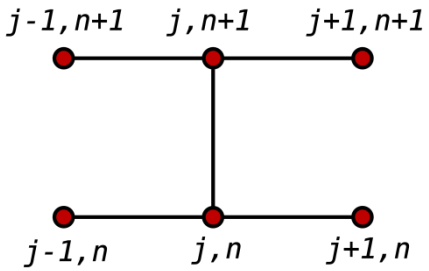
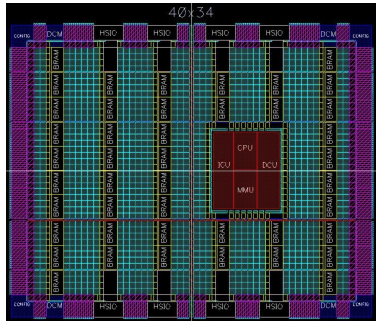
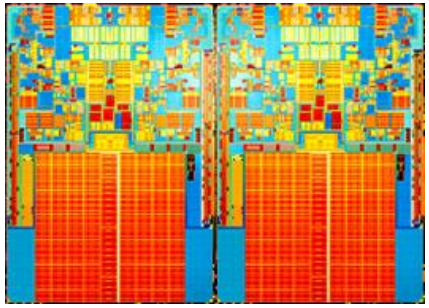
Microsoft Research

Synthesis of Data-Parallel GPU Software into GPU code, Multicore Software and FPGA Hardware







Satnam Singh
Microsoft Research, Cambridge UK

Microsoft Research

BIOINFORMATICS ORIGINAL PAPER

Vol. 23 no. 2 2007, pages 156–161
doi:10.1093/bioinformatics/btl582

Sequence analysis

Striped Smith–Waterman speeds database searches six times over other SIMD implementations

Michael Farrar

Received on June 22, 2006; revised on November 13, 2006; accepted on November 14, 2006
Advance Access publication November 16, 2006
Associate Editor: Nikolaus Rajewsky

ABSTRACT

Motivation: The only algorithm guaranteed to find the optimal local alignment is the Smith–Waterman. It is also one of the slowest due to the number of computations required for the search. To speed up the algorithm, Single-Instruction Multiple-Data (SIMD) instructions have been used to parallelize the algorithm at the instruction level.

Results: A faster implementation of the Smith–Waterman algorithm is presented. This algorithm achieved 2–8 times performance improvement over other SIMD based Smith–Waterman implementations. On a 2.0 GHz Xeon Core 2 Duo processor, speeds of >3.0 billion cell updates/s were achieved.

Availability: http://farrar.michael.googlepages.com/Smith-waterman

Contact: farrar.michael@gmail.com

search. A disadvantage introduced by processing the values vertically is that conditional branches are placed in the inner loop to compute F. With conditional code the execution time is dependent on the length of the query string and the database, the scoring matrix and gap penalties. A speedup of over six times was reported over an optimized non-SIMD implementation.

This paper presents a new Smith–Waterman implementation where the SIMD registers are parallel to the query sequence, but are accessed in a striped pattern. Like the Rognes implementation, the query profile is calculated once for the database search, but the conditional F calculations are moved outside the inner loop. Calculations speeds of >3.0 GCUPS are achieved. This is a speedup of 2–8 times over the Wozniak and Rognes SIMD implementations.

BMC Bioinformatics

BioMed Central

Research

Open Access

CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment

Svetlin A Manavski*[1,2] and Giorgio Valle[1]

Address: [1]CRIBI, University of Padova, Padova, Italy and [2]Elaide, Srl, Padova, Italy

Email: Svetlin A Manavski* - svetlin.manavski@cribi.unipd.it; Giorgio Valle - giorgio.valle@unipd.it

* Corresponding author

from Italian Society of Bioinformatics (BITS): Annual Meeting 2007
Naples, Italy. 26-28 April 2007

Published: 26 March 2008

BMC Bioinformatics 2008, **9**(Suppl 2):S10 doi:10.1186/1471-2105-9-S2-S10

This article is available from: http://www.biomedcentral.com/1471-2105/9/S2/S10

Abstract

Background: Searching for similarities in protein and DNA databases has become a routine procedure in Molecular Biology. The Smith-Waterman algorithm has been available for more than 25 years. It is based on a dynamic programming approach that explores all the possible alignments between two sequences; as a result it returns the optimal local alignment. Unfortunately, the computational cost is very high, requiring a number of operations proportional to the product of the length of two sequences. Furthermore, the exponential growth of protein and DNA databases makes the Smith-Waterman algorithm unrealistic for searching similarities in large sets of sequences. For these reasons heuristic approaches such as those implemented in FASTA and BLAST tend to be preferred, allowing faster execution times at the cost of reduced sensitivity. The main motivation of our work is to exploit the huge computational power of commonly available graphic cards, to develop high performance solutions for sequence alignment.

Results: In this paper we present what we believe is the fastest solution of the exact Smith-Waterman algorithm running on commodity hardware. It is implemented in the recently released CUDA programming environment by NVidia. CUDA allows direct access to the hardware primitives of the last-generation Graphics Processing Units (GPU) G80. Speeds of more than 3.5 GCUPS (Giga Cell Updates Per Second) are achieved on a workstation running two GeForce 8800 GTX. Exhaustive tests have been done to compare our implementation to SSEARCH and BLAST, running on a 3 GHz Intel Pentium IV processor. Our solution was also compared to a recently published GPU implementation and to a Single Instruction Multiple Data (SIMD) solution. These tests show that our implementation performs from 2 to 30 times faster than any other previous attempt available on commodity hardware.

BMC Bioinformatics

BioMed Central

Methodology article

Open Access

160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)

Isaac TS Li[1], Warren Shum[2] and Kevin Truong*[1,2]

Address: [1]Institute of Biomaterials and Biomedical Engineering, University of Toronto, 164 College Street, Toronto, Ontario, M5S 3G9, Canada and [2]Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Circle, Toronto, Ontario, M5S 3G4, Canada

Email: Isaac TS Li - isaac.li@utoronto.ca; Warren Shum - warren.shum@utoronto.ca; Kevin Truong* - kevin.truong@utoronto.ca

* Corresponding author

Published: 7 June 2007

BMC Bioinformatics 2007, **8**:185 doi:10.1186/1471-2105-8-185

Received: 19 February 2007
Accepted: 7 June 2007

This article is available from: http://www.biomedcentral.com/1471-2105/8/185

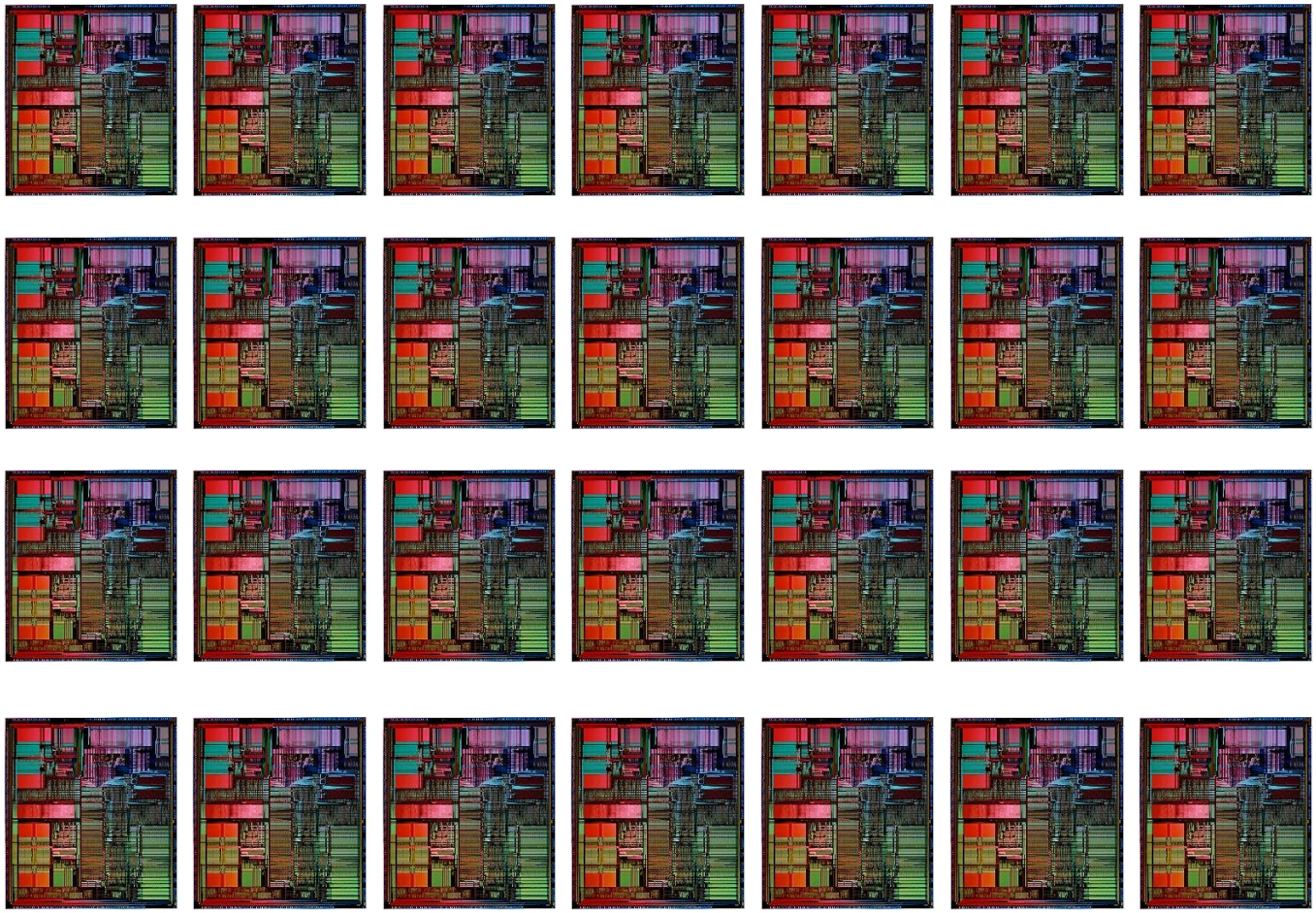
Abstract

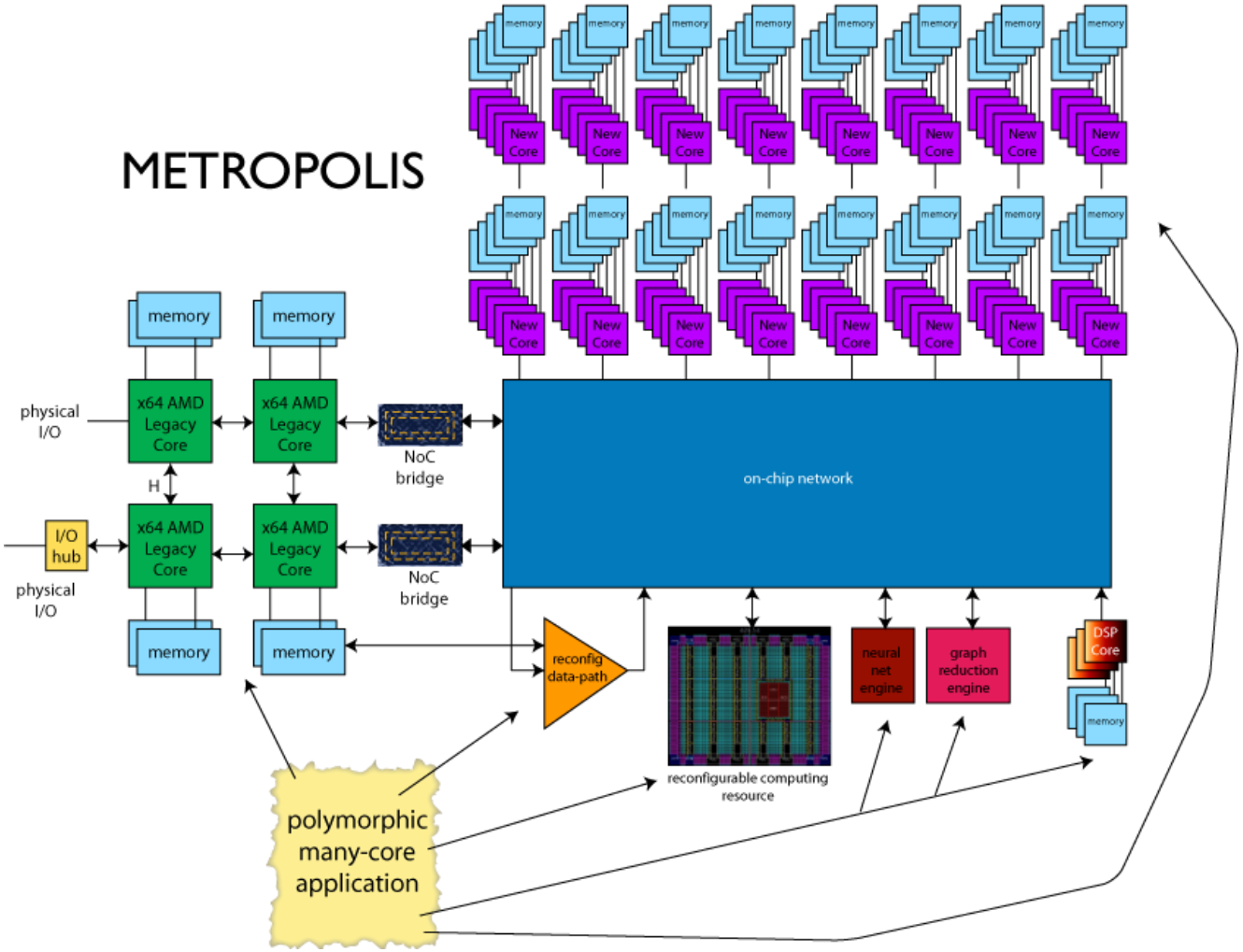
Background: To infer homology and subsequently gene function, the Smith-Waterman (SW) algorithm is used to find the optimal local alignment between two sequences. When searching sequence databases that may contain hundreds of millions of sequences, this algorithm becomes computationally expensive.

