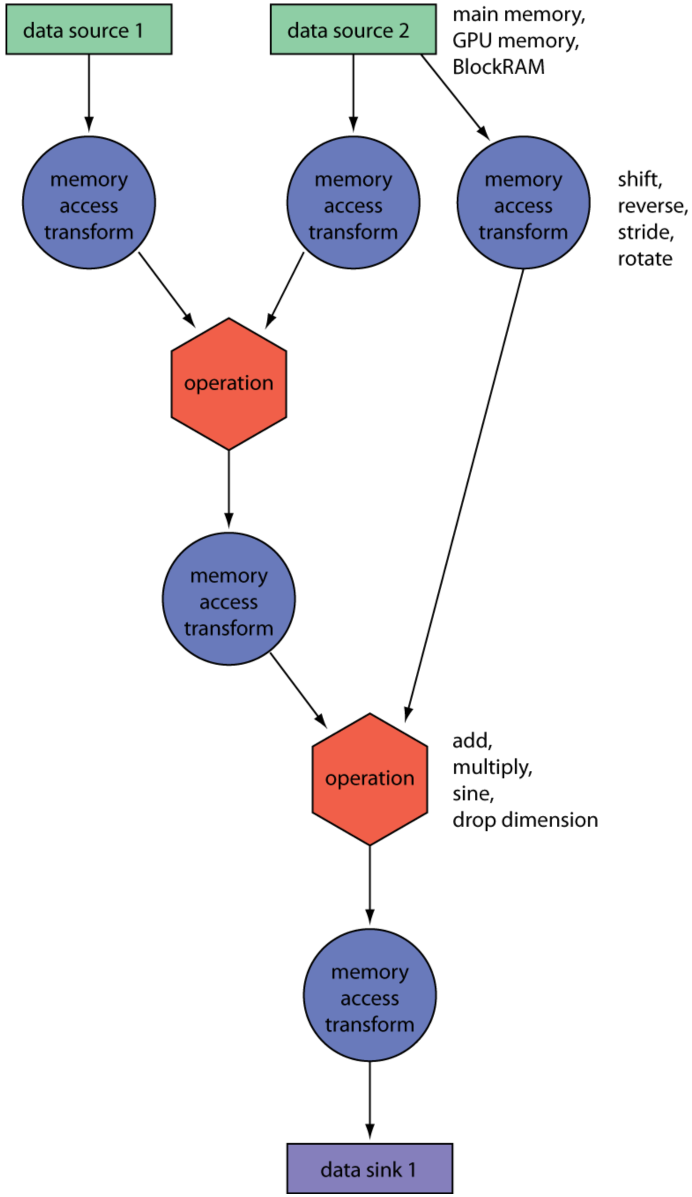
```
open System
open Microsoft.ParallelArrays
let main(args) =
    let x = new FloatParallelArray (Array.map float32 [|1; 2; 3; 4;  5 |])
    let y = new FloatParallelArray (Array.map float32 [|6; 7; 8; 9; 10 |])
    let z = x + y
    use dx9Target = new DX9Target()
    let zv = dx9Target.ToArray1D(z)
    printf "%A\n" zv
    0
```

```
open System
open Microsoft.ParallelArrays
[<EntryPoint>]
let main(args) =
    let x = new FloatParallelArray (Array.map float32 [|1; 2; 3; 4;  5 |])
    let y = new FloatParallelArray (Array.map float32 [|6; 7; 8; 9; 10 |])
    let z = x + y
    use multicoreTarget = new X64MulticoreTarget()
    let zv = multicoreTarget.ToArray1D(z)
    printf "%A\n" zv
    0
```

```
open System
open Microsoft.ParallelArrays
[<EntryPoint>]
let main(args) =
    let x = new FloatParallelArray (Array.map float32 [|1; 2; 3; 4;  5 |])
    let y = new FloatParallelArray (Array.map float32 [|6; 7; 8; 9; 10 |])
    let z = x + y
    use fpgaTarget = new FPGATarget("adder") ;
    let vhdl = fpgaTarget.ToArray1D(z)
    0
```




```
let rec convolve (shifts : int -> int [])
                 (kernel : float32 []) i
                 (a : FloatParallelArray)
  = let e = kernel.[i] * ParallelArrays.Shift(a, shifts i)
    if i = 0 then
      e
    else
      e + convolve shifts kernel (i-1) a
```


data source 1
data source 2
main memory,
GPU memory,
BlockRAM
memory access transform
memory access transform
memory access transform
shift,
reverse,
stride,
rotate
operation
memory access transform
operation
add,
multiply,
sine,
drop dimension
memory access transform
data sink 1

+, -, *, /, min, max, multiply-add, power

abs, ceiling, cos, fraction, floor, log2, negate, pow2, reciprocal, rsqrt, sin, sqrt

not, and, or

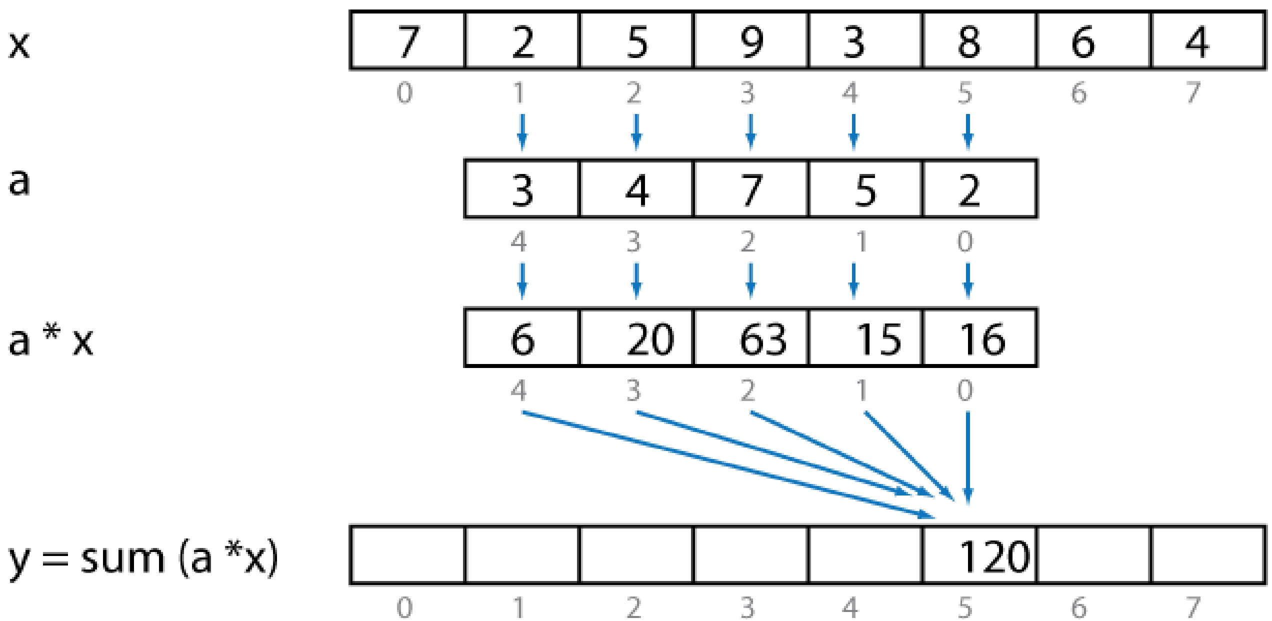
==, >=, <. <=, /=

sum, product, maxval, minval, any, all

add/drop dimension, expand, gather, replicate, rotate, section, shift, stretch, transpose

Inner product, outer product

$$y_t = \sum_{k=0}^{N-1} a_k x_{t-k}$$



```
public static int[] SequentialFIRFunction(int[] weights, int[] input)
        {
            int[] window = new int[size];
            int[] result = new int[input.Length];
            // Clear to window of x values to all zero.
            for (int w = 0; w < size; w++)
                window[w] = 0;
            // For each sample...
            for (int i = 0; i < input.Length; i++)
            {
                // Shift in the new x value
                for (int j = size - 1; j > 0; j--)
                    window[j] = window[j - 1];
                window[0] = input[i];
                // Compute the result value
                int sum = 0;
                for (int z = 0; z < size; z++)
                    sum += weights[z] * window[z];
                result[i] = sum;
            }
            return result;
        }
```

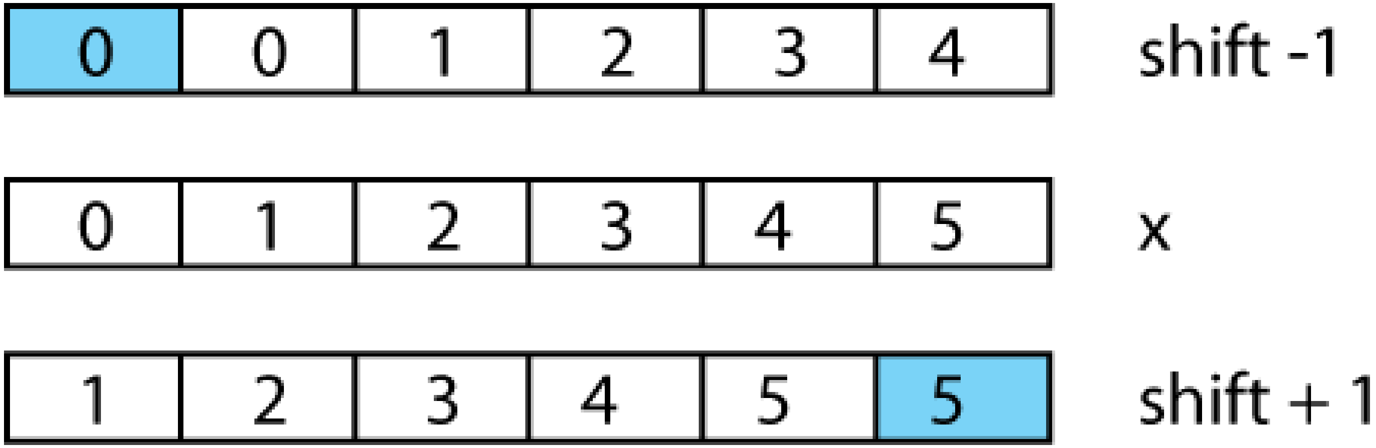
$$y_t = \sum_{k=0}^{N-1} a_k x_{t-k}$$

$y = [y[0], y[1], y[2], y[3], y[4], y[5], y[6], y[7]]$

$y[0] = a[0]x[0] + \mathbf{a[1]x[-1]} + a[2]x[-2] + a[3]x[-3] + a[4]x[-4]$
$y[1] = a[0]x[1] + \mathbf{a[1]x[0]} + a[2]x[-1] + a[3]x[-2] + a[4]x[-3]$
$y[2] = a[0]x[2] + \mathbf{a[1]x[1]} + a[2]x[0] + a[3]x[-1] + a[4]x[-2]$
$y[3] = a[0]x[3] + \mathbf{a[1]x[2]} + a[2]x[1] + a[3]x[0] + a[4]x[-1]$
$y[4] = a[0]x[4] + \mathbf{a[1]x[3]} + a[2]x[2] + a[3]x[1] + a[4]x[0]$
$y[5] = a[0]x[5] + \mathbf{a[1]x[4]} + a[2]x[3] + a[3]x[2] + a[4]x[1]$
$y[6] = a[0]x[6] + \mathbf{a[1]x[5]} + a[2]x[4] + a[3]x[3] + a[4]x[2]$
$y[7] = a[0]x[7] + \mathbf{a[1]x[6]} + a[2]x[5] + a[3]x[4] + a[4]x[3]$