Results: In this paper, we focused on accelerating the Smith-Waterman algorithm by using FPGA-based hardware that implemented a module for computing the score of a single cell of the SW matrix. Then using a grid of this module, the entire SW matrix was computed at the speed of field propagation through the FPGA circuit. These modifications dramatically accelerated the algorithm's computation time by up to 160 folds compared to a pure software implementation running on the same FPGA with an Altera Nios II softprocessor.

Conclusion: This design of FPGA accelerated hardware offers a new promising direction to seeking computation improvement of genomic database searching.

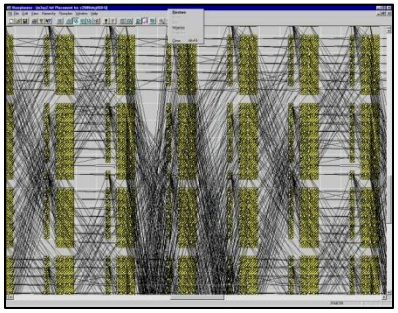
Microsoft Research

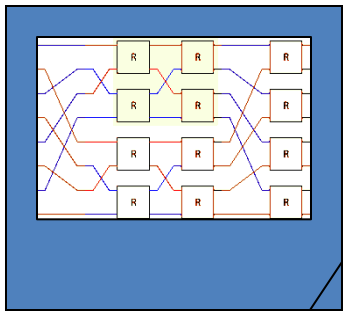



METROPOLIS
memory
New Core
x64 AMD Legacy Core
physical I/O
H
I/O hub
NoC bridge
on-chip network
reconfig data-path
reconfigurable computing resource
neural net engine
graph reduction engine
DSP Core
polymorphic many-core application

locks
monitors
condition variables
spin locks
priority inversion

FPGA
hardware
(VHDL, ISE)





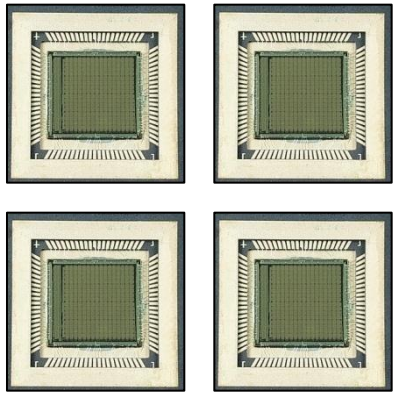
GPU code (HLSL, DX9)

Machine

data parallel
Descriptions
C++, C#, F#…

Collection

SSE3



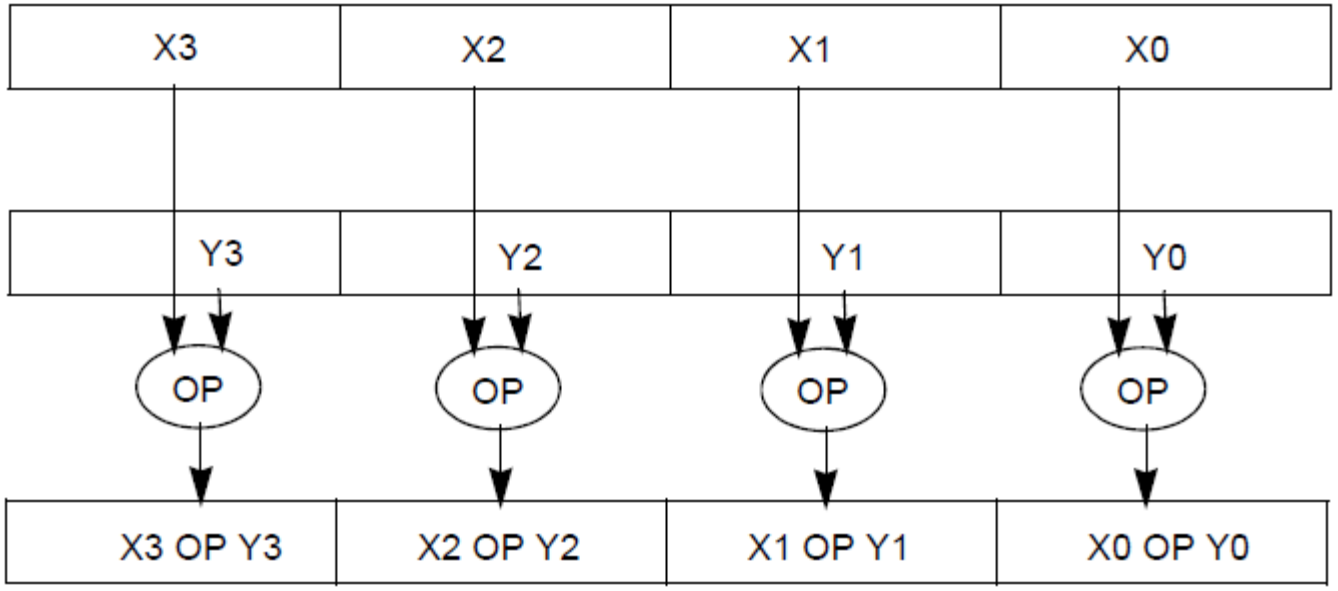
SSE3
X64
multicore

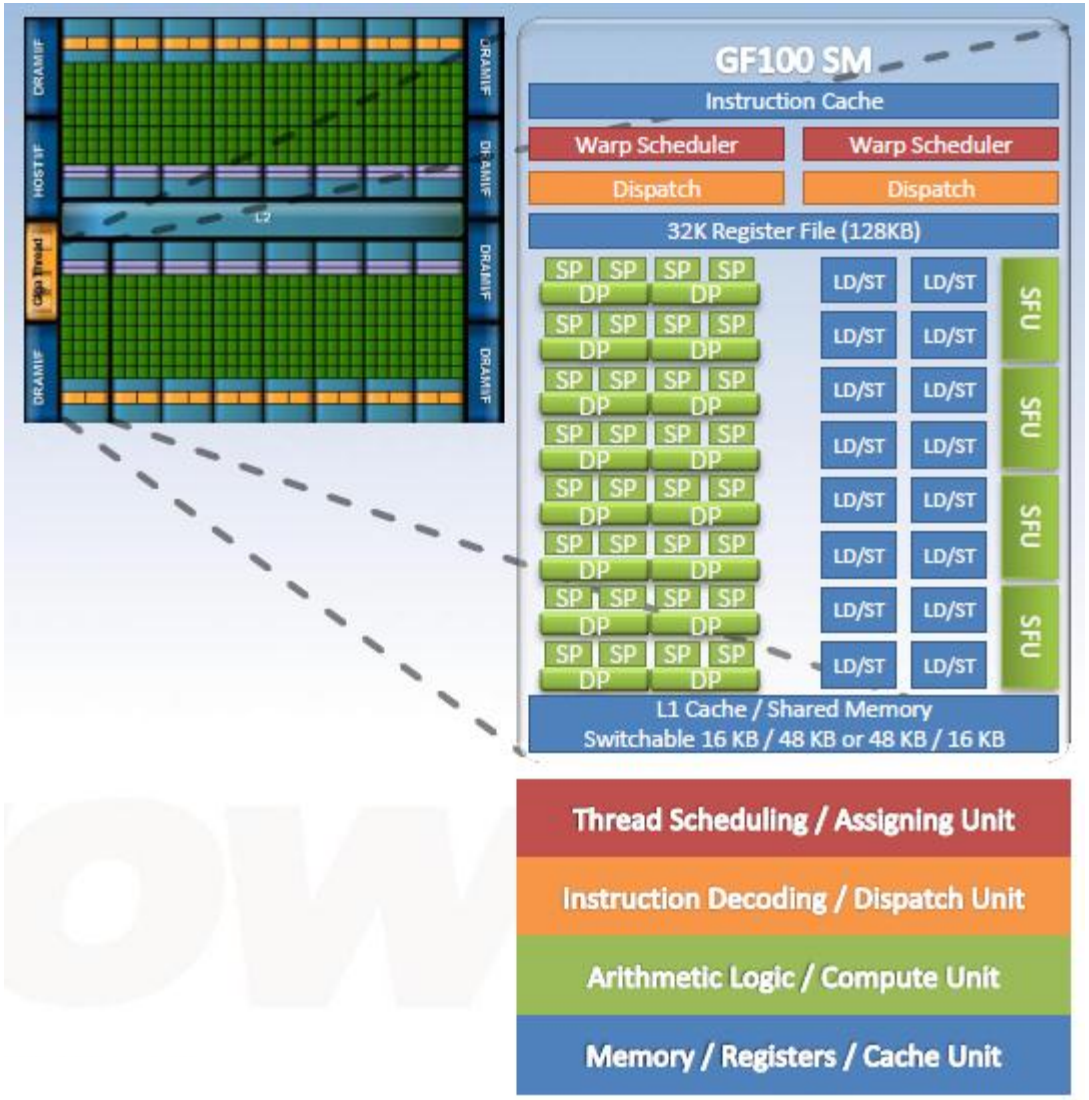
SSE2: ADDPS

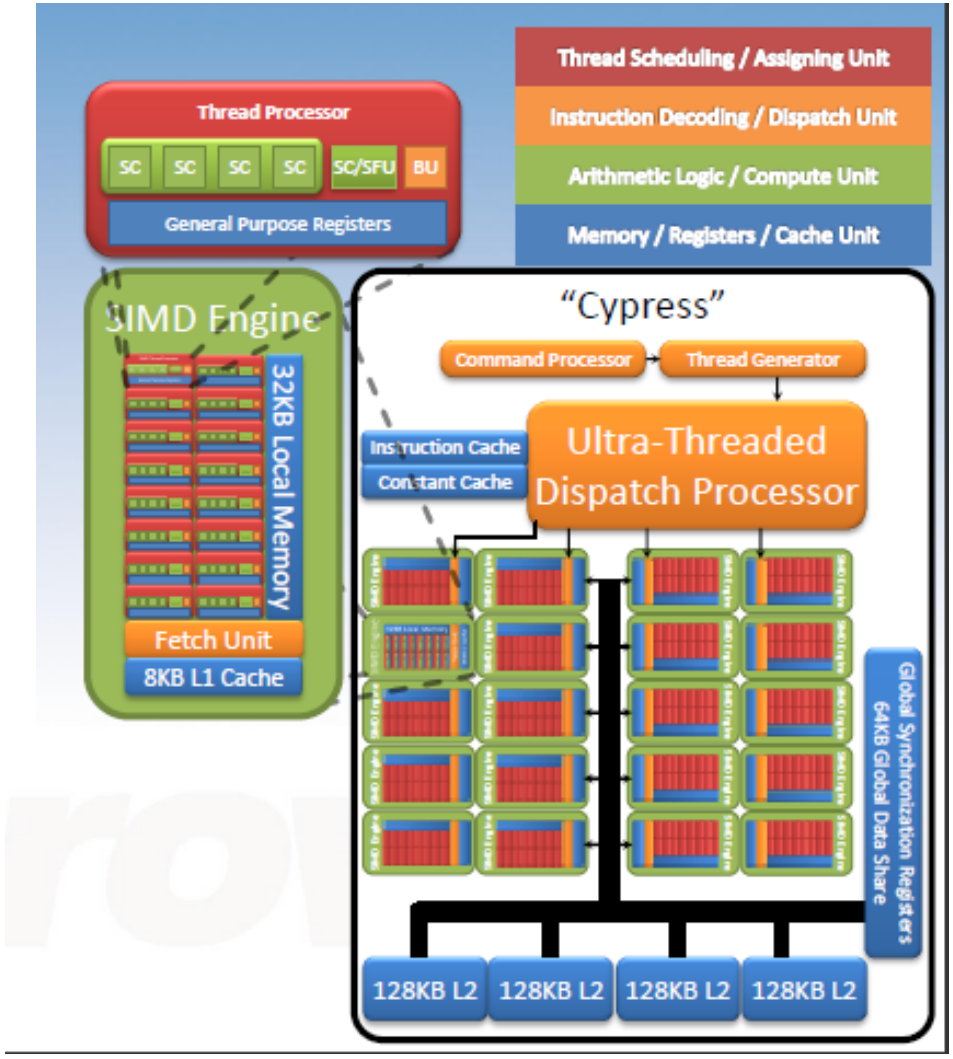
__m128 _mm_add_ps (__m128 a , __m128 b);

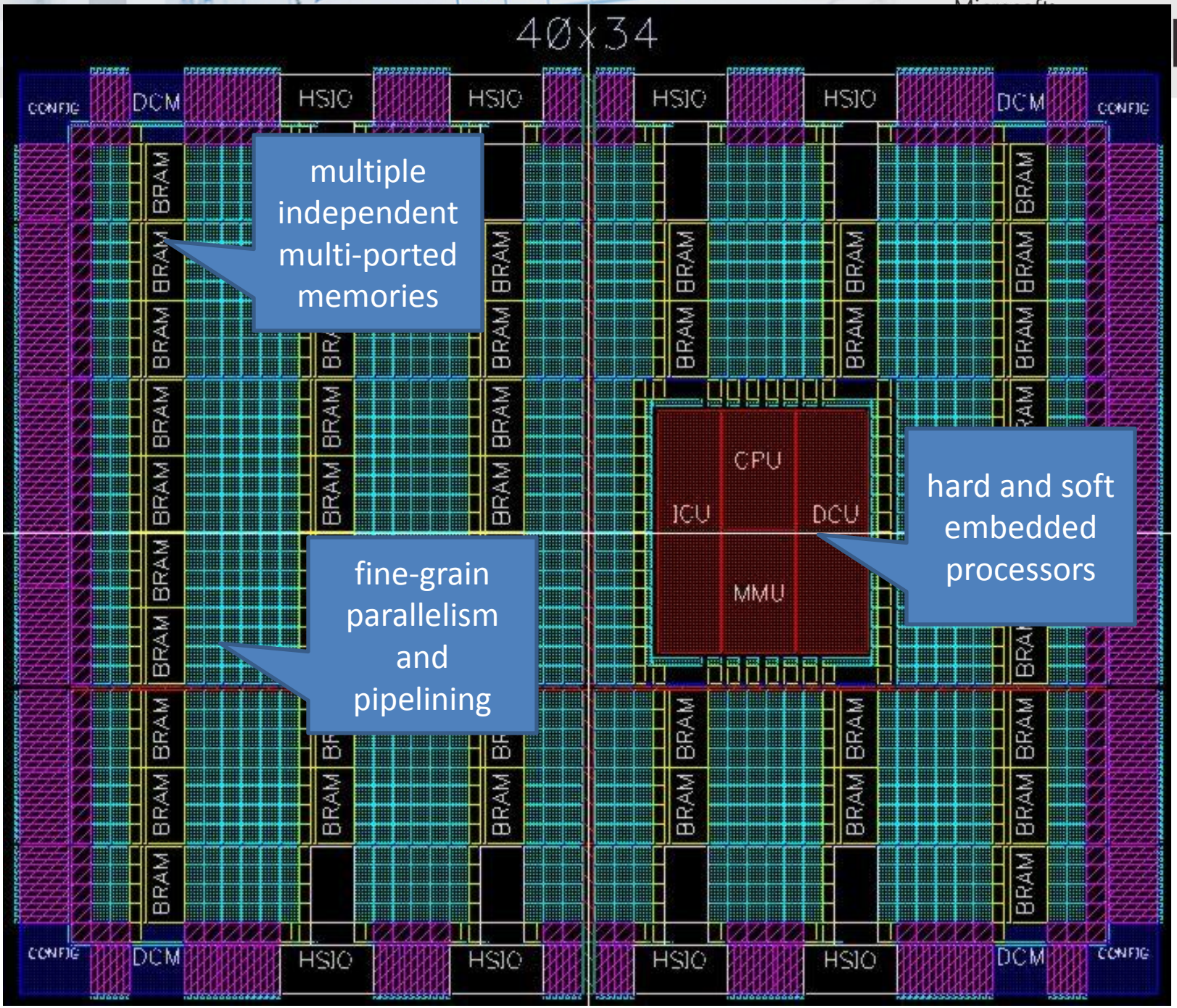
r0 := x0 + y0
r1 := x1 + y1
r2 := x2 + y2
r3 := x3 + y3

128-bits MMX/

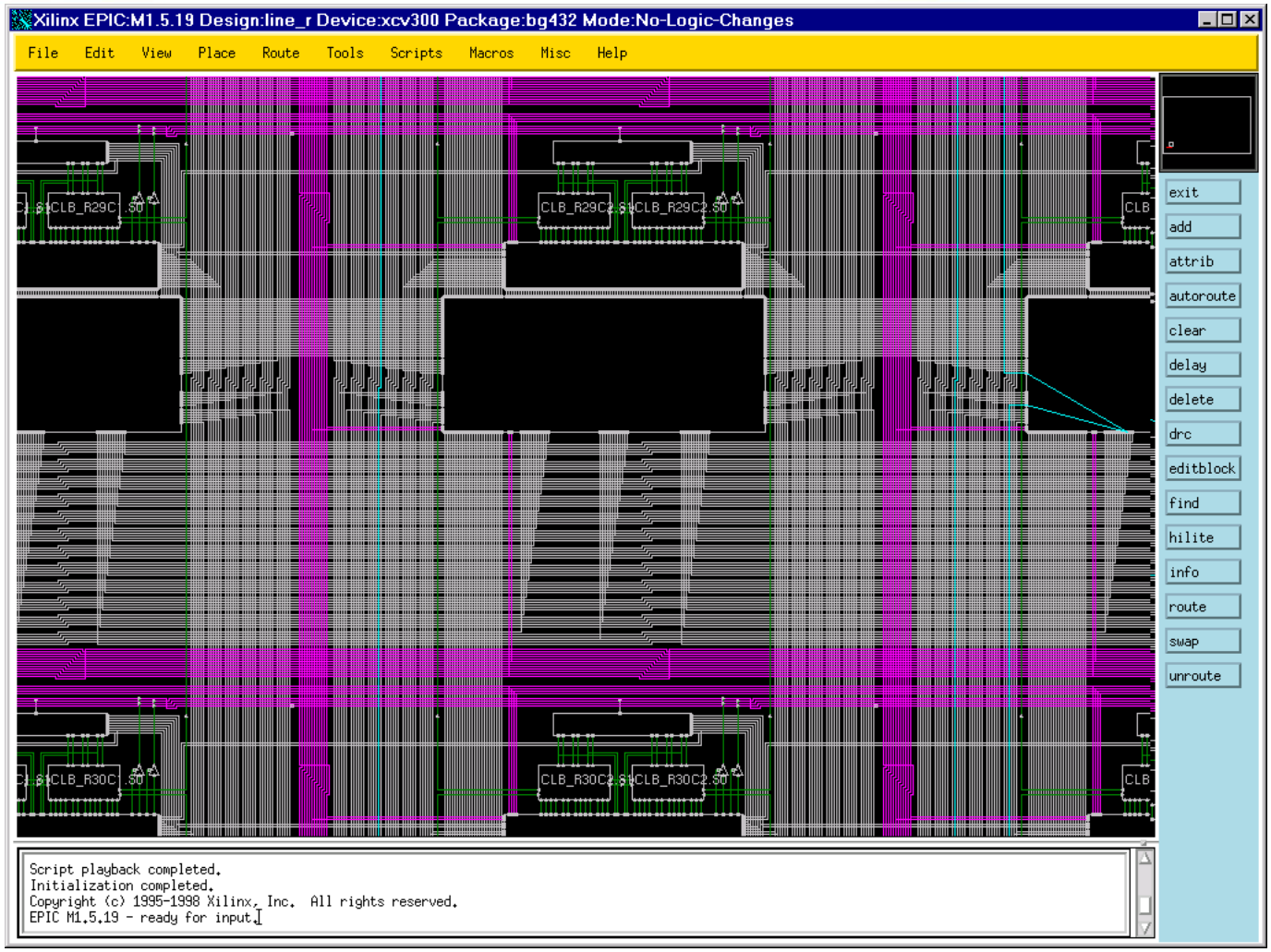



GF100 SM
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch
Dispatch
32K Register File (128KB)
SP
DP
LD/ST
SFU
L1 Cache / Shared Memory
Switchable 16 KB / 48 KB or 48 KB / 16 KB
Thread Scheduling / Assigning Unit
Instruction Decoding / Dispatch Unit
Arithmetic Logic / Compute Unit
Memory / Registers / Cache Unit


Thread Scheduling / Assigning Unit
Instruction Decoding / Dispatch Unit
Arithmetic Logic / Compute Unit
Memory / Registers / Cache Unit
Thread Processor
SC
SC
SC
SC
SC/SFU
BU
General Purpose Registers
SIMD Engine
32KB Local Memory
Fetch Unit
8KB L1 Cache
"Cypress"
Command Processor
Thread Generator
Instruction Cache
Constant Cache
Ultra-Threaded Dispatch Processor
Global Synchronization Registers
64KB Global Data Share
128KB L2
128KB L2
128KB L2
128KB L2


40x34
CONFIG
DCM
HSIO
BRAM
CPU
ICU
DCU
MMU
multiple independent multi-ported memories
fine-grain parallelism and pipelining
hard and soft embedded processors