$y = [y[0], y[1], y[2], y[3], y[4], y[5], y[6], y[7]]$
= a[0] * [x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]] +
a[1] * [x[-1], x[0], x[1], x[2], x[3], x[4], x[5], x[6]] +
a[2] * [x[-2], x[-1], x[0], x[1], x[2], x[3], x[4], x[5]] +
a[3] * [x[-3], x[-2], x[-1], x[0], x[1], x[2], x[3], x[4]] +
a[4] * [x[-4], x[-3], x[-2], x[-1], x[0], x[1], x[2], x[3]]

shift (x, 0) = [7, 2, 5, 9, 3, 8, 6, 4] = x
shift (x, -1) = [7, 7, 2, 5, 9, 3, 8, 6]
shift (x, -2) = [7, 7, 7, 2, 5, 9, 3, 8]



$$\begin{aligned} y &= [y[0], y[1], y[2], y[3], y[4], y[5], y[6], y[7]] \\ &= a[0] * [x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]] + \\ &\quad a[1] * [x[-1], x[0], x[1], x[2], x[3], x[4], x[5], x[6]] + \\ &\quad a[2] * [x[-2], x[-1], x[0], x[1], x[2], x[3], x[4], x[5]] + \\ &\quad a[3] * [x[-3], x[-2], x[-1], x[0], x[1], x[2], x[3], x[4]] + \\ &\quad a[4] * [x[-4], x[-3], x[-2], x[-1], x[0], x[1], x[2], x[3]] \end{aligned}$$

$$\begin{aligned} y = \quad & a[0] * \text{shift}(x, 0) + \\ & a[1] * \text{shift}(x, -1) + \\ & a[2] * \text{shift}(x, -2) + \\ & a[3] * \text{shift}(x, -3) + \\ & a[4] * \text{shift}(x, -4) \end{aligned}$$

shift (x, 0)	7	2	5	9	3	8	6	4
shift (x, -1)	7	7	2	5	9	3	8	6
shift (x, -2)	7	7	7	2	5	9	3	8
shift (x, -3)	7	7	7	7	2	5	9	3
shift (x, -4)	7	7	7	7	7	2	5	9

a[0] * shift (x, 0)	14	4	10	18	6	16	12	8
a[1] * shift (x, -1)	35	35	10	25	45	15	40	30
a[2] * shift (x, -2)	49	49	49	14	35	63	21	56
a[3] * shift (x, -3)	28	28	28	28	8	20	36	12
a[4] * shift (x, -4)	21	21	21	21	21	6	15	27
	+	+	+	+	+	+	+	+
y =	147	137	118	106	115	120	124	133

```
using Microsoft.ParallelArrays;
using A = Microsoft.ParallelArrays.ParallelArrays;
namespace AcceleratorSamples
{
    public class Convolver
    {
```

```
  for (int i = 0; i < a.Length; i++)
    ypar += a[i] * A.Shift(xpar, -i);
```

```
            var n = x.Length;
            var ypar = new FloatParallelArray(0.0f, new [] { n });
            for (int i = 0; i < a.Length; i++)
                ypar += a[i] * A.Shift(xpar, -i);
            float[] result = computeTarget.ToArray1D(ypar);
            return result;
        }
    }
}
```

shift (x, 0, 0)

7	2	5	9	3	8	6	4
2	8	7	4	8	9	3	5

→ a[0] * shift (x, 0, 0)

14	4	10	18	6	16	12	8
4	16	14	8	16	18	6	10

shift (x, 0, -1)

7	7	2	5	9	3	8	6
2	2	8	7	4	8	9	3

→ a[1] * shift (x, 0, -1)

35	35	10	25	45	15	40	30
10	10	40	35	20	40	45	15

shift (x, 0, -2)

7	7	7	2	5	9	3	8
2	2	2	8	7	4	8	9

→ a[2] * shift (x, 0, -2)

49	49	49	14	35	63	21	56
14	14	14	56	49	28	56	63

shift (x, 0, -3)

7	7	7	7	2	5	9	3
2	2	2	2	8	7	4	8

→ a[3] * shift (x, 0, -3)

28	28	28	28	8	20	36	12
8	8	8	8	32	28	16	32

shift (x, 0, -4)

7	7	7	7	7	2	5	9
2	2	2	2	2	8	7	4

→ a[4] * shift (x, 0, -4)

21	21	21	21	21	6	15	27
6	6	6	6	6	24	21	12

↓ + ↓ + ↓ + ↓ + ↓ + ↓ + ↓ + ↓ +

y[0] =	147	137	118	106	115	120	124	133
y[1] =	42	54	82	113	123	138	144	132

```
using Microsoft.ParallelArrays;
using A = Microsoft.ParallelArrays.ParallelArrays;
namespace AcceleratorSamples
{
    public class Convolver
    {
        public static float[,] Convolver1D_2DInput
          (Target computeTarget, float[] a, float[,] x)
```

```
var shiftBy = new [] {0, 0} ;
for (var i = 0; i < a.Length; i++)
 {
   shiftBy[1] = -i;
   ypar += a[i] * A.Shift(xpar, shiftBy);
}
```


```
                ypar += a[i] * A.Shift(xpar, shiftBy);
```


```
            }
            var result = computeTarget.ToArray2D(ypar);
            return result;
        }
    }
}
```

Microsoft Research

shift (x, 0, 0)

7	2	5	9	3	8	6	4
2	8	7	4	8	9	3	5

→ a[0] * shift (x, 0, 0)
2

14	4	10	18	6	16	12	8
4	16	14	8	16	18	6	10

shift (x, -1, 0)

7	2	5	9	3	8	6	4
7	2	5	9	3	8	6	4

→ a[1] * shift (x, -1, 0)
5

35	10	25	45	15	40	30	20
35	10	25	45	15	40	30	20

shift (x, -2, 0)

7	2	5	9	3	8	6	4
7	2	5	9	3	8	6	4

→ a[2] * shift (x, -2, 0)
7

49	14	35	63	21	56	42	28
49	14	35	63	21	56	42	28

shift (x, -3, 0)

7	2	5	9	3	8	6	4
7	2	5	9	3	8	6	4

→ a[3] * shift (x, -3, 0)
4

28	8	20	36	12	32	24	16
28	8	20	36	12	32	24	16

shift (x, -4, 0)

7	2	5	9	3	8	6	4
7	2	5	9	3	8	6	4

→ a[4] * shift (x, -4, 0)
3

21	6	15	27	9	24	18	12
21	6	15	27	9	24	18	12

↓ + ↓ + ↓ + ↓ + ↓ + ↓ + ↓ + ↓ +

y[0] =	147	42	105	189	63	168	126	84
y[1] =	137	54	109	179	73	170	120	86

Microsoft Research

```
using System;
using Microsoft.ParallelArrays;
namespace AcceleratorSamples
{
    public class Convolver2D
    {
        static FloatParallelArray convolve(Func<int, int[]> shifts, float[] kernel,
                                           int i, FloatParallelArray a)
        {
```

```
static FloatParallelArray convolveXY(float[] kernel,
                                     FloatParallelArray input)
{
  FloatParallelArray convolveX
    =    convolve(i => new [] { -i, 0 }, kernel,
                                          kernel.Length - 1, input);
  return convolve(i => new [] { 0, -i }, kernel,
                                          kernel.Length - 1, convolveX);
}
```

```
            return e + convolve(shifts, kernel, i - 1, a);
}
            var inputArray = new FloatParallelArray(inputData);
            var result = dx9Target.ToArray2D(convolveXY (testKernel, inputArray));
            for (var row = 0; row < inputSize; row++)
            {
                for (var col = 0; col < inputSize; col++)
                    Console.Write("{0} ", result[row, col]);
                Console.WriteLine();
            }
        }
    }
}
```

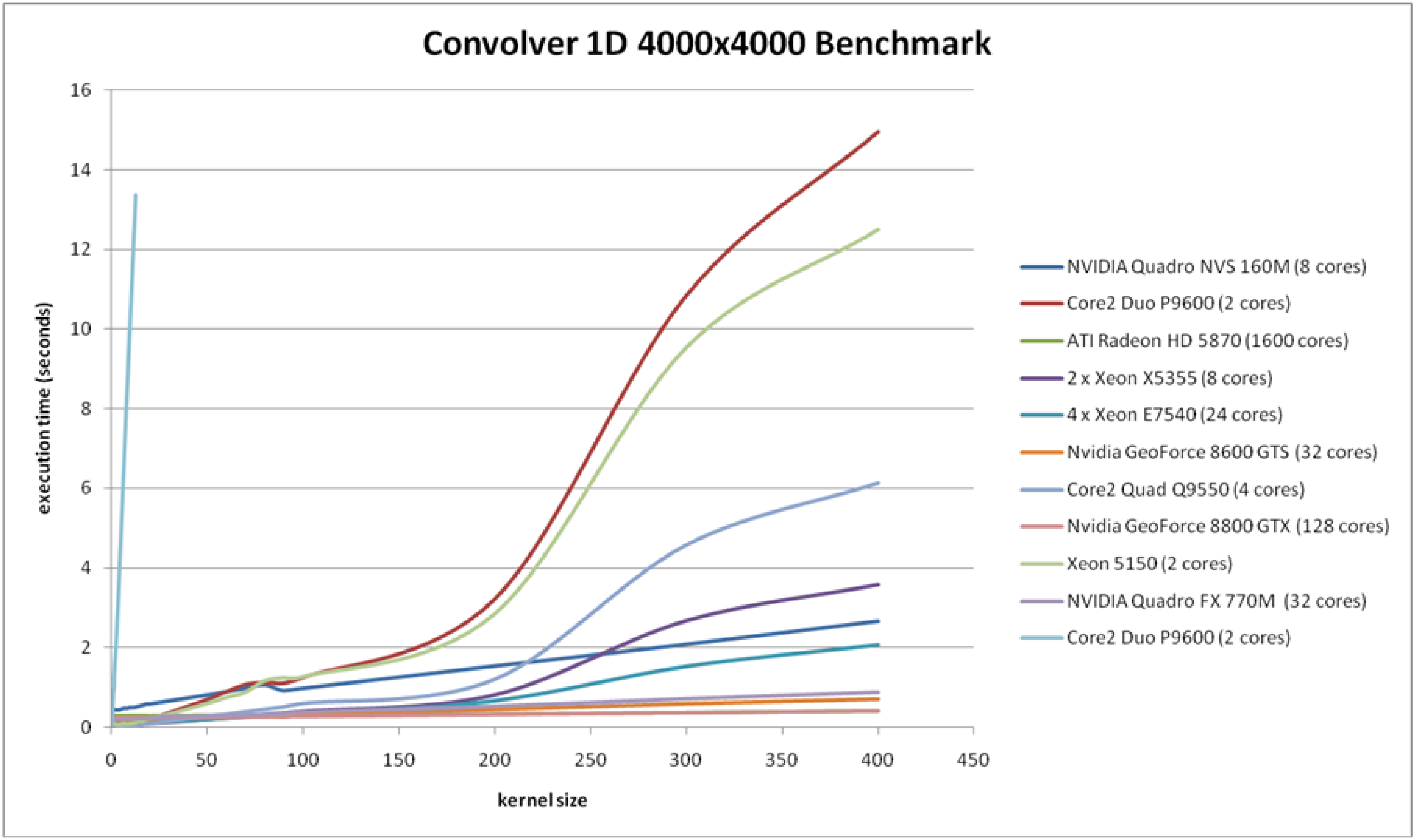
```
using System;
using System.Linq;
using Microsoft.ParallelArrays;
namespace AcceleratorSamples
{
 static FloatParallelArray convolve(this FloatParallelArray a,
                                    Func<int, int[]> shifts,
                                    float[] kernel)
 { return kernel
          .Select((k, i) => k * ParallelArrays.Shift(a, shifts(i)))
          .Aggregate((a1, a2) => a1 + a2);
 }

 static FloatParallelArray convolveXY(this FloatParallelArray input,
                                      float[] kernel)
 { return input
           .convolve(i => new[] { -i, 0 }, kernel)
           .convolve(i => new[] { 0, -i }, kernel);
 }
                for (int col = 0; col < inputSize; col++)
                    Console.Write("{0} ", result[row, col]);
                Console.WriteLine();
            }
        }
    }
}
```