Microsoft Research



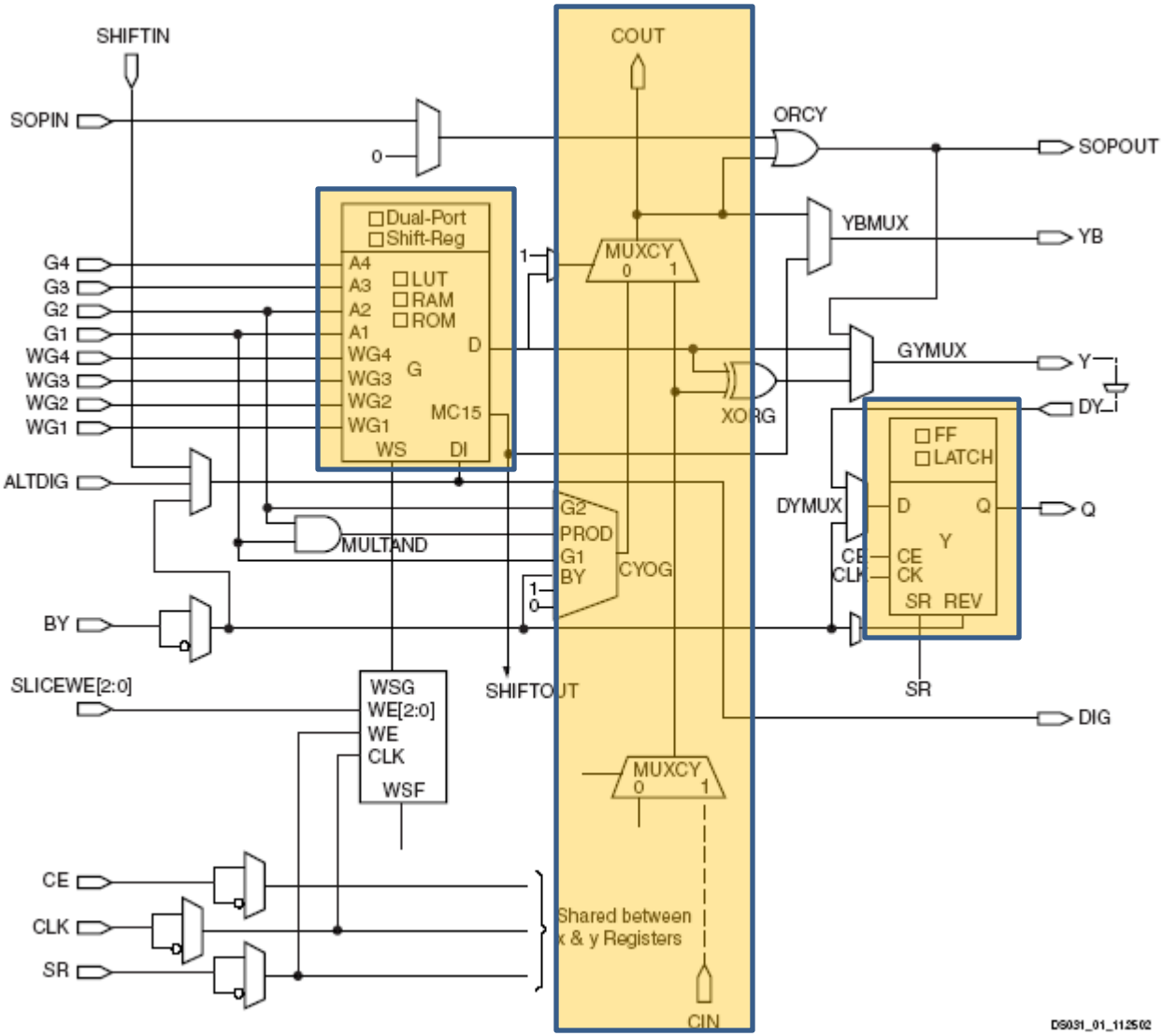
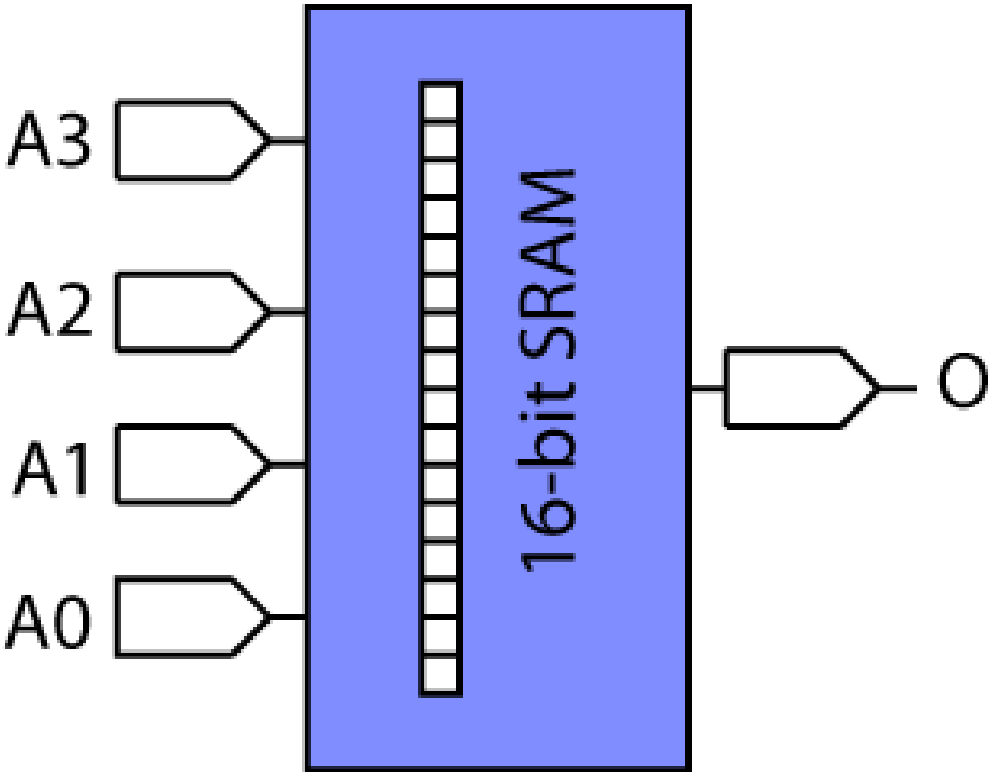
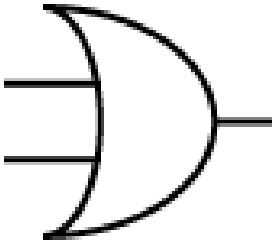


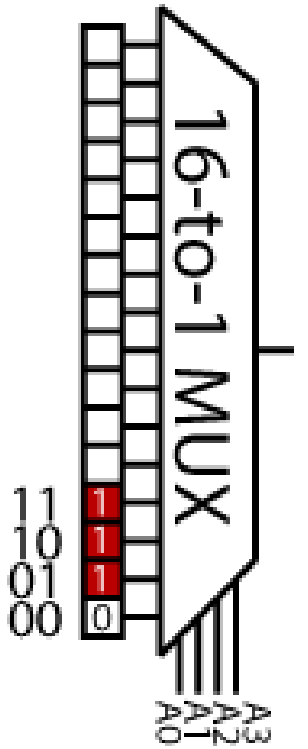
Figure 16: **Virtex-II Slice (Top Half)**

LUT4 (OR)

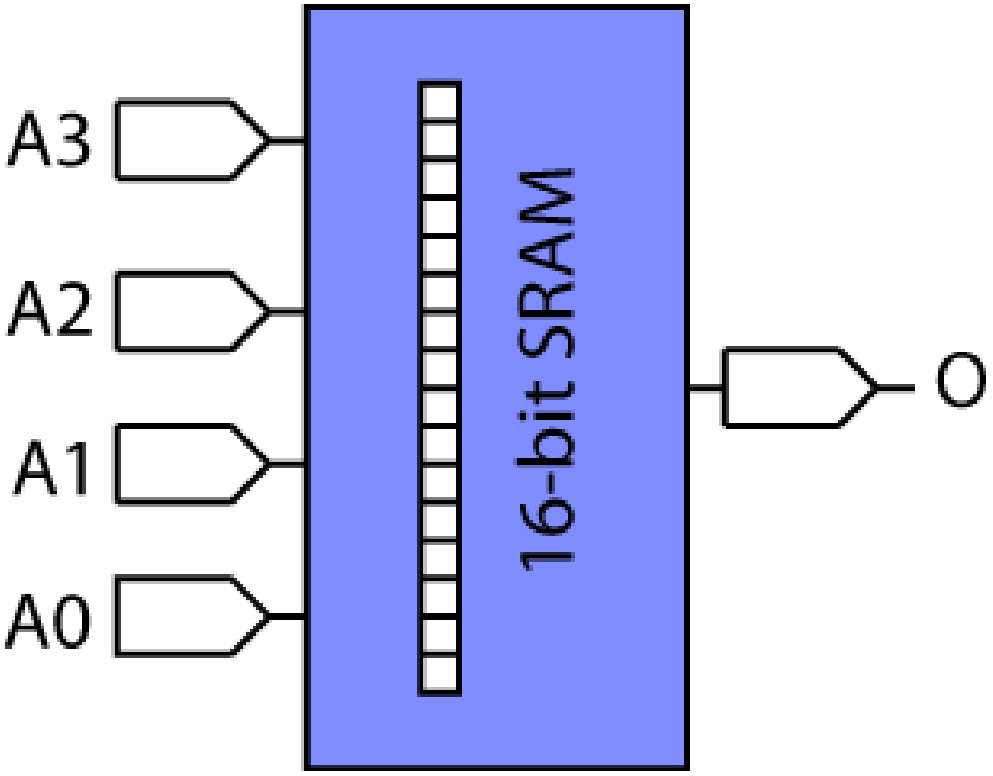


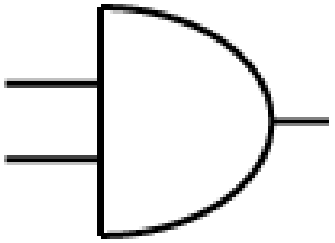


A1	A0	O
0	0	0
0	1	1
1	0	1
1	1	1

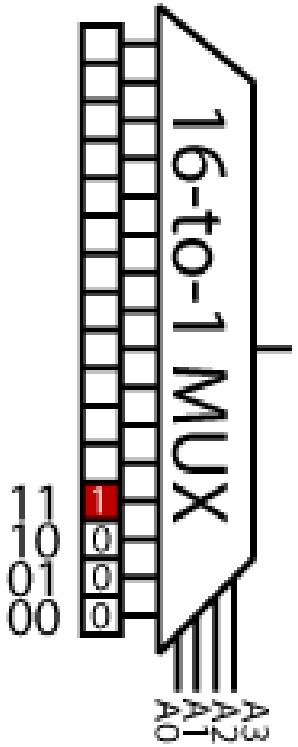


LUT4 (AND)

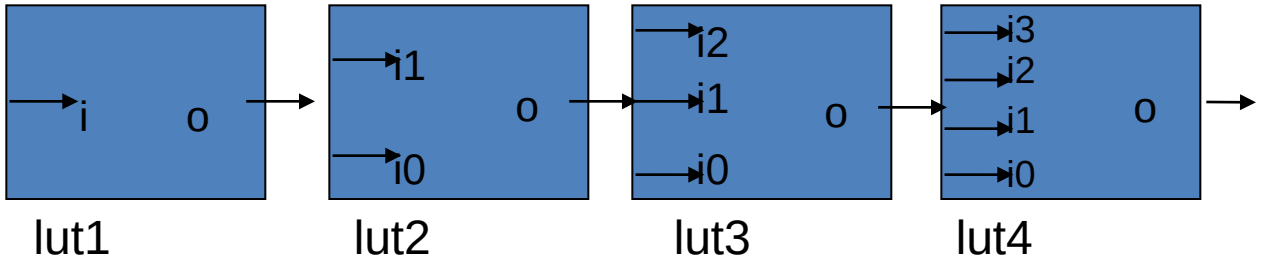




A1	A0	O
0	0	0
0	1	0
1	0	0
1	1	1



LUTs are higher order functions

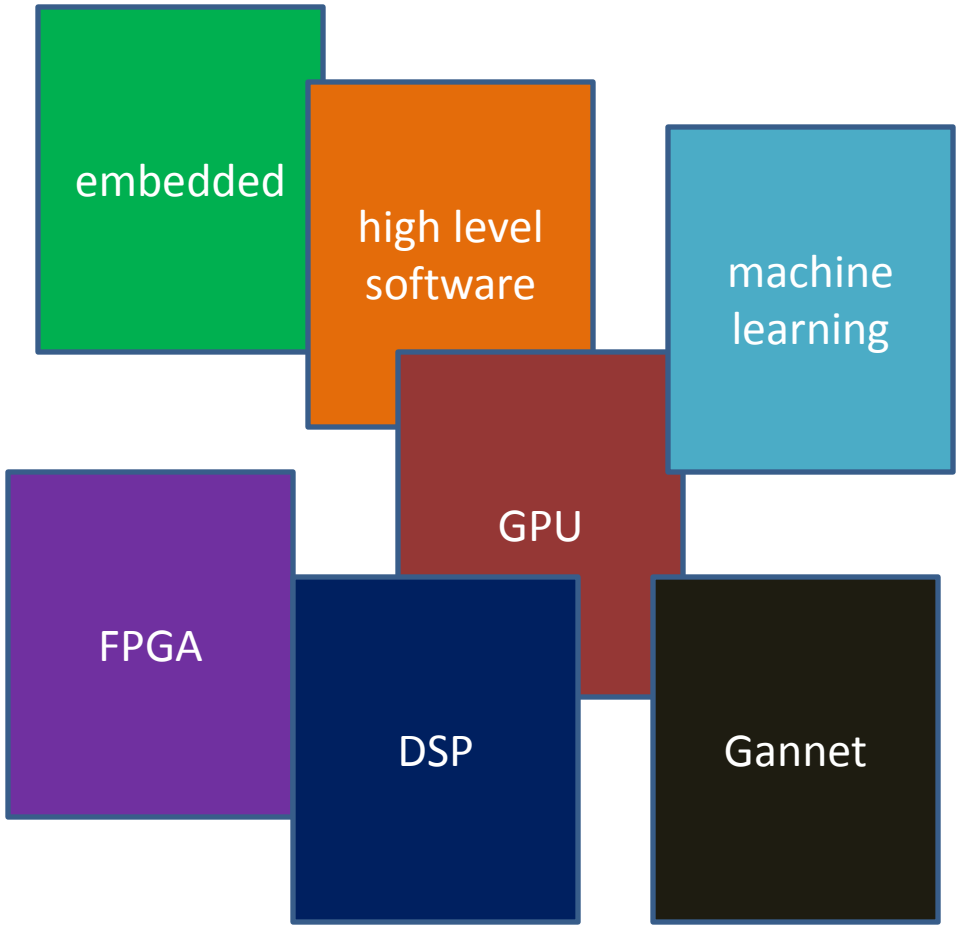


inv = **lut1** not

and2 = **lut2** (&&)

mux = **lut3** (λ s d0 d1 . **if** s **then** d1 **else** d0)

universal language?



grand unification theory

polygots

Microsoft Research

Self Imposed Constraints

Effort vs. Reward

CUDA
OpenCL
HLSL
DirectCompute

Accelerator

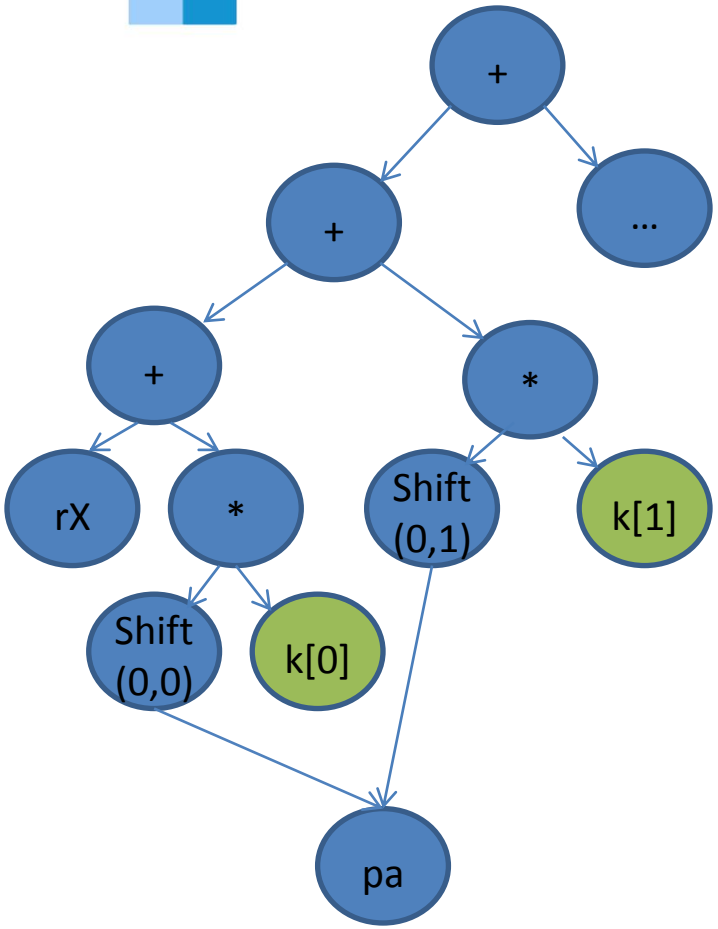
low effort	medium effort	high effort
low reward	medium reward	high reward

```
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();
        }
    }
}
```

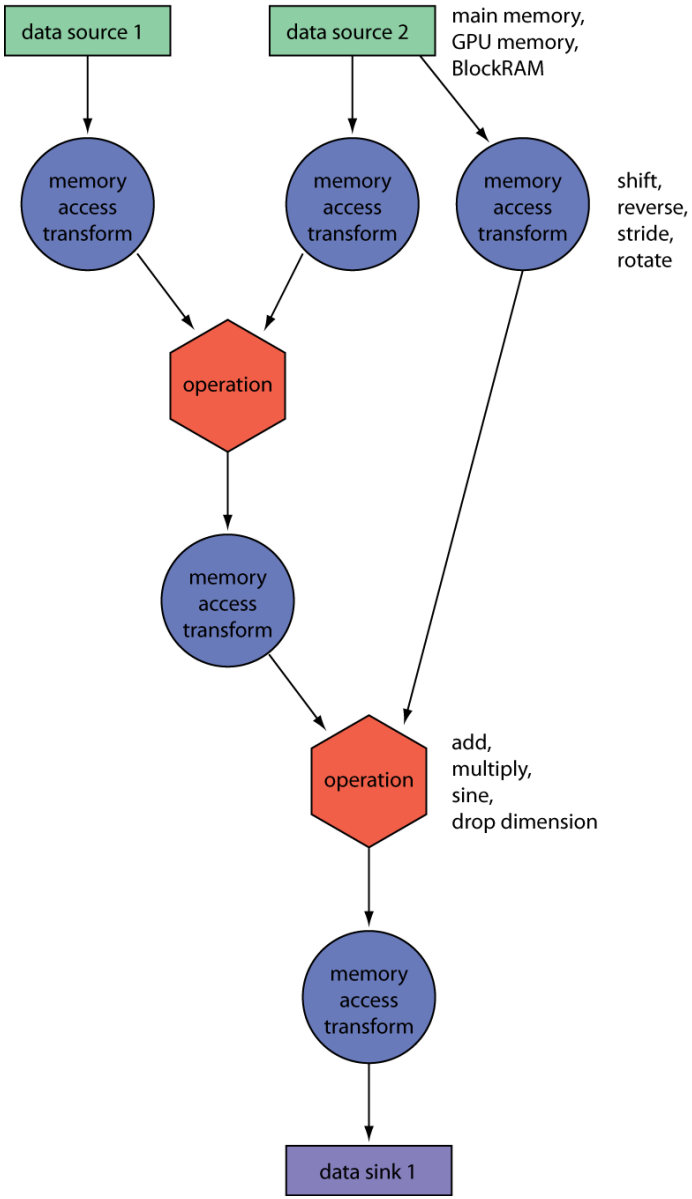
```
using System;
using Microsoft.ParallelArrays;

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) ;
        }
    }
}
```

```
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
```




```
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

```
normcdf :: Float → Float
normcdf x | x < 0       = 1 − w
          | otherwise   = w
  where
    w     = 1.0 − 1.0 / sqrt (2.0 ∗ π) ∗
            exp (−l ∗ l / 2.0) ∗ poly k
    k     = 1.0 / (1.0 + 0.2316419 ∗ l)
    l     = abs x
    poly  = horner coeff
    coeff = [0.0, 0.31938153,
             − 0.356563782, 1.781477937,
             − 1.821255978, 1.330274429]

horner coeff x = foldr1 madd coeff
  where
    madd a b = b ∗ x + a
```

```
blackscholes :: Vector Float    -- Stock prices
             → Vector Float     -- Option strikes
             → Vector Float     -- Option years
             → Vector Float
blackscholes ss xs ts =
    zipWith3 (λs x t → blackscholes1 s x t r v)
      ss xs ts
  where
    r = ...
    v = ...

blackscholes1 :: Float    -- Stock price
              → Float     -- Option strike
              → Float     -- Option years
              → Float     -- Riskless rate
              → Float     -- Volatility rate
              → Float
blackscholes1 s x t r v =
    s * normcdf d1 − x * exp (−r * t) * normcdf d2
  where
    d1 = (log (s / x) + (r + v * v / 2) * t) / (v * sqrt t)
    d2 = d1 − v * sqrt t
```

```
static float Horner(float[] coe, float x)
{
    float result = 0.0f;
    foreach (var c in coe)
    {
        result = result + x * c;
    }
    return result;
}
```

```
static FloatParallelArray Horner(float[] coe, FloatParallelArray x)
{
    FloatParallelArray result = new FloatParallelArray(0.0f, x.Shape);
    foreach (var c in coe)
    {
        result = result + x * c;
    }
    return result;
}
```

```
static float NormCdf(float x)
{
    var coe = new []{ 0.0f, 0.31938153f, 0.356563782f, 1.781477937f, 1.821255978f, 1.330274429f };
    float poly = Horner(coe, x);
    float l = Math.Abs(x);
    float k = (float) (1.0f/(1.0 + 0.2316419f*l));
    float w = (float)(1.0f - 1.0f / Math.Sqrt(2.0f * Math.PI) * Math.Exp(-l * l / 2.0f) *
               poly * k);
    if (x < 0)
        return 1.0f - w;
    else
        return w;
}
```

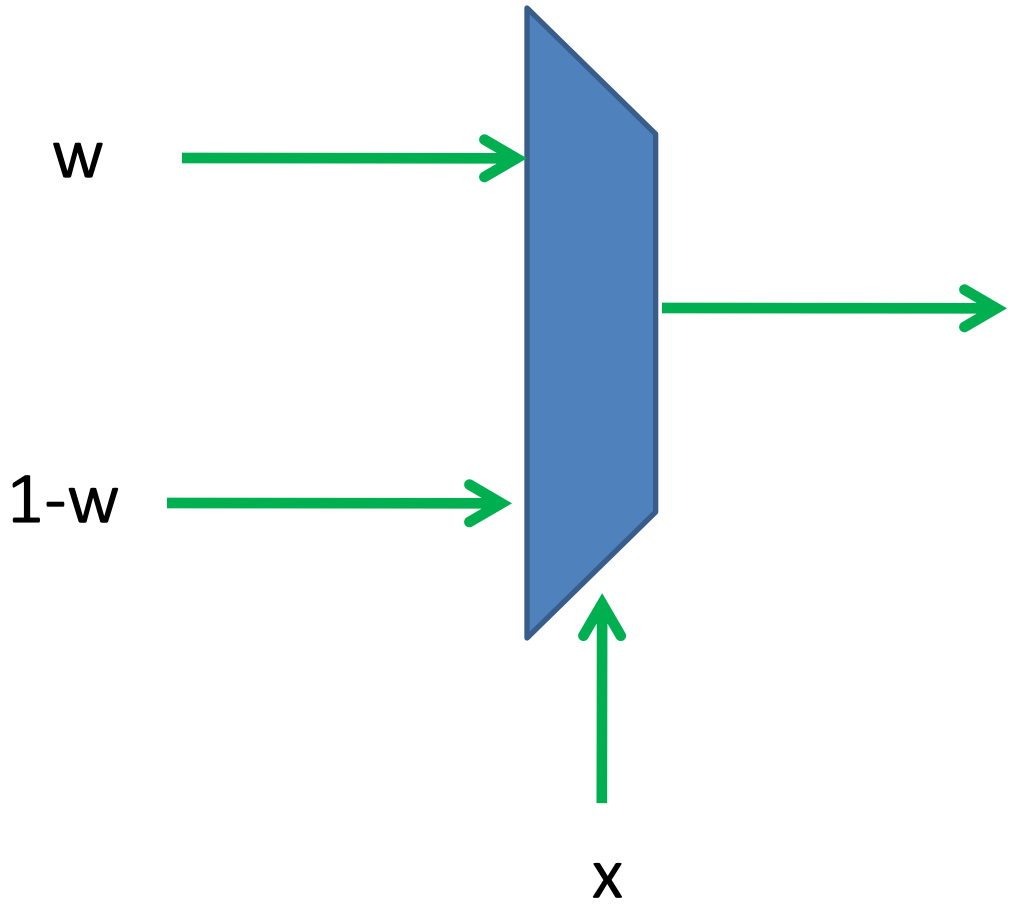
```
static FloatParallelArray NormCdf(FloatParallelArray x)
{
    var coe = new[] { 0.0f, 0.31938153f, 0.356563782f, 1.781477937f, 1.821255978f, 1.330274429f };
    FloatParallelArray poly = Horner(coe, x);
    FloatParallelArray l = ParallelArrays.Abs(x);
    FloatParallelArray k = 1.0f / (1.0f + 0.2316419f * l);
    FloatParallelArray e = new FloatParallelArray(2.718281828459045f, l.Shape);
    FloatParallelArray w = 1.0f - 1.0f / (float)(Math.Sqrt(2.0f * Math.PI)) *
                       ParallelArrays.Pow(e, -l * l / 2.0f) * poly * k;
    return ParallelArrays.Select(x, w, 1.0f - w);
}
```

```
static float NormCdf(float x)
{
    var coe = new []{ 0.0f, 0.31938153f, 0.356563782f, 1.781477937f, 1.821255978f, 1.330274429f };
    float poly = Horner(coe, x);
    float l = Math.Abs(x);
    float k = (float) (1.0f/(1.0 + 0.2316419f*l));
    float w = (float)(1.0f - 1.0f / Math.Sqrt(2.0f * Math.PI) * Math.Exp(-l * l / 2.0f) *
               poly * k);
    if (x < 0)
        return 1.0f - w;
    else
        return w;
}
```