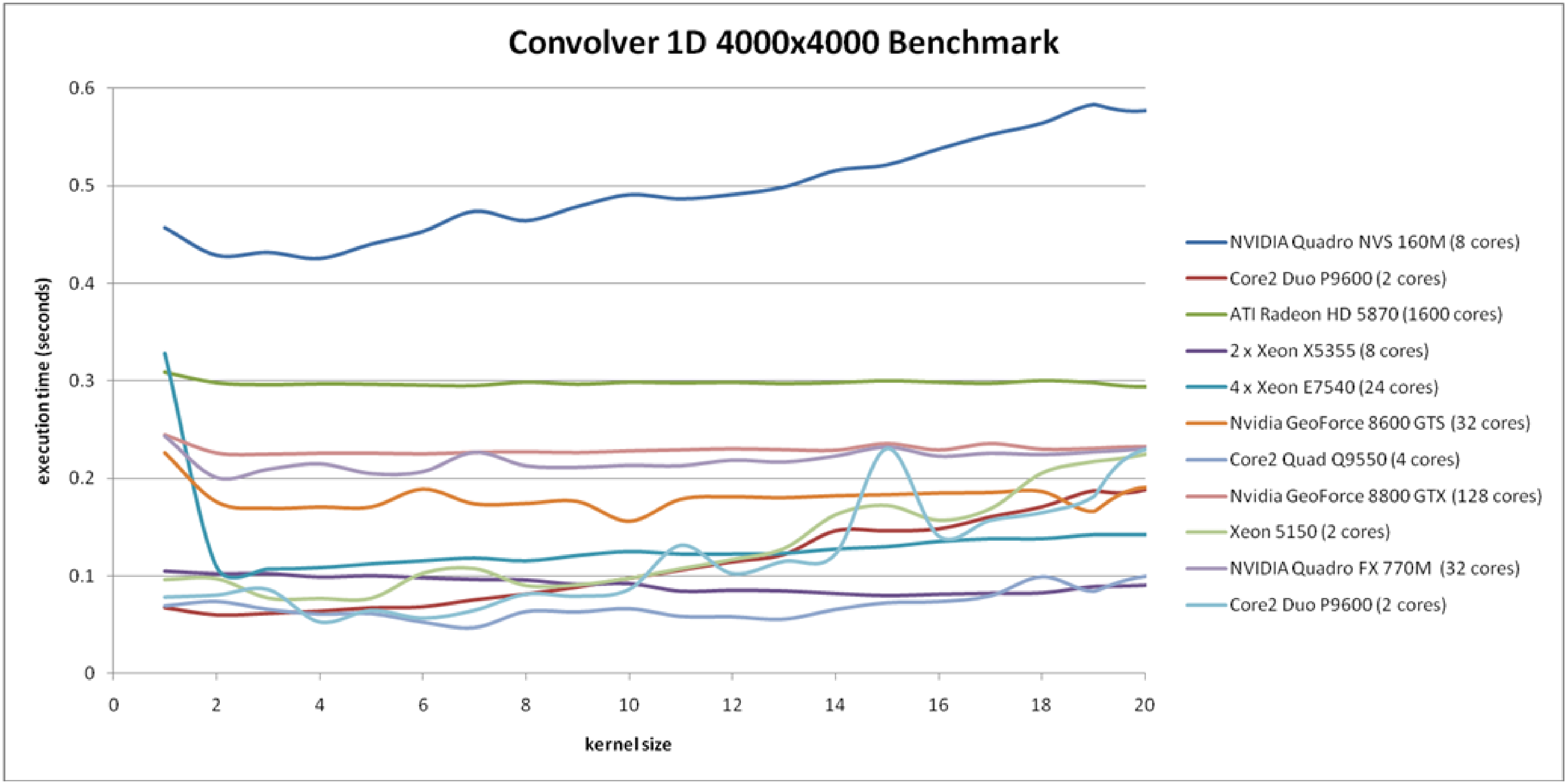
```
FPA ConvolveXY(Target &tgt, int height, int width, int filterSize, float filter[], FPA input, float *resultArray)
{
    // Convolve in X (row) direction.
    size_t dims[] = {height,width};
    FPA smoothX = FPA(0,dims, 2);
    intptr_t counts[] = {0,0};
    int filterHalf = filterSize/2;
    float scale;
    for (int i = -filterHalf; i <= filterHalf; i++)
    {
        counts[0] = i;
        scale = filter[i + filterHalf];
        smoothX += Shift(input, counts, 2) * scale;
    }

    // Convolve in Y (col) direction.
    counts[0] = 0;
    FPA result = FPA(0,dims, 2);
    for (int i = -filterHalf; i <= filterHalf; i++)
    {
        counts[1] = i;
        scale = filter[filterHalf + i];
        result += Shift(smoothX, counts, 2) * scale;
    }
    tgt.ToArray(result, resultArray, height, width, width * sizeof(float));
        return smoothX ;
};
```

```
open System
open Microsoft.ParallelArrays
[<EntryPoint>]
let main(args) =
    // Declare a filter kernel for the convolution
    let testKernel = Array.map float32 [| 2; 5; 7; 4; 3 |]
    // Specify the size of each dimension of the input array
    let inputSize = 10
    // Create a pseudo-random number generator
```

```
let convolveXY kernel input
   = // First convolve in the X direction and then in Y
     let convolveX = convolve (fun i -> [| -i; 0 |]) kernel
                                 (kernel.Length - 1) input
     let convolveY = convolve (fun i -> [| 0; -i |]) kernel
                              (kernel.Length - 1) convolveX
     convolveY
```

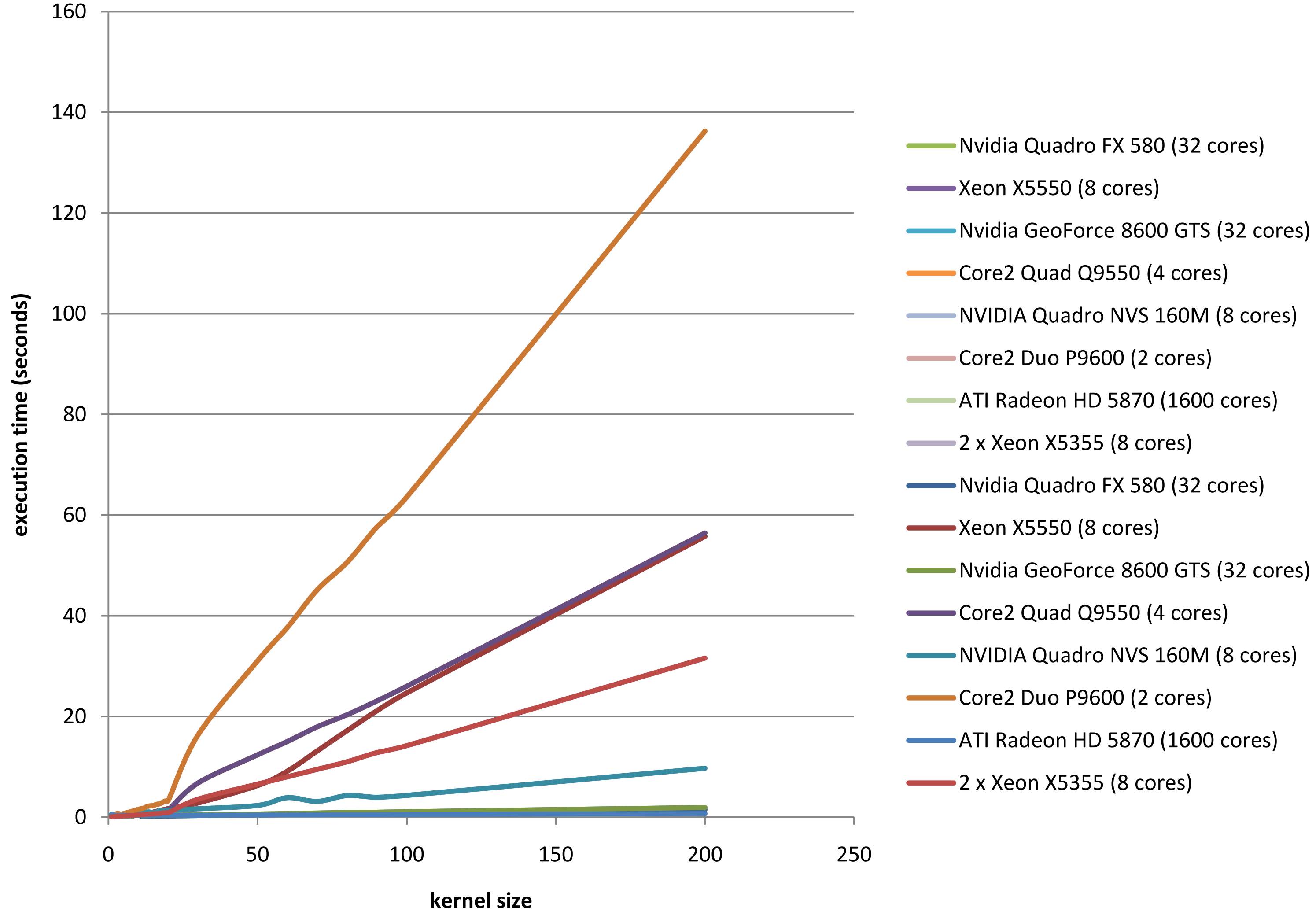
```
              e + convolve shifts kernel (i-1) a
    // Declare a 2D convolver
    let convolveXY kernel input
       = // First convolve in the X direction and then in the Y direction
         let convolveX = convolve (fun i -> [| -i; 0 |]) kernel (kernel.Length - 1) input
         let convolveY = convolve (fun i -> [| 0; -i |]) kernel (kernel.Length - 1) convolveX
         convolveY
    // Create a DX9 target and use it to convolve the test input
    use dx9Target = new DX9Target()
    let convolveDX9 = dx9Target.ToArray2D (convolveXY testKernel testArray)
    printfn "DX9: -> \r\n%A" convolveDX9
    0
```


Convolver 1D 4000x4000 Benchmark
execution time (seconds)
kernel size
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300
350
400
450
NVIDIA Quadro NVS 160M (8 cores)
Core2 Duo P9600 (2 cores)
ATI Radeon HD 5870 (1600 cores)
2 x Xeon X5355 (8 cores)
4 x Xeon E7540 (24 cores)
Nvidia GeoForce 8600 GTS (32 cores)
Core2 Quad Q9550 (4 cores)
Nvidia GeoForce 8800 GTX (128 cores)
Xeon 5150 (2 cores)
NVIDIA Quadro FX 770M (32 cores)
Core2 Duo P9600 (2 cores)