```
static FloatParallelArray NormCdf(FloatParallelArray x)
{
    var coe = new[] { 0.0f, 0.31938153f, 0.356563782f, 1.781477937f, 1.821255978f, 1.330274429f };
    FloatParallelArray poly = Horner(coe, x);
    FloatParallelArray l = ParallelArrays.Abs(x);
    FloatParallelArray k = 1.0f / (1.0f + 0.2316419f * l);
    FloatParallelArray e = new FloatParallelArray(2.718281828459045f, l.Shape);
    FloatParallelArray w = 1.0f - 1.0f / (float)(Math.Sqrt(2.0f * Math.PI)) *
                       ParallelArrays.Pow(e, -l * l / 2.0f) * poly * k;
    return ParallelArrays.Select(x, w, 1.0f - w);
}
```

```
if (x < 0)
   return 1.0f - w;
else
   return w;
```

```
ParallelArrays.Select(x, w, 1.0f - w);
```

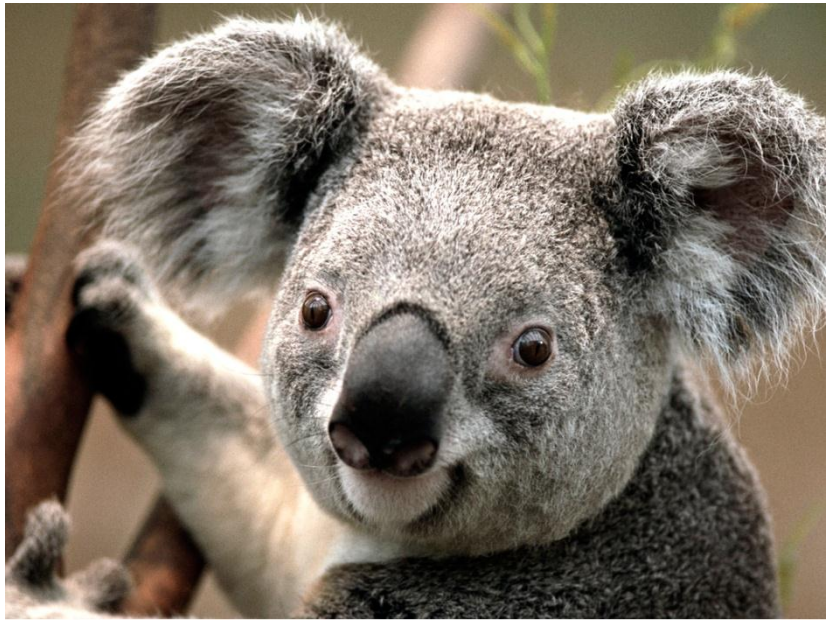

w
1-w
x

```
static float BlackCholes1(float s, float x, float t, float r, float v)
{
    float d1 = (float)((Math.Log(s / x) +
                        (r + v * v / 2) * t) / (v * Math.Sqrt(t)));
    float d2 = (float)(d1 - v * Math.Sqrt(t));
    return (float)(s * NormCdf(d1) - x * Math.Exp(-r * t) * NormCdf(d2));
}
```

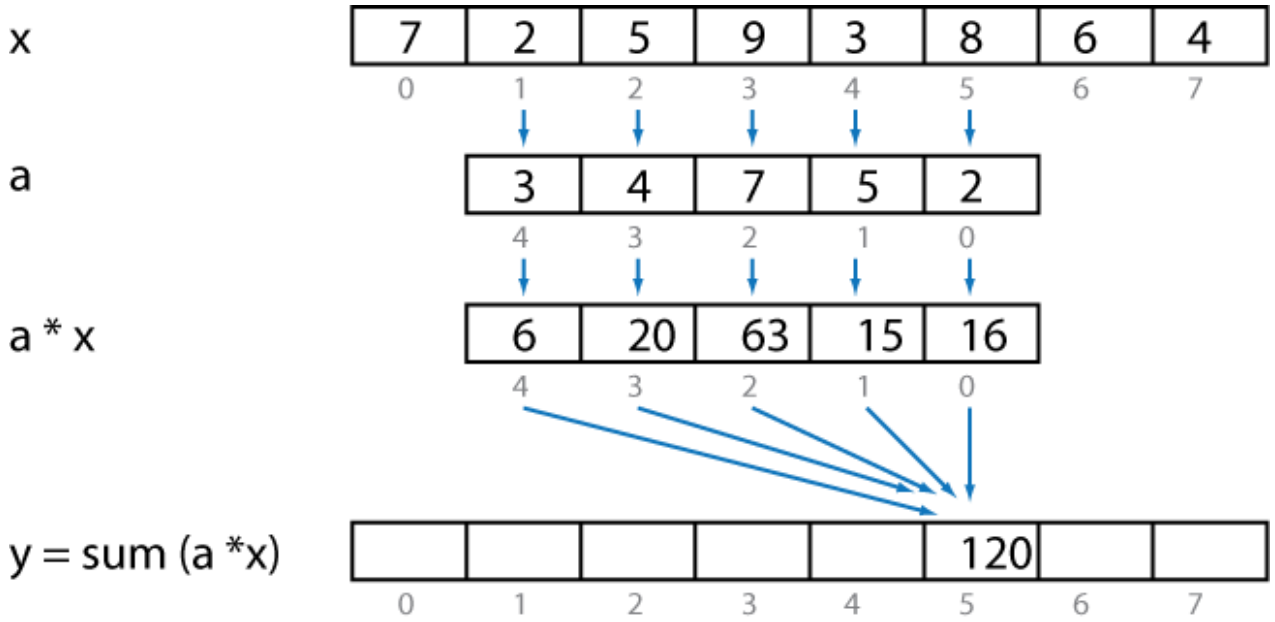
```
static FloatParallelArray BlackCholes1(FloatParallelArray ss,
                                       FloatParallelArray xs,
                                       FloatParallelArray ts, float r, float v)
{
    FloatParallelArray d1 = ParallelArrays.Log2(ss / xs) +
                            ((r + v * v / 2) * ts) / (v * ParallelArrays.Sqrt(ts));
    FloatParallelArray d2 = (d1 - v * ParallelArrays.Sqrt(ts));
    FloatParallelArray e = new FloatParallelArray(2.718281828459045f, ts.Shape);
    return (ss * NormCdf(d1) - xs * ParallelArrays.Pow(e, -r * ts) * NormCdf(d2));
}
```

```
static float[] BlackScholes(float[] ss, float[] xs, float[] ts)
{
    float r = 1.3f;
    float v = 2.5f;
    var result = new float[ss.GetLength(0)];
    for (int i = 0; i < ss.GetLength(0); i++)
    {
        result[i] = BlackCholes1(ss[i], xs[i], ts[i], r, v);
    }
    return result;
}
```

```
static FloatParallelArray BlackScholes(FloatParallelArray ss,
                                       FloatParallelArray xs,
                                       FloatParallelArray ts)
{
    float r = 1.3f;
    float v = 2.5f;
    return BlackCholes1(ss, xs, ts, r, v);
}
```



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



```
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;
        }
```

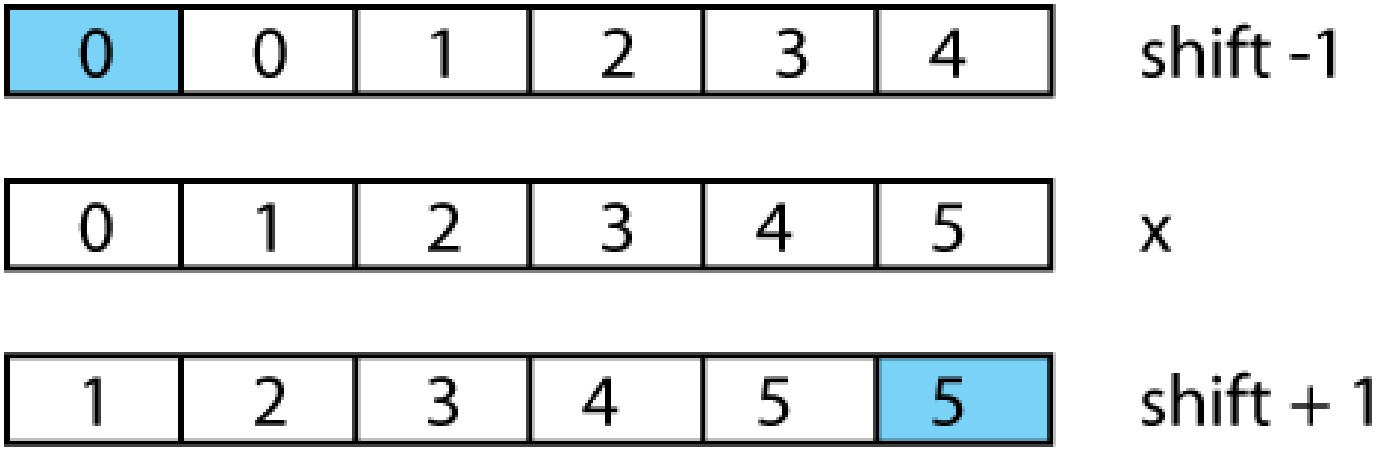
$$y_t = \sum_{k=0}^{N-1} a_k x_{t-k}$$

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

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

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

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]



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]]

y = a[0] * shift (x, 0) +
a[1] * shift (x, -1) +
a[2] * shift (x, -2) +
a[3] * shift (x, -3) +
a[4] * shift (x, -4)

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

Microsoft Research

```
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 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

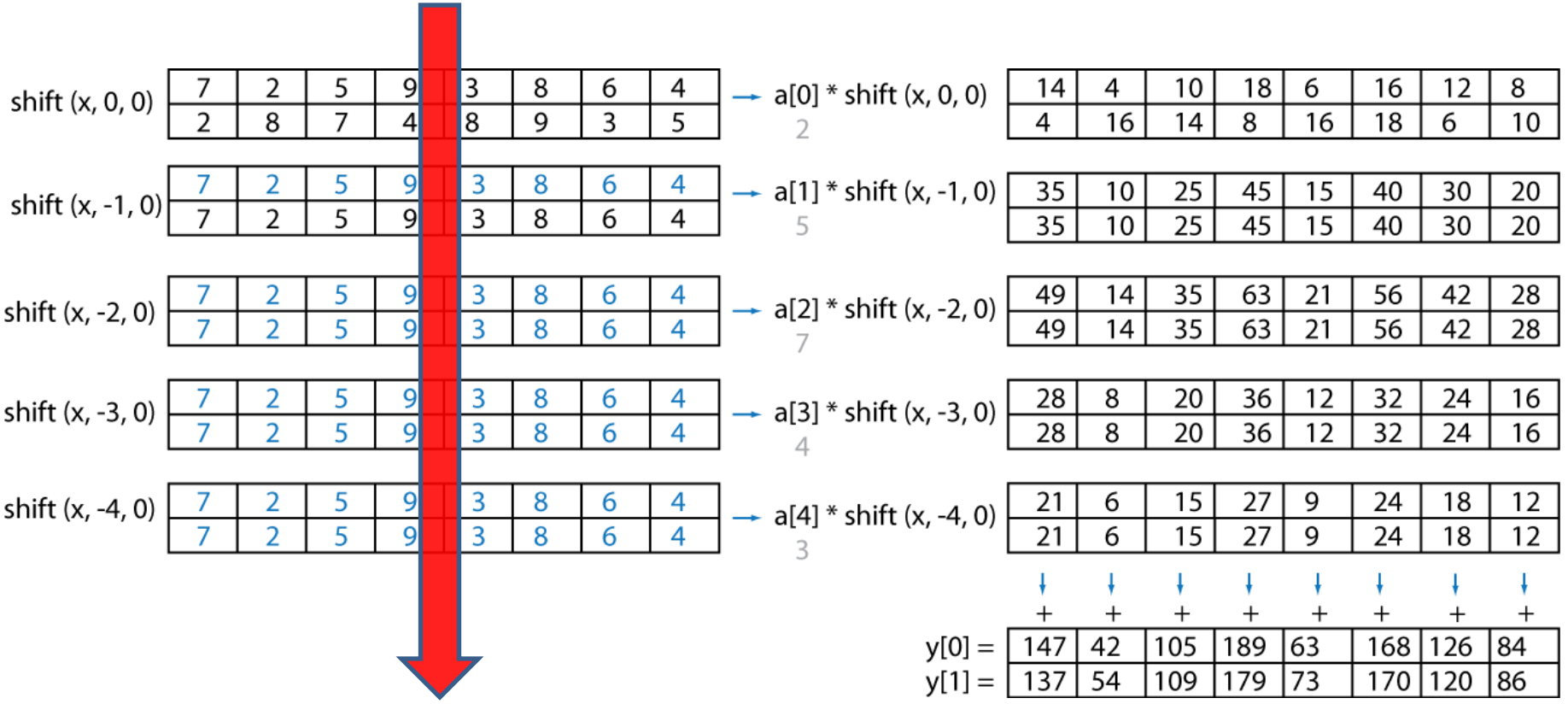
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