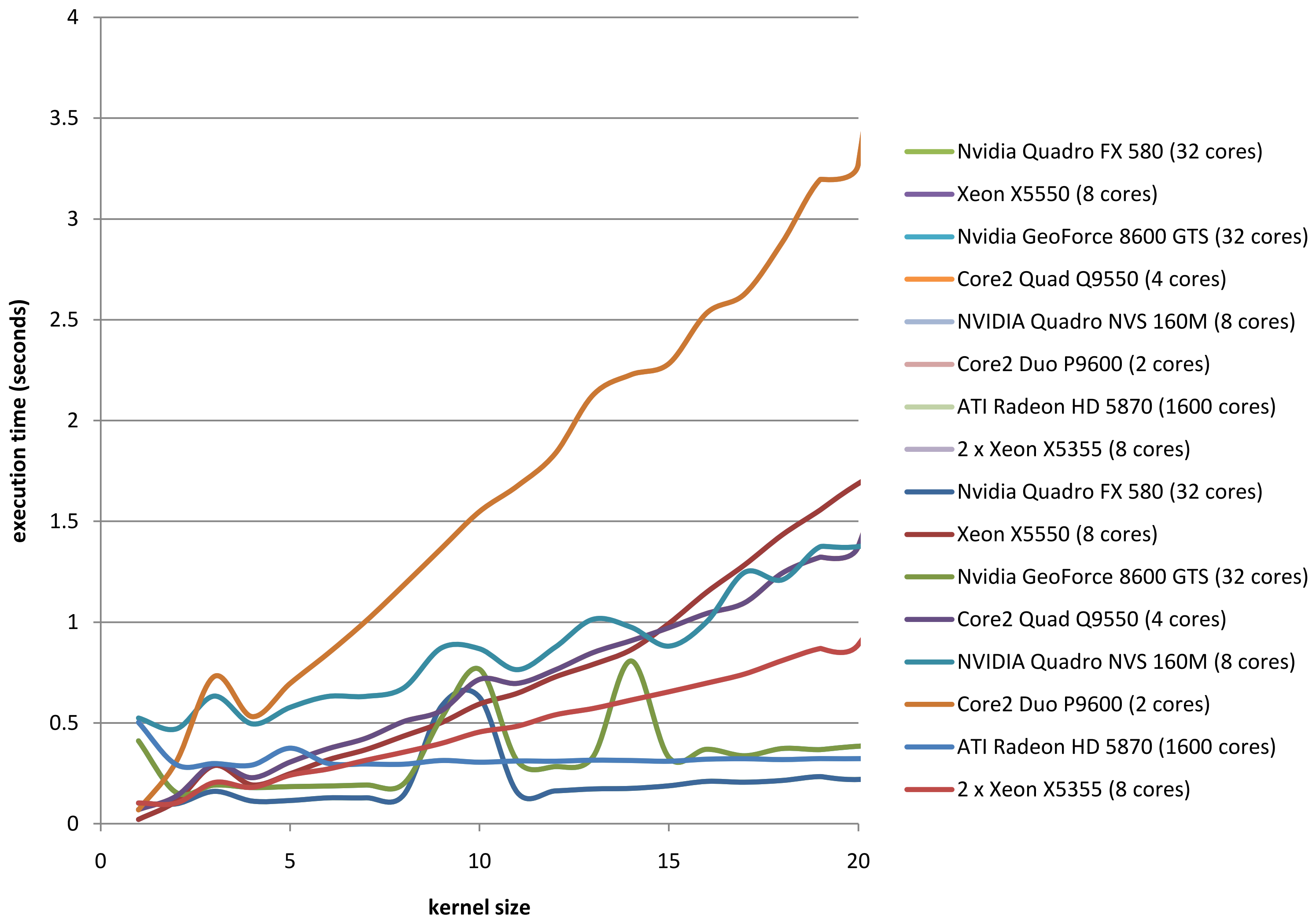


Convolver 1D 4000x4000 Benchmark
execution time (seconds)
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
kernel size
NVIDIA Quadro NVS 160M (8 cores)
Core2 Duo P9600 (2 cores)
ATI Radeon HD 5870 (1600 cores)
2 x Xeon X5355 (8 cores)
4 x Xeon E7540 (24 cores)
Nvidia GeoForce 8600 GTS (32 cores)
Core2 Quad Q9550 (4 cores)
Nvidia GeoForce 8800 GTX (128 cores)
Xeon 5150 (2 cores)
NVIDIA Quadro FX 770M (32 cores)
Core2 Duo P9600 (2 cores)

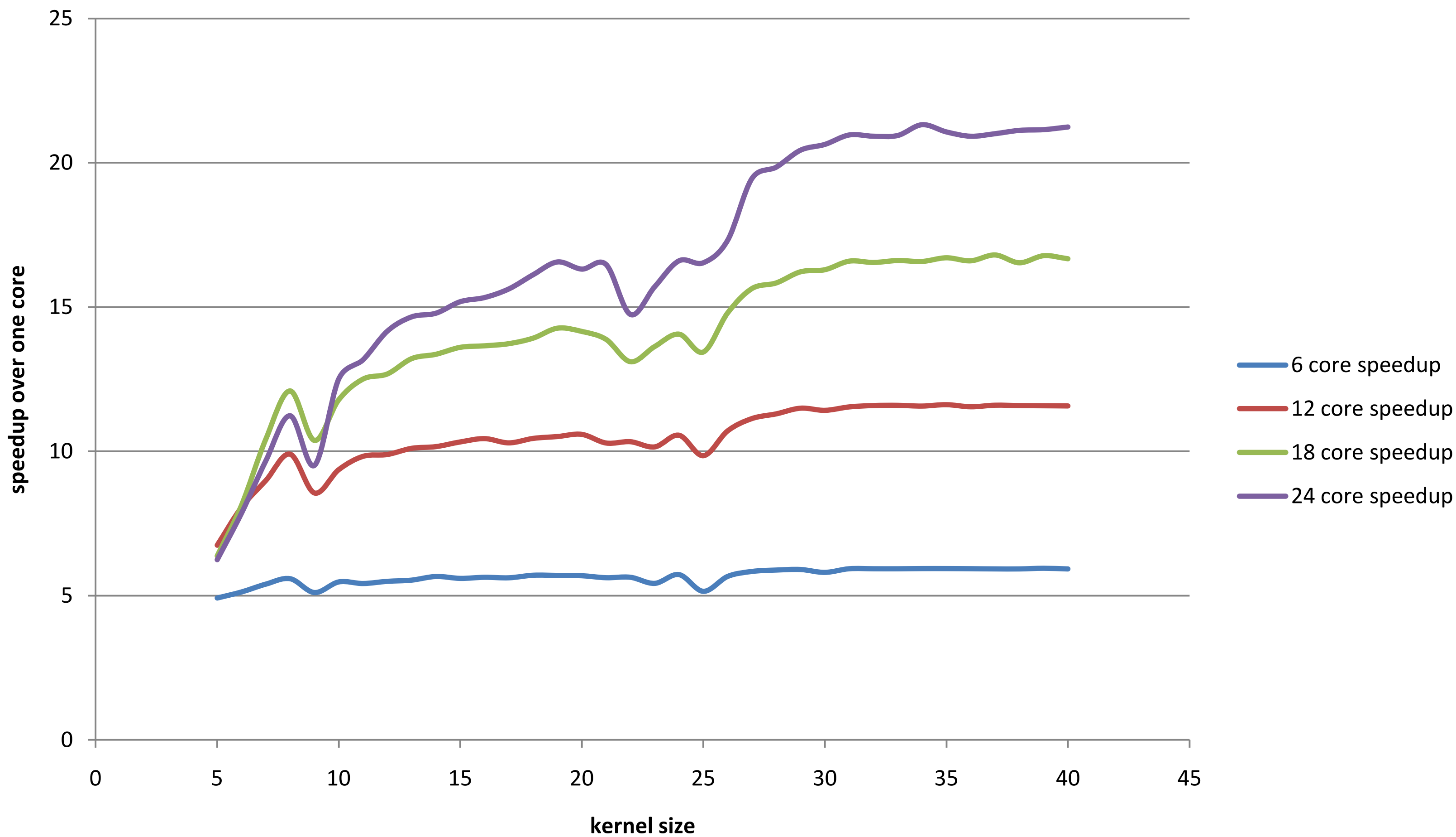
Convolver 2D 4000x4000 Benchmark



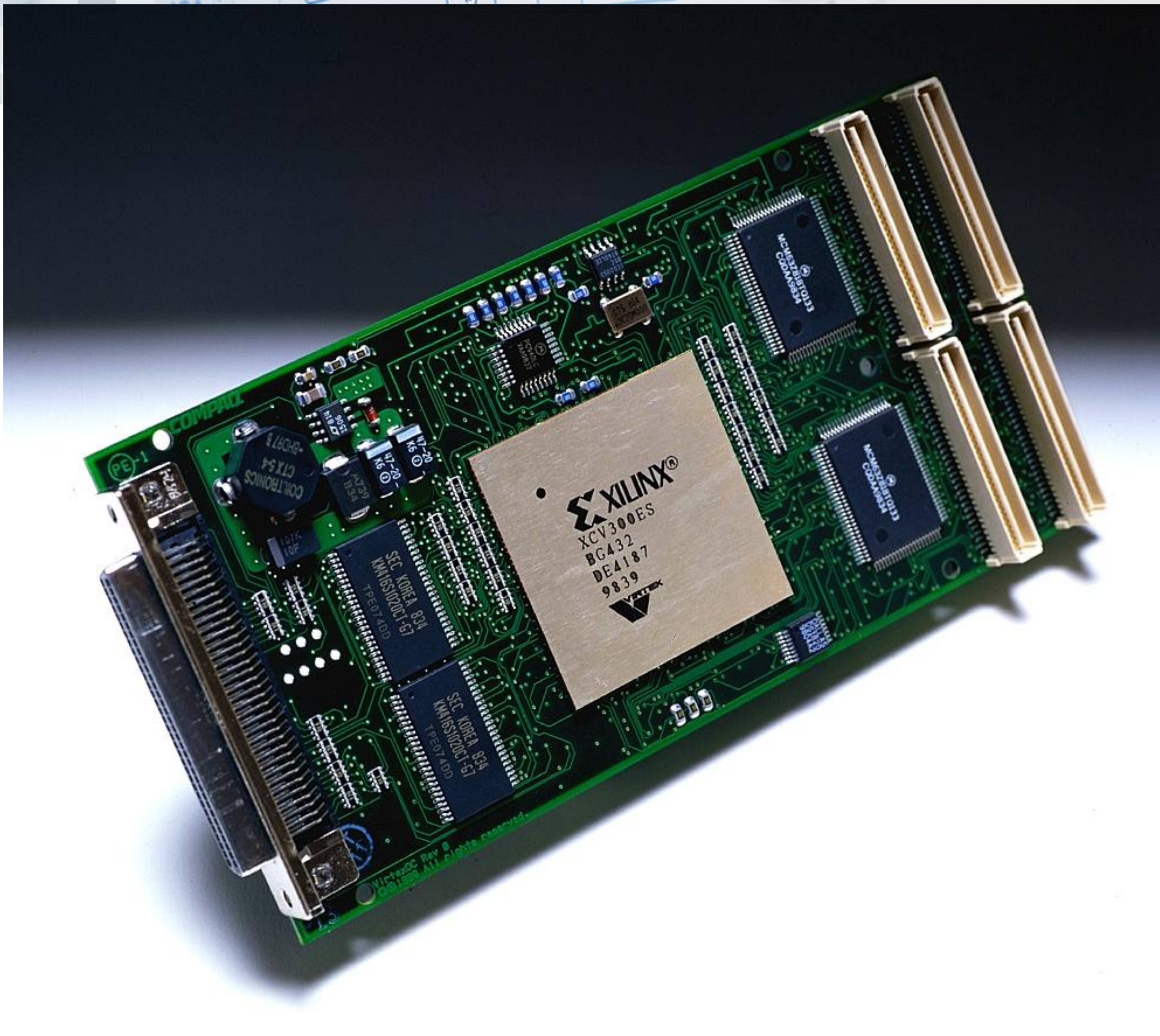
Convolver 2D 4000x4000 Benchmark



x64 multicore target benchmark for 2D convolver (24 core server Xeon E7540)