\+ \+ \+ \+ \+ \+ \+ \+

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

Microsoft Research

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

```
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);
}
```

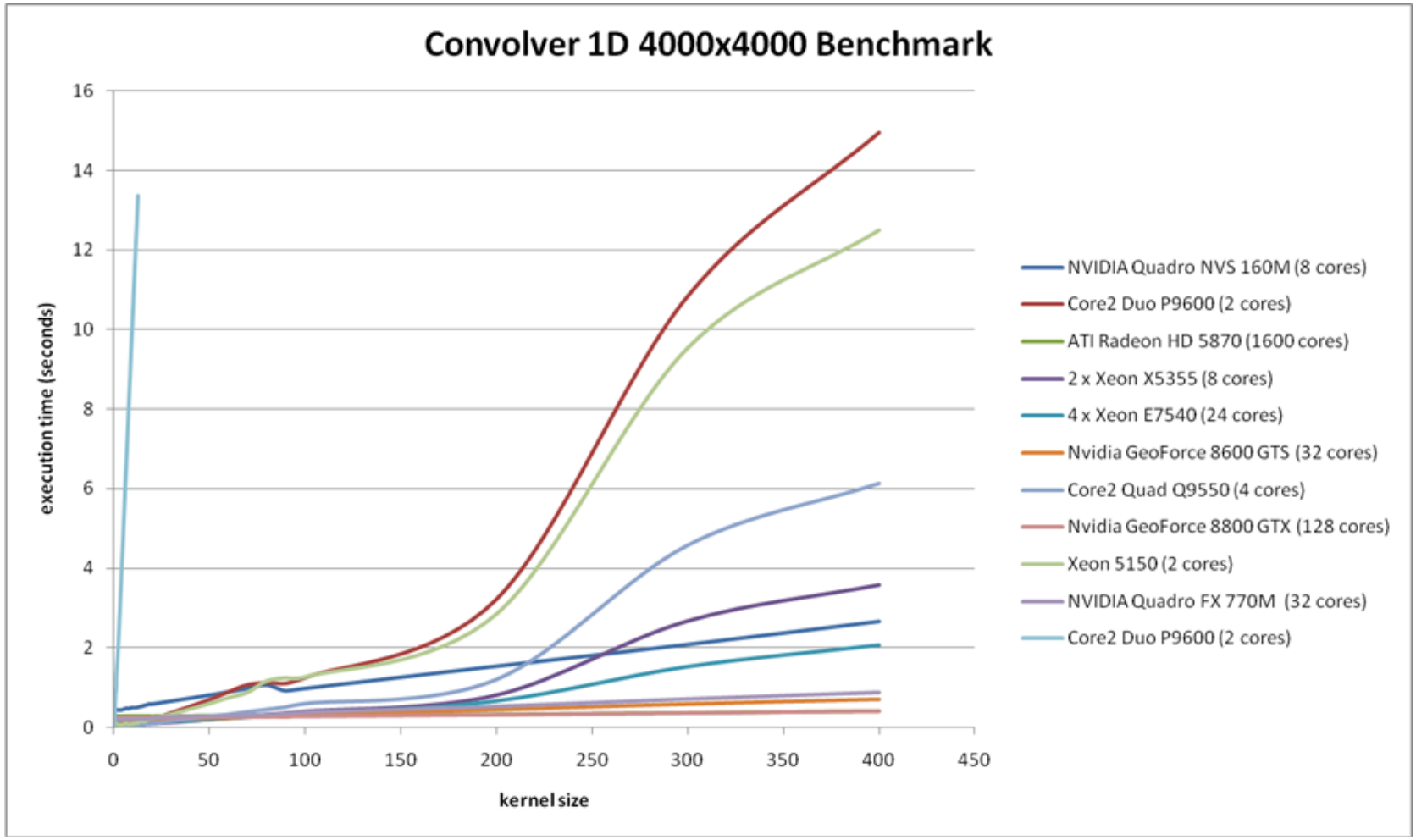
```
            for (int col = 0; col < inputSize; col++)
                Console.Write("{0} ", result[row, col]);
            Console.WriteLine();
        }
    }
}
```

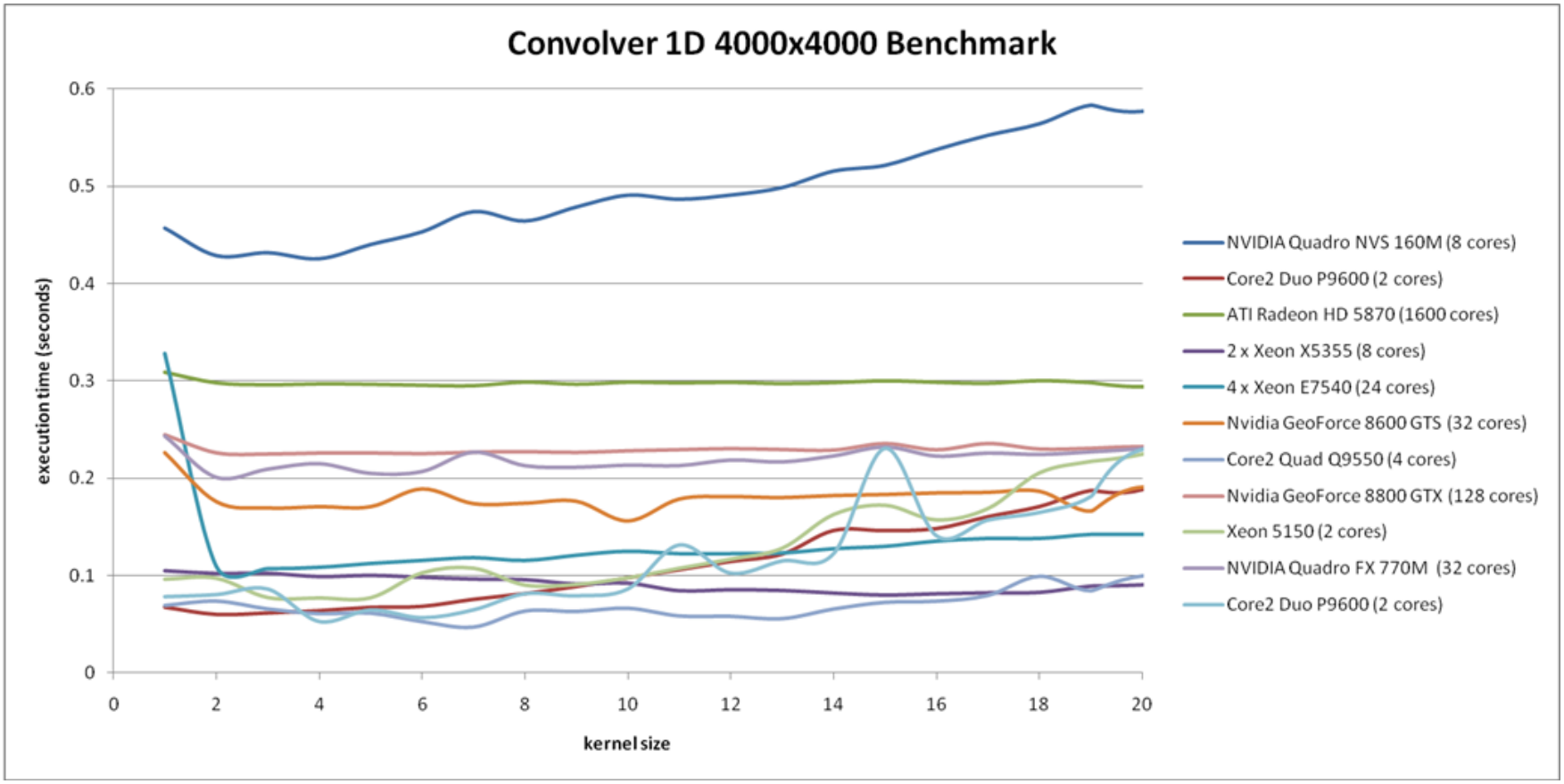
```
FPA ConvolveXY(Target &tgt, int height, int width, int filterSize, float filter[], FPA input, float *resultArray)
{
    // Convolve in X (row) direction.
    size_t dims[] = {height,width};
    FPA smoothX = FPA(0,dims, 2);
    intptr_t counts[] = {0,0};
    int filterHalf = filterSize/2;
    float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

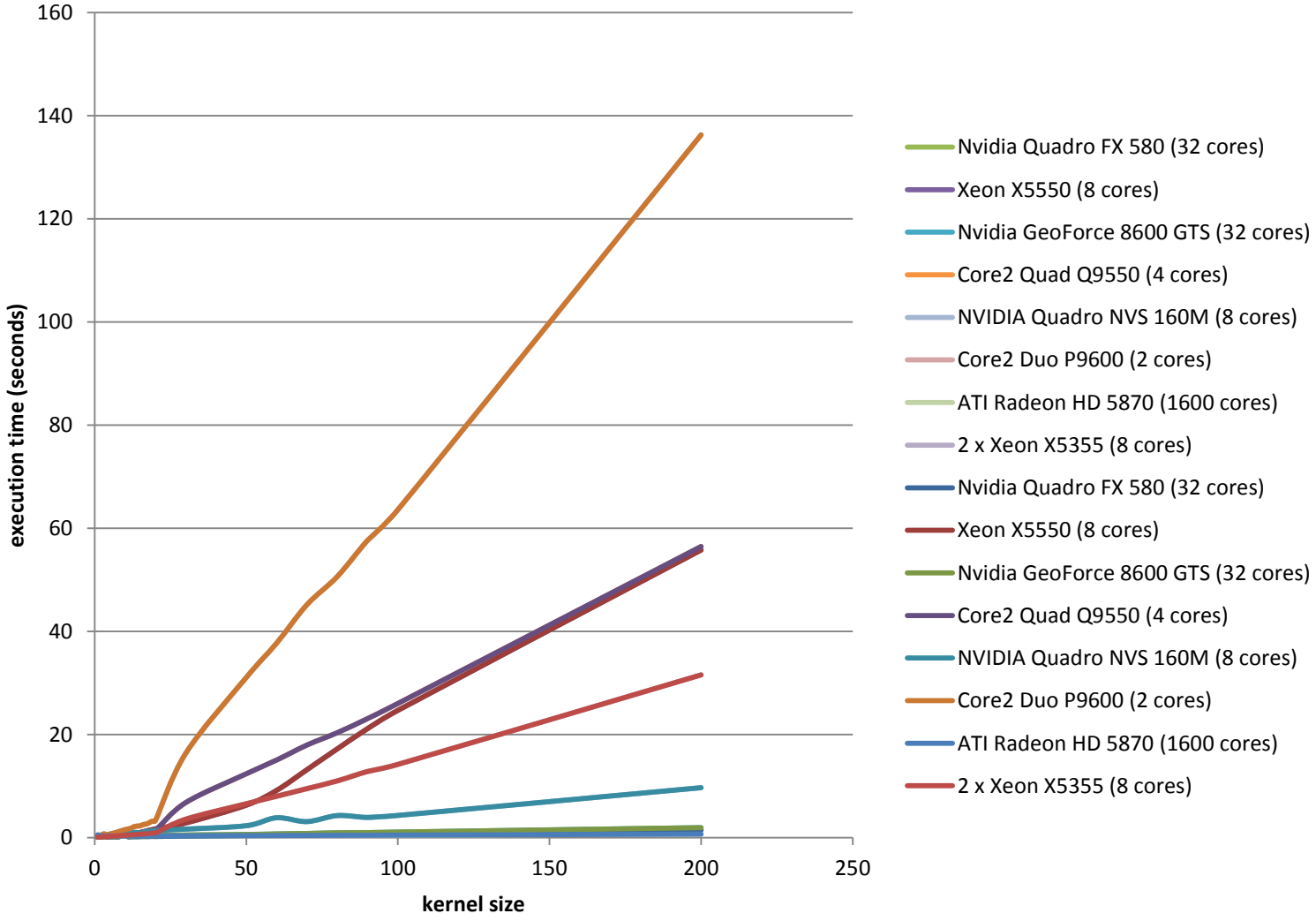
```
let convolveXY kernel input
  = // First convolve in the X direction and then in Y
    let convolveX = convolve (fun i -> [| -i; 0 |]) kernel
                             (kernel.Length - 1) input
    let convolveY = convolve (fun i -> [| 0; -i |]) kernel
                             (kernel.Length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