Width	Height	Iters		JIT	Setup	Execute
1000	1000	20		0.05	2.45	6.1
2000	2000	20		0.05	2.4	24.25
3000	3000	20		0.1	2.65	47.6


XILINX®
XCV300ES
BG432
DE4187
9839
COMPAQ
SEC KOREA 834
SEC KOREA 834


BEE3
FPGA_B
FPGA_C
15 3:43PM

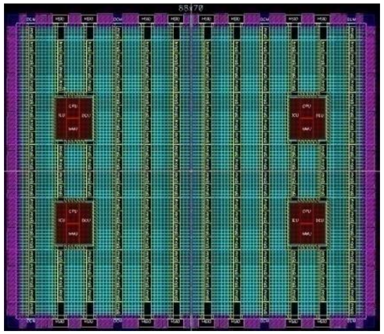
FPGAs as Co-Processors

XD2000i FPGA in-socket accelerator for Intel FSB

XD2000F FPGA in-socket accelerator for AMD socket F

XD1000 FPGA co-processor module for socket 940

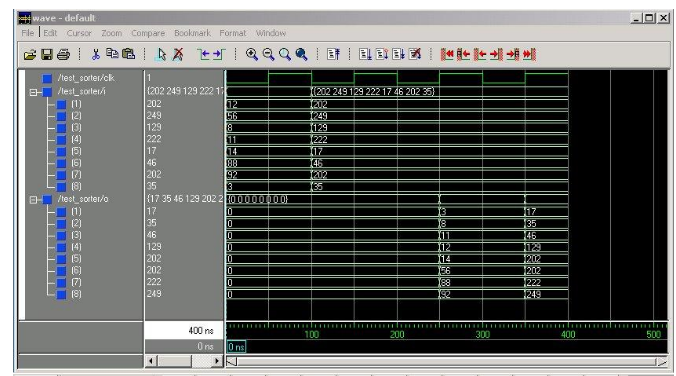
Microsoft Research

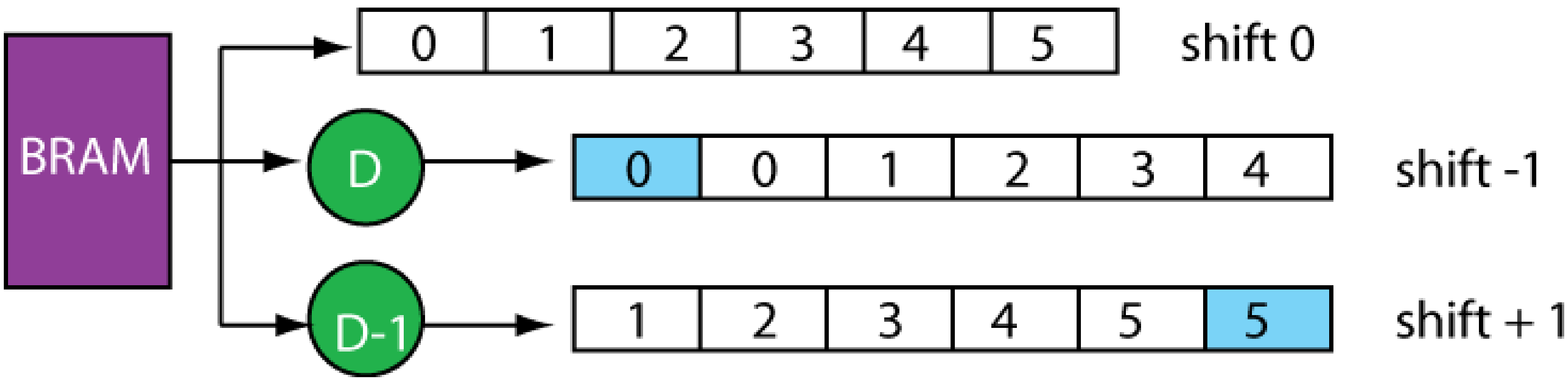


opportunity

scientific computing
data mining
search
image processing
financial analytics

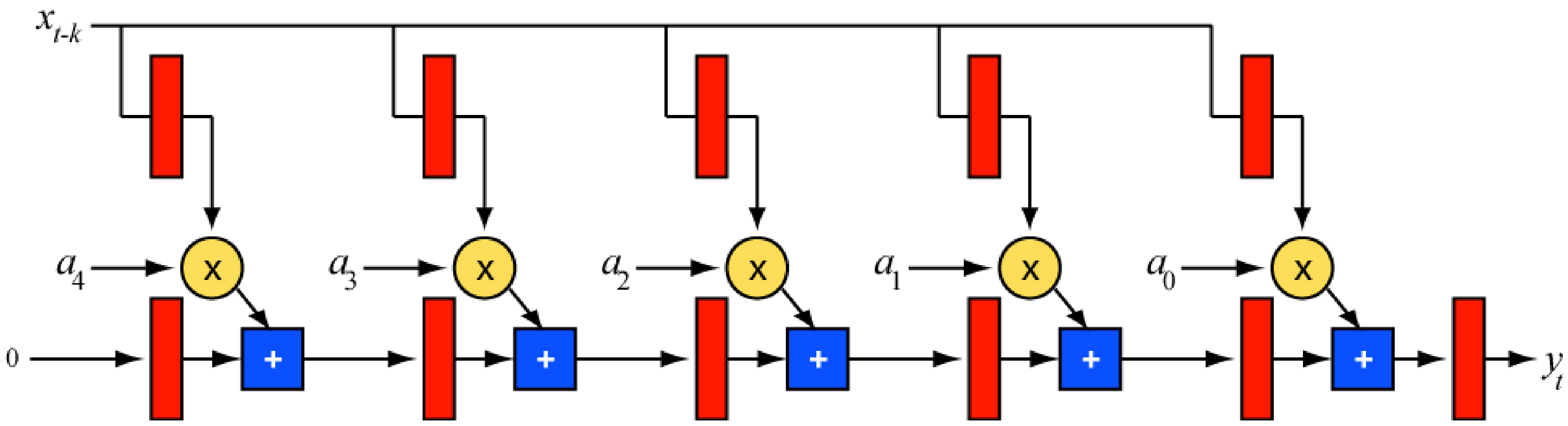
challenge





BRAM
D
D-1
0 1 2 3 4 5
shift 0
0 0 1 2 3 4
shift -1
1 2 3 4 5 5
shift + 1

Convolver



$$y_t = \sum_{k=0}^{N-1} a_k x_{t-k}$$

2D Convolver

32-bit integer input data
32-bit integer coefficients
3 taps

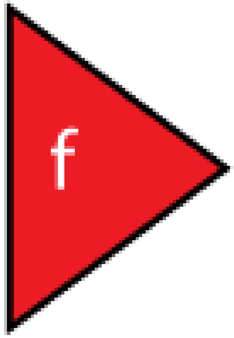
Virtex-5 FPGA

XC5VLX50T-2

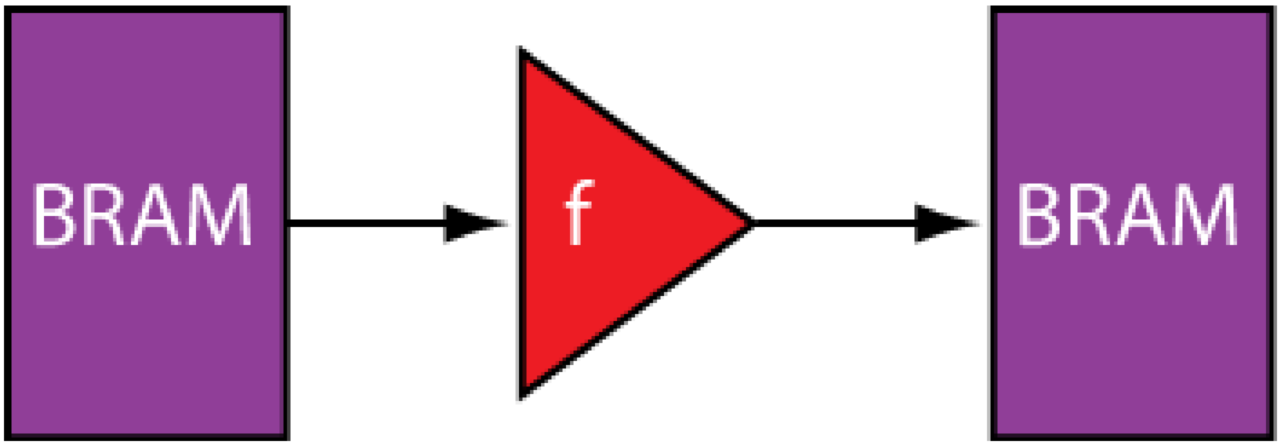
175 MHz

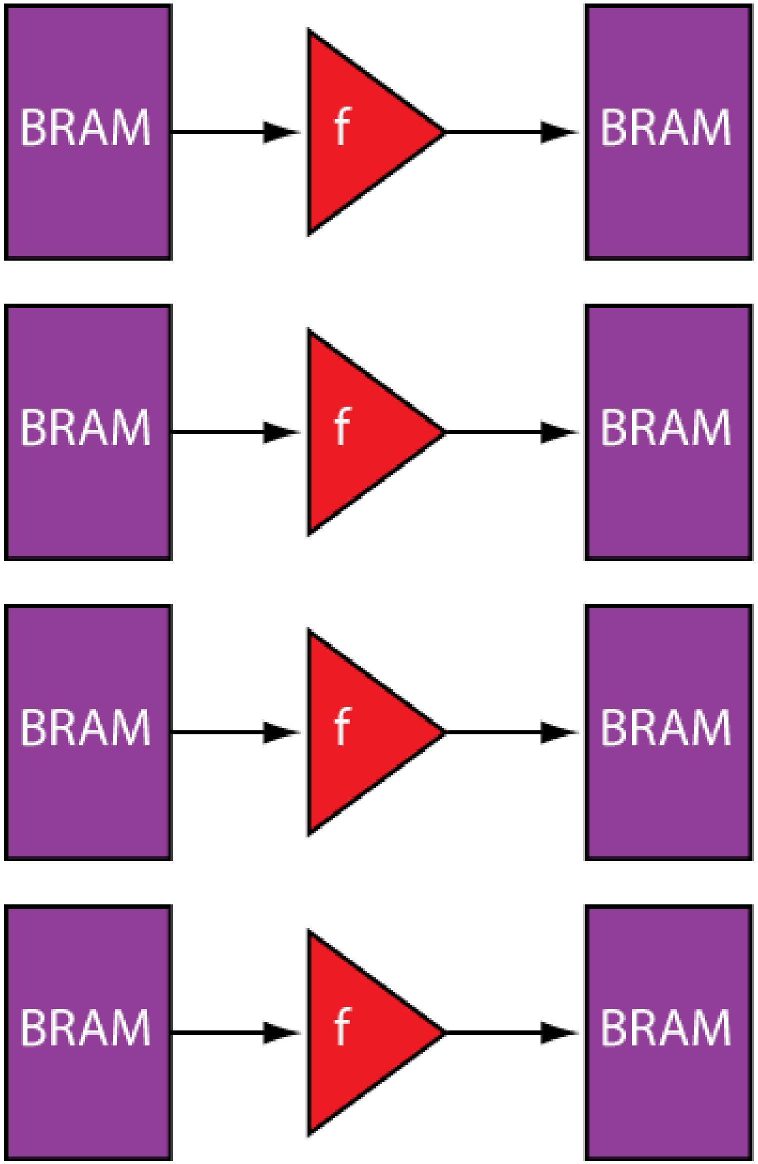
BRAM to BRAM

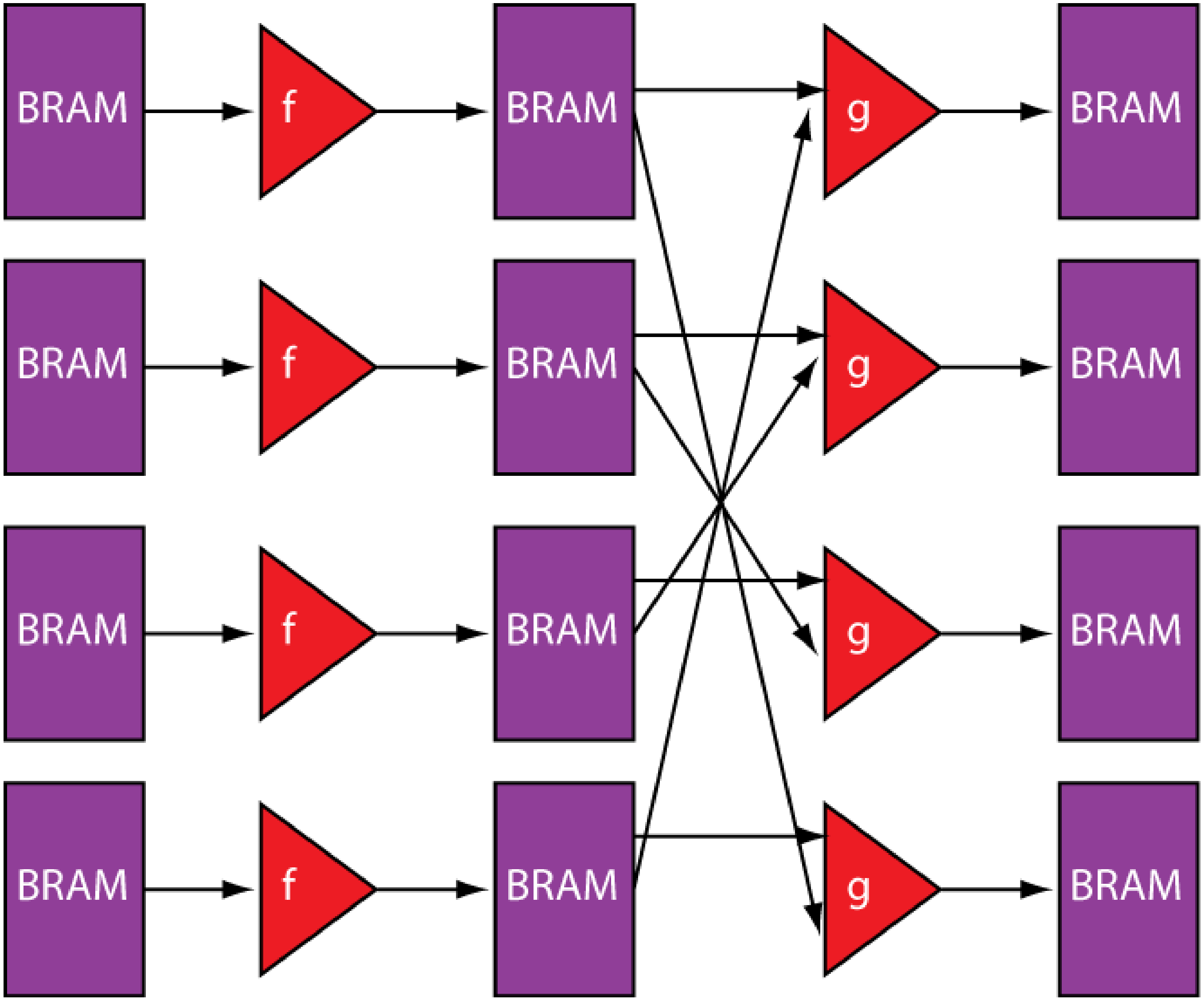
36Kbits
38,304
dual-ported



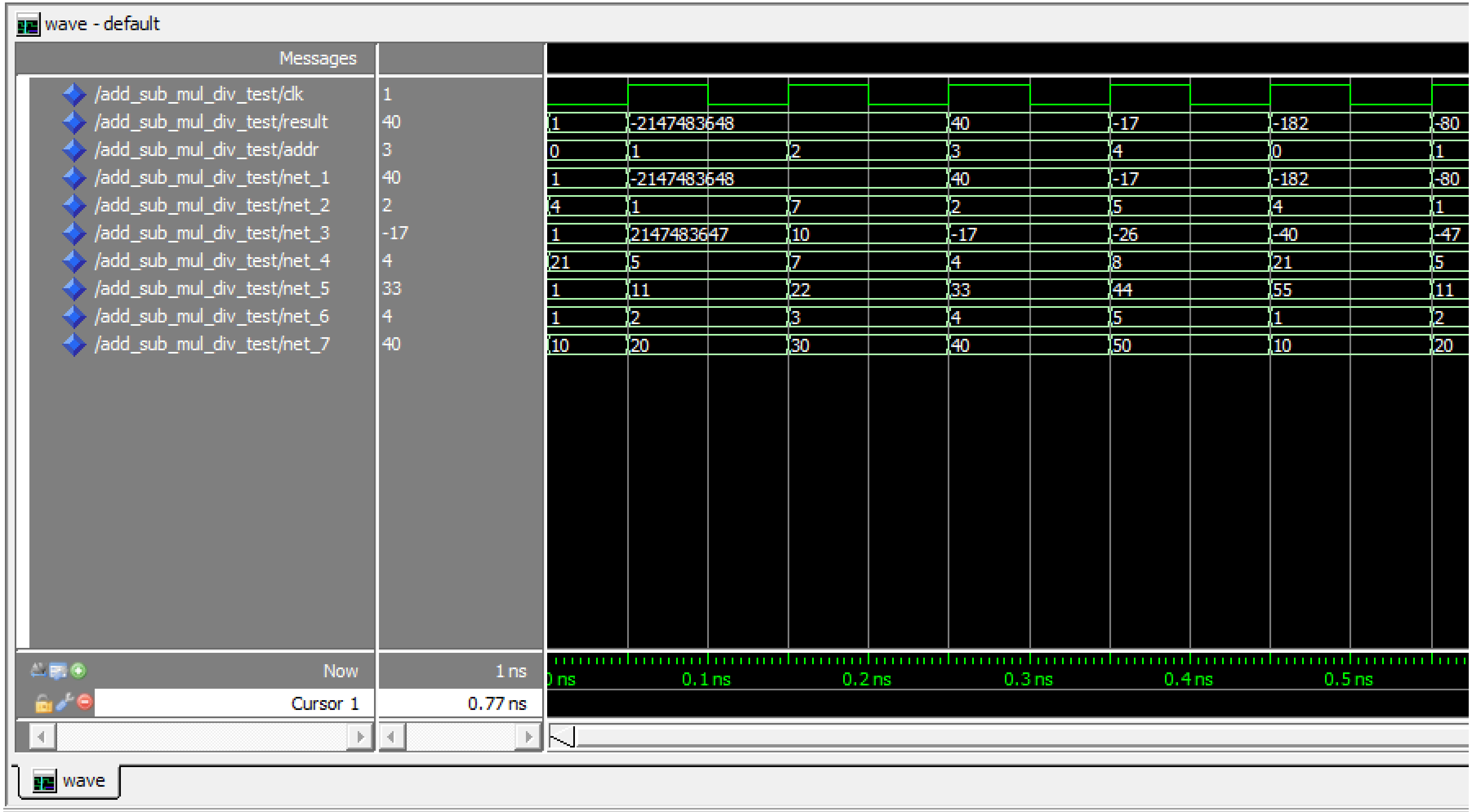
FPGA basic logic (LUTS)
DSP blocks (current max 2016)

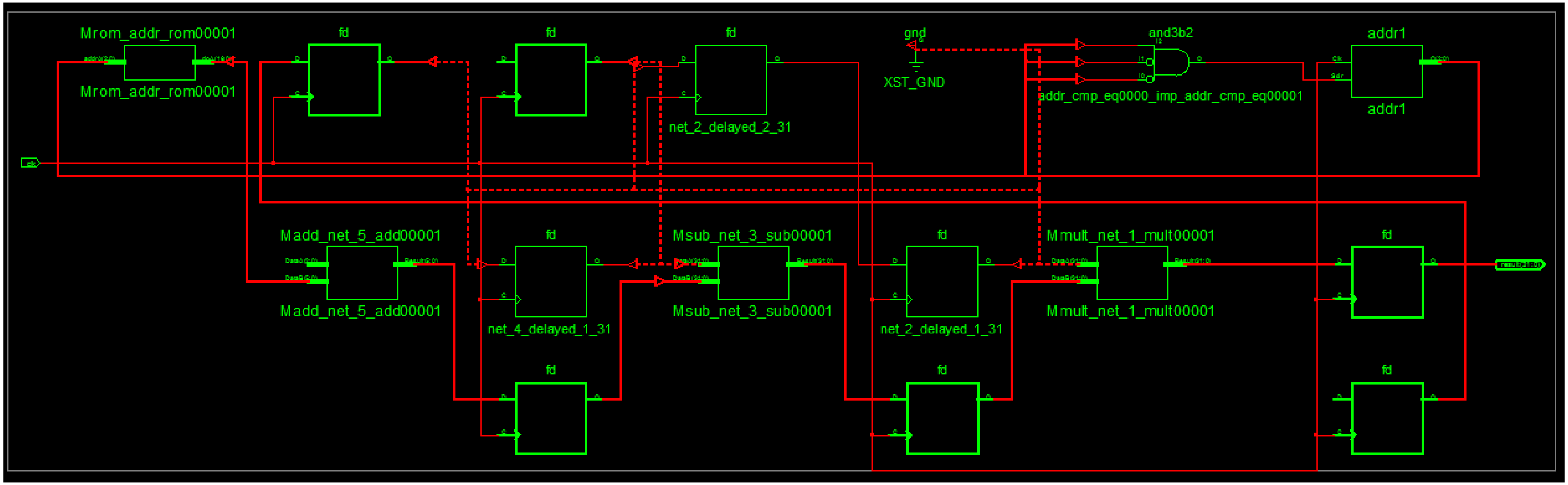

BRAM
f
BRAM


BRAM
f
BRAM
BRAM
f
BRAM
BRAM
f
BRAM
BRAM
f
BRAM

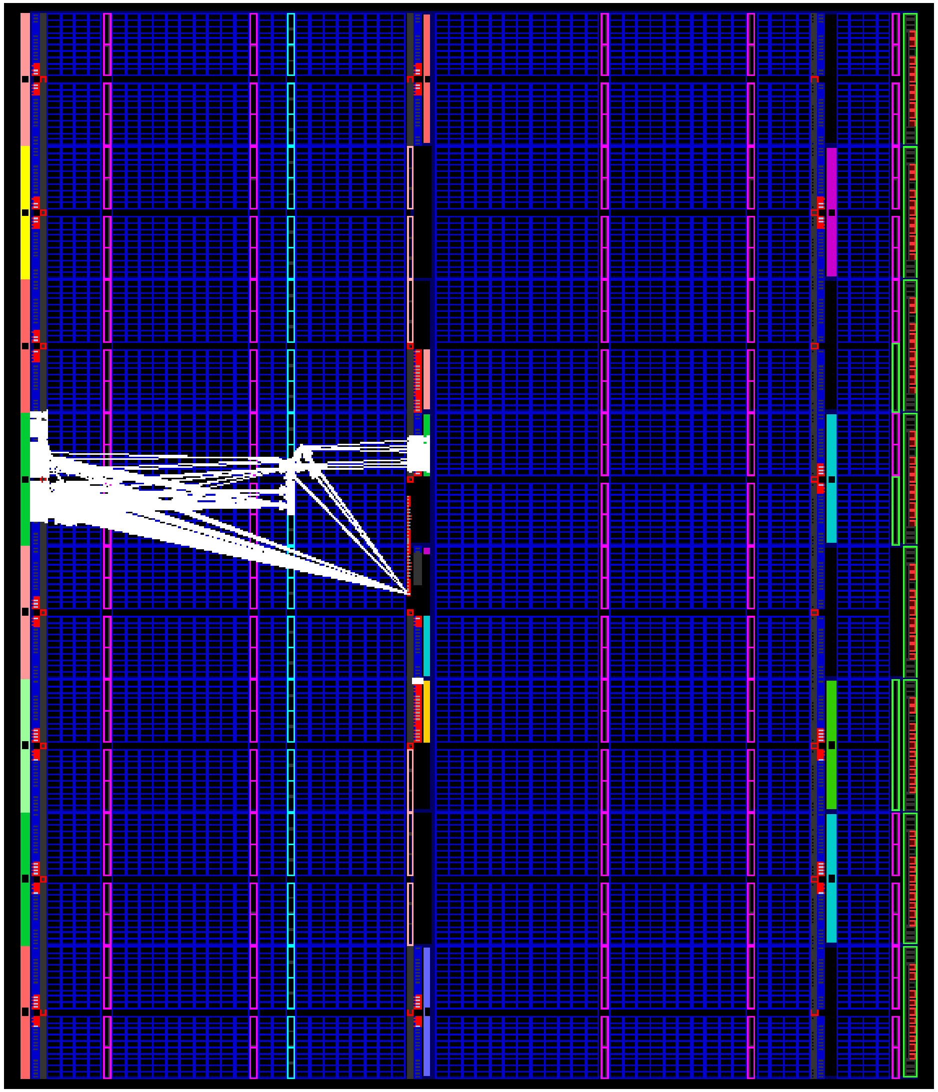

BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM

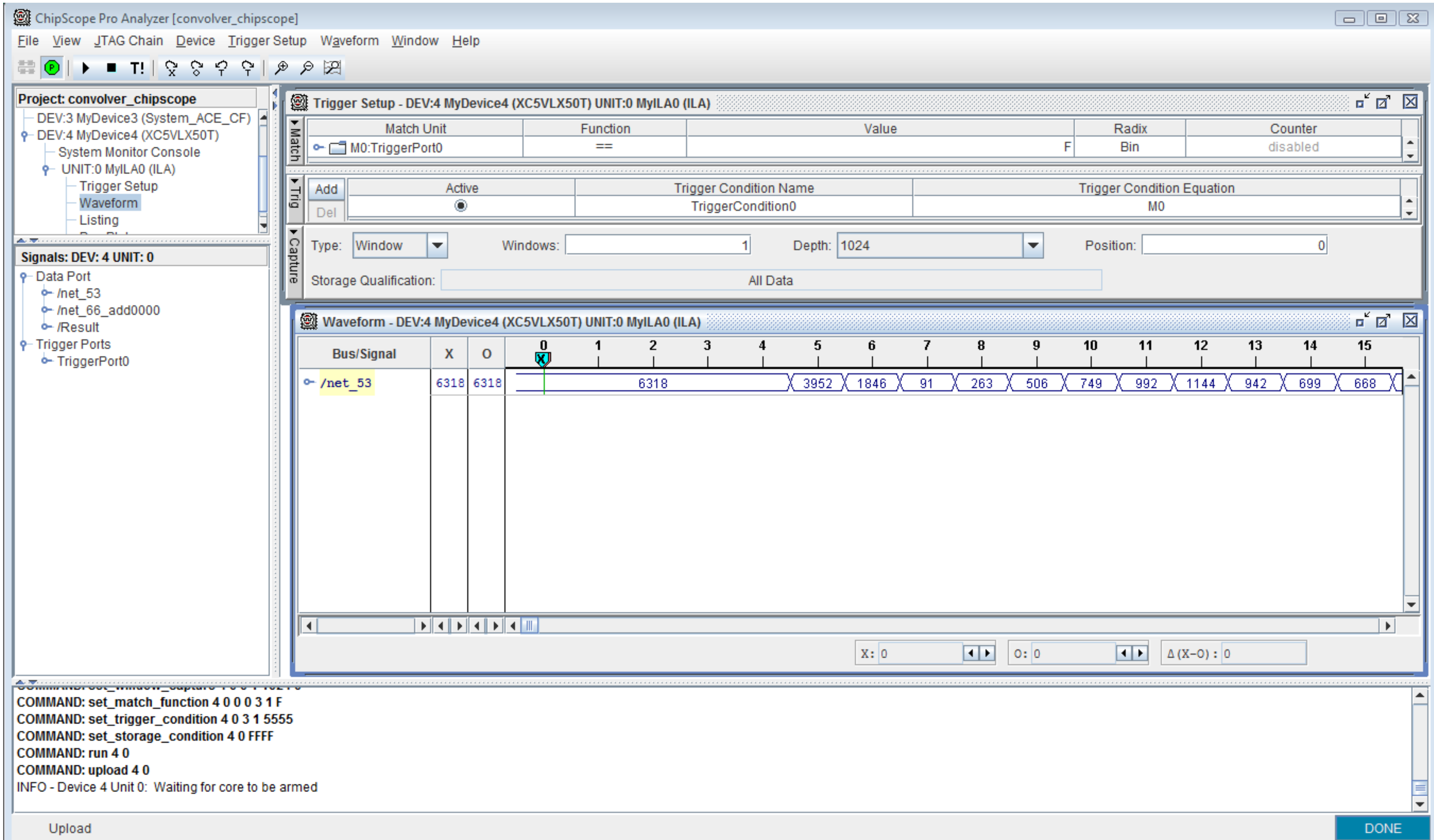
```
FPA ConvolveX(Target &tgt, int height, int width, int filterSize,
              float filter[], FPA input, float *resultArray)
{
    // Convolve in X direction.
    size_t dims[] = {height,width};
    FPA smoothX = FPA(0,dims, 2);
    intptr_t counts[] = {0,0};
    int filterHalf = filterSize/2;
    float scale;
    for (int i = -filterHalf; i <= filterHalf; i++)
    {
        counts[1] = i;
        scale = filter[i + filterHalf];
        smoothX += Shift(input, counts, 2) * scale;
    }
    tgt.ToArray(smoothX, resultArray, height, width,
                width * sizeof(float));
    return smoothX ;
};
```

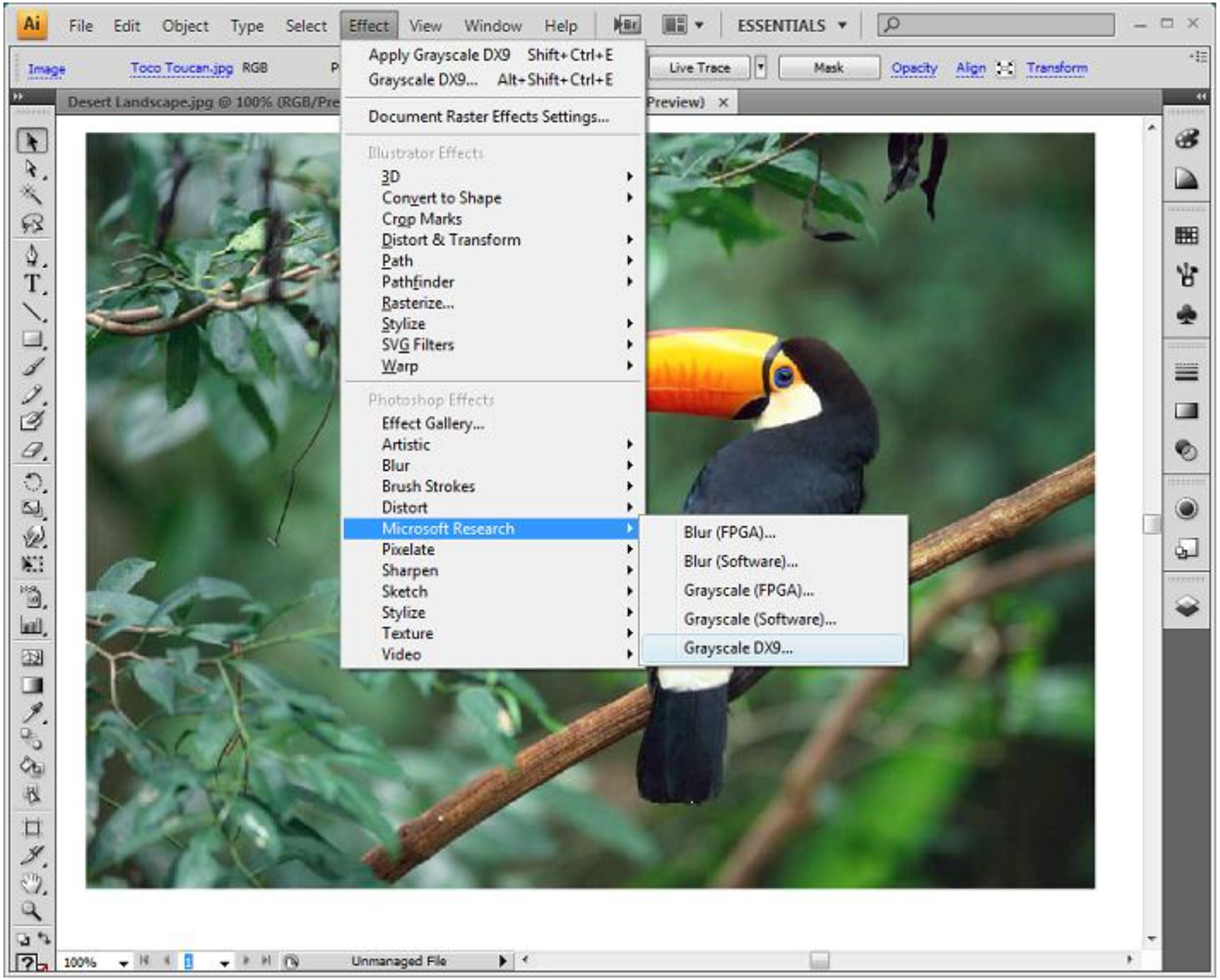

wave - default
Messages
/add_sub_mul_div_test/clk 1
/add_sub_mul_div_test/result 40
/add_sub_mul_div_test/addr 3
/add_sub_mul_div_test/net_1 40
/add_sub_mul_div_test/net_2 2
/add_sub_mul_div_test/net_3 -17
/add_sub_mul_div_test/net_4 4
/add_sub_mul_div_test/net_5 33
/add_sub_mul_div_test/net_6 4
/add_sub_mul_div_test/net_7 40
1 -2147483648 40 -17 -182 -80
0 1 2 3 4 0 1
1 -2147483648 40 -17 -182 -80
4 1 7 2 5 4 1
1 2147483647 10 -17 -26 -40 -47
21 5 7 4 8 21 5
1 11 22 33 44 55 11
1 2 3 4 5 1 2
10 20 30 40 50 10 20
Now 1 ns
Cursor 1 0.77 ns
0 ns 0.1 ns 0.2 ns 0.3 ns 0.4 ns 0.5 ns
wave

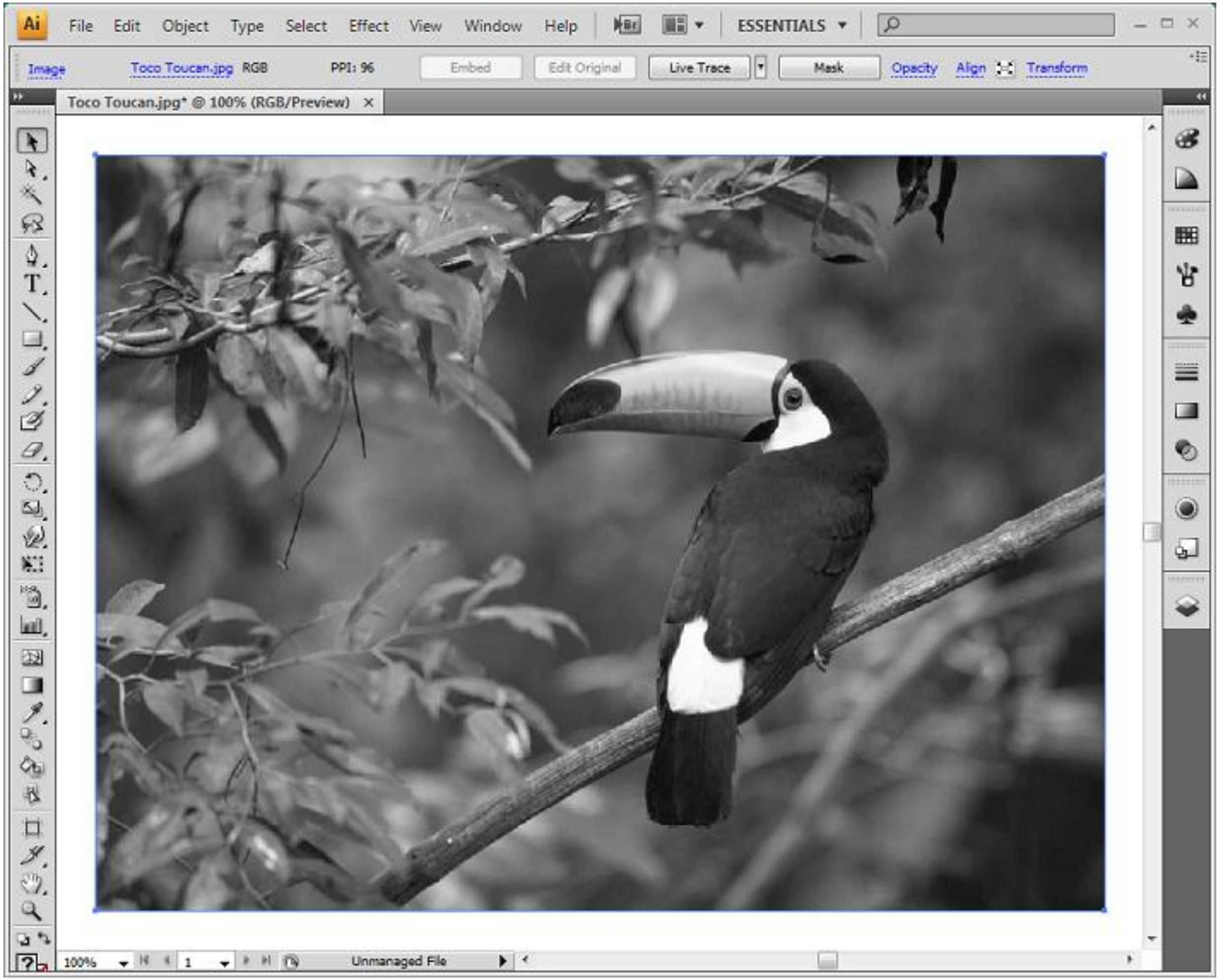


8.249ns max delay
3 x DSP48Es
63 slice registers
24 slice LUTs




ChipScope Pro Analyzer [convolver_chipscope]
File View JTAG Chain Device Trigger Setup Waveform Window Help
Project: convolver_chipscope
DEV:3 MyDevice3 (System_ACE_CF)
DEV:4 MyDevice4 (XC5VLX50T)
System Monitor Console
UNIT:0 MyILA0 (ILA)
Trigger Setup
Waveform
Listing
Signals: DEV: 4 UNIT: 0
Data Port
/net_53
/net_66_add0000
/Result
Trigger Ports
TriggerPort0
Trigger Setup - DEV:4 MyDevice4 (XC5VLX50T) UNIT:0 MyILA0 (ILA)
Match Unit
Function
Value
Radix
Counter
M0:TriggerPort0
==
F
Bin
disabled
Add
Del
Active
Trigger Condition Name
Trigger Condition Equation
TriggerCondition0
M0
Type: Window
Windows: 1
Depth: 1024
Position: 0
Storage Qualification: All Data
Waveform - DEV:4 MyDevice4 (XC5VLX50T) UNIT:0 MyILA0 (ILA)
Bus/Signal
X
O
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
/net_53
6318 6318
6318 3952 1846 91 263 506 749 992 1144 942 699 668
X: 0
O: 0
Δ(X-O): 0
COMMAND: set_match_function 4 0 0 0 3 1 F
COMMAND: set_trigger_condition 4 0 3 1 5555
COMMAND: set_storage_condition 4 0 FFFF
COMMAND: run 4 0
COMMAND: upload 4 0
INFO - Device 4 Unit 0: Waiting for core to be armed
Upload
DONE


File Edit Object Type Select Effect View Window Help
ESSENTIALS
Image Toco Toucan.jpg RGB
Live Trace
Mask
Opacity Align Transform
Desert Landscape.jpg @ 100% (RGB/Preview)
Apply Grayscale DX9 Shift+Ctrl+E
Grayscale DX9... Alt+Shift+Ctrl+E
Document Raster Effects Settings...
Illustrator Effects
3D
Convert to Shape
Crop Marks
Distort & Transform
Path
Pathfinder
Rasterize...
Stylize
SVG Filters
Warp
Photoshop Effects
Effect Gallery...
Artistic
Blur
Brush Strokes
Distort
Microsoft Research
Pixelate
Sharpen
Sketch
Stylize
Texture
Video
Blur (FPGA)...
Blur (Software)...
Grayscale (FPGA)...
Grayscale (Software)...
Grayscale DX9...
100%
Unmanaged File


Ai
File Edit Object Type Select Effect View Window Help
ESSENTIALS
Image Toco Toucan.jpg RGB PPI: 96 Embed Edit Original Live Trace Mask Opacity Align Transform
Toco Toucan.jpg* @ 100% (RGB/Preview)
100%
1
Unmanaged File

```
// Compute grayscale
Target &tgt = CreateDX9Target();
float* grayF = (float*) malloc(sizeof(float) * pixels) ;
FPA red   = FPA(redF,   rectHeight, rectWidth) ;
FPA green = FPA(greenF, rectHeight, rectWidth);
FPA blue  = FPA(blueF,  rectHeight, rectWidth);
FPA sum = Add (77 * red, Add (151 * green, 28 * blue)) ;
FPA gray = Divide (sum, 256) ;
tgt.ToArray(gray, grayF, rectHeight, rectWidth, rectWidth * sizeof(float));
// Update Photoshop image buffer
  pixel = (uint8*)data;
  for(int32 pixelY = 0; pixelY < rectHeight; pixelY++)
  {
        for(int32 pixelX = 0; pixelX < rectWidth; pixelX++)
        {
              uint8 gray = (uint8) grayF[pixelX+pixelY*rectWidth] ;
              pixel[0] = (uint8)gray ;
              pixel[1] = (uint8)gray ;
              pixel[2] = (uint8)gray ;
              pixel = pixel + 3 ;
              bigPixel++;
              fPixel++;
              dissolve++;
              if (maskPixel != NULL)
                    maskPixel++;
        }
        pixel += (dataRowBytes - 3*rectWidth);
        bigPixel += (dataRowBytes / 2 - 3*rectWidth);
        fPixel += (dataRowBytes / 4 - 3*rectWidth);
        if (maskPixel != NULL)
              maskPixel += (maskRowBytes - rectWidth);
  }
```

CUDA

```
//Compute and store results
   __syncthreads();
   #pragma unroll
   for(int i = ROWS_HALO_STEPS;
          i < ROWS_HALO_STEPS + ROWS_RESULT_STEPS; i++){
       float sum = 0;

       #pragma unroll
       for(int j = -KERNEL_RADIUS; j <= KERNEL_RADIUS; j++)
           sum += c_Kernel[KERNEL_RADIUS - j] *
s_Data[threadIdx.y][threadIdx.x + i * ROWS_BLOCKDIM_X + j];

       d_Dst[i * ROWS_BLOCKDIM_X] = sum;
   }
```

Blogs Home

Sign in | Join

Search
RSS
OPML

Satnam Singh's MSDN Blog

GPGPU and x64 Multicore Programming with Accelerator from F#

Microsoft recently released a preview of the Accelerator V2 GPU and x64 multicore programming system on Microsoft Connect. This system provides a civilized level of abstraction for writing data-parallel programs that execute on GPUs and multicore processors. An experimental FPGA target is under development.

Even on my low end graphics card I get pretty impressive performance results for the 2D convolver that is described in this blog. All 8 cores of my 64-bit Windows 7 workstation are also effectively exercised by the x64 multicore target, which exploits SIMD processor instructions and multithreading. I wonâ t say anything about performance in this blog post since what I want to focus on is how to use Accelerator from the F# functional programming language. We will work backwards by starting off with a complete implementation of a two dimensional convolver. Step by step we show how this convolver is expressed using Accelerator from F#.

First here is the beautiful implementation of a two dimensional convolver. The rest of this post explains why this code works.

```
open System
open Microsoft.ParallelArrays

[<EntryPoint>]
let main(args) =

    // Declare a filter kernel for the convolution
    let testKernel = Array.map float32 [| 2; 5; 7; 4; 3 |]

    // Specify the size of each dimension of the input array
    let inputSize = 10

    // Create a pseudo-random number generator
    let random = Random (42)

    // Declare a psueduo-input data array
    let testData = Array2D.init inputSize inputSize (fun i j -> float32 (random.NextDouble() *
                                                           float (random.Next(1, 100))))

    // Create an Accelerator float parallel array for the F# input array
    use testArray = new FloatParallelArray(testData)

    // Declare a function to convolve in the X or Y direction
    let rec convolve (shifts : int -> int []) (kernel : float32 []) i (a : FloatParallelArray)
       = let e = kernel.[i] * ParallelArrays.Shift(a, shifts i)
         if i = 0 then
           e
         else
           e + convolve shifts kernel (i-1) a

    // Declare a 2D convolver
    let convolveXY kernel input
       = // First convolve in the X direction and then in the Y direction
         let convolveX = convolve (fun i -> [| -i; 0 |]) kernel (kernel.Length - 1) input
         let convolveY = convolve (fun i -> [| 0; -i |]) kernel (kernel.Length - 1) convolveX
         convolveY
```

This Blog
Home
Email
Links

Syndication
RSS 2.0
Atom 1.0

Recent Posts
A C# implementation of a convolver using Accelerator for GPGPU and multicore targets using LINQ operator
An F# Functional Geometry Description of Escher's Fish
Distracted by Abstraction - A poem about F#
GPGPU and x64 Multicore Programming with Accelerator from F#

Tags
No tags have been created or used yet.

Archives
January 2010 (2)
December 2009 (2)

Search for “Microsoft Accelerator V2”