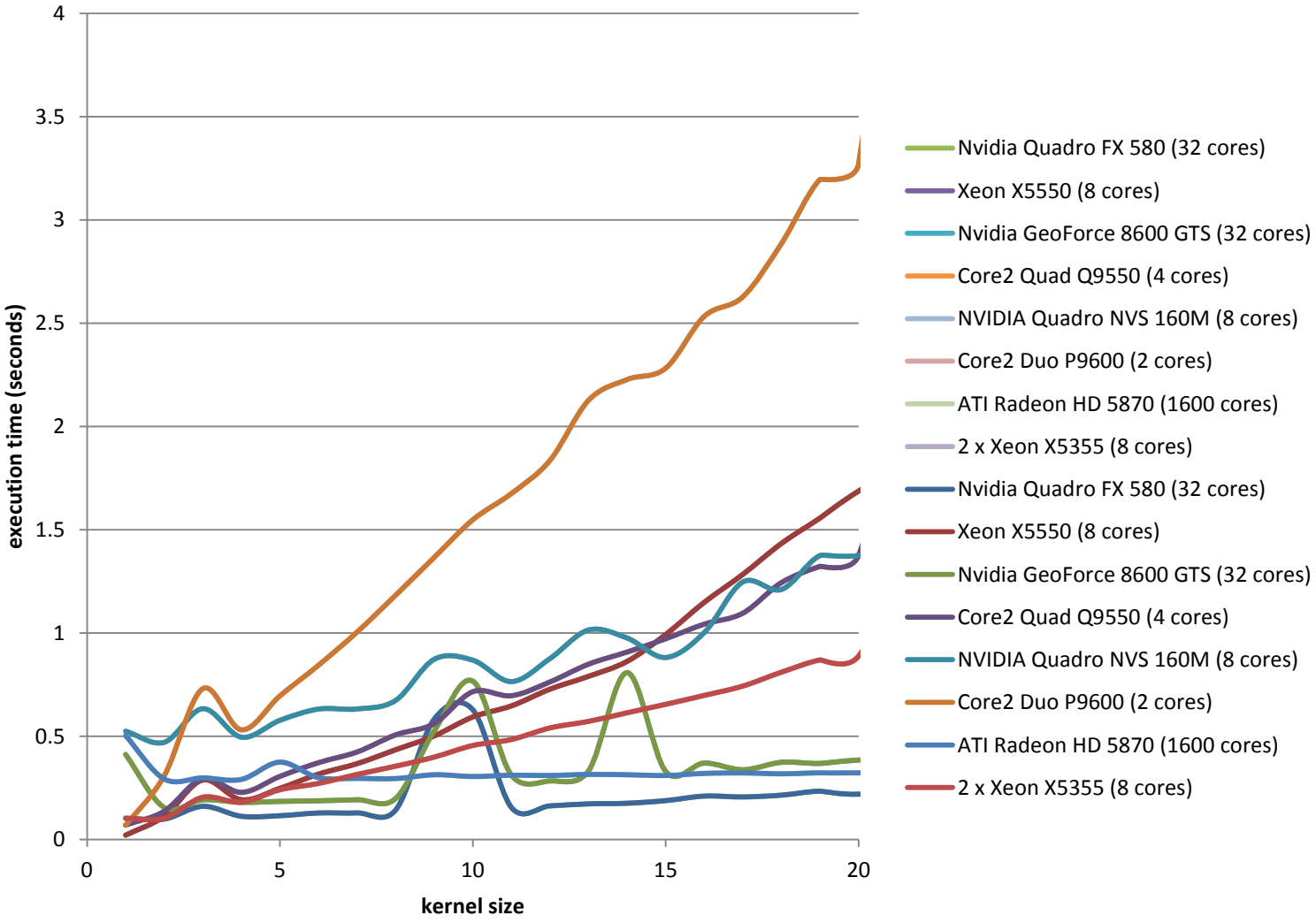

Convolver 1D 4000x4000 Benchmark
execution time (seconds)
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
14
16
18
20
kernel size
NVIDIA Quadro NVS 160M (8 cores)
Core2 Duo P9600 (2 cores)
ATI Radeon HD 5870 (1600 cores)
2 x Xeon X5355 (8 cores)
4 x Xeon E7540 (24 cores)
Nvidia GeoForce 8600 GTS (32 cores)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

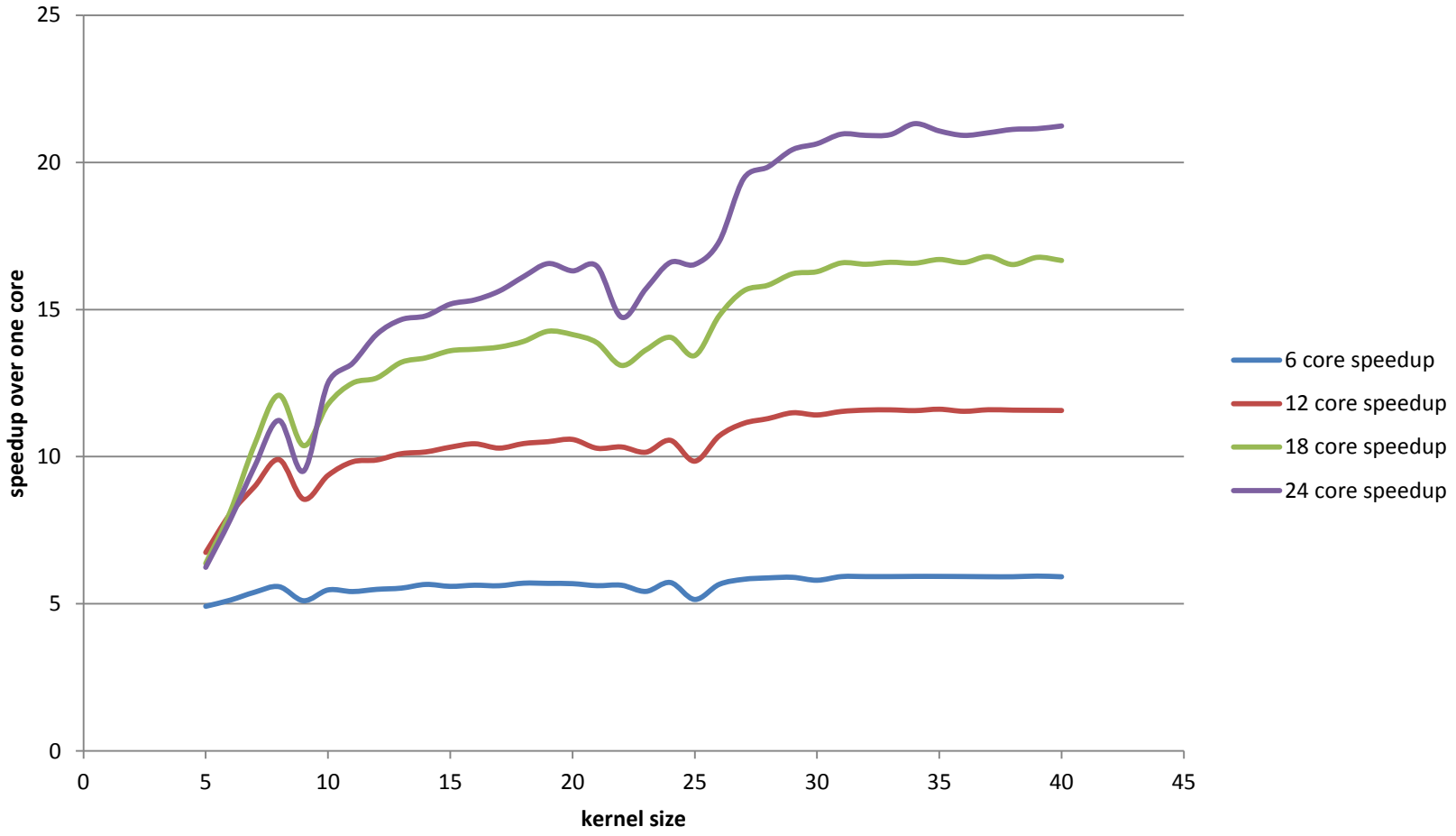


Microsoft Research

Convolver 2D 4000x4000 Benchmark



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




XILINX®
XCV300ES
BG432
DE4187
9839
COMPAQ

Microsoft Research

UK

DRIVING LICENCE

1. SINGH
2. DR SATNAM

3. [redacted] INDIA

4a. 24-01-08 4b. 26-02-12 4c. DVLA

5. [redacted] 12

7. Satnam Singh

8. 22 STOREYS WAY, CAMBRIDGE, CB3 0DT

S99GA

9. B,BE,C1,C1E,D1,D1E,*f,k,l,n,p*


BEE3
FPGA_B
FPGA_C
15 3:43PM

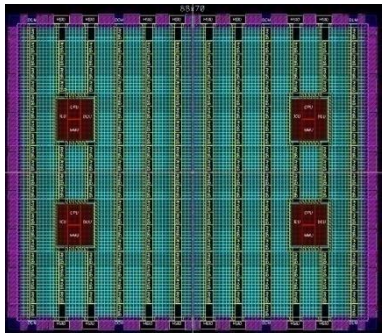
Microsoft Research

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

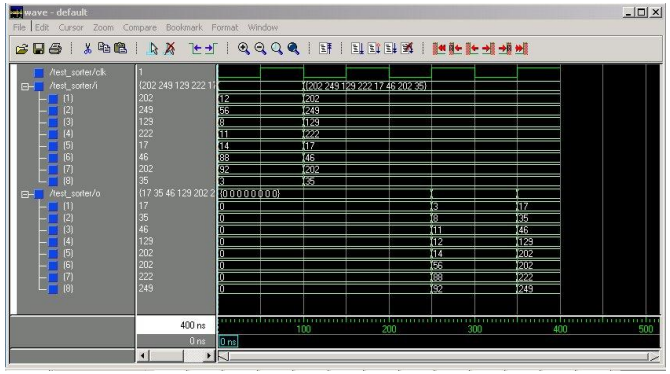


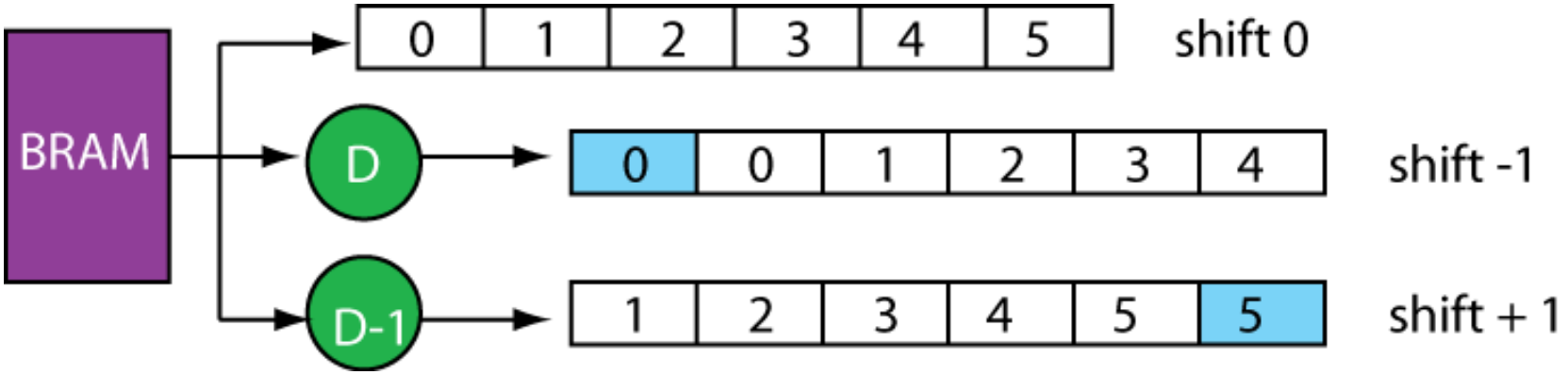
opportunity

scientific computing
data mining
search
image processing
financial analytics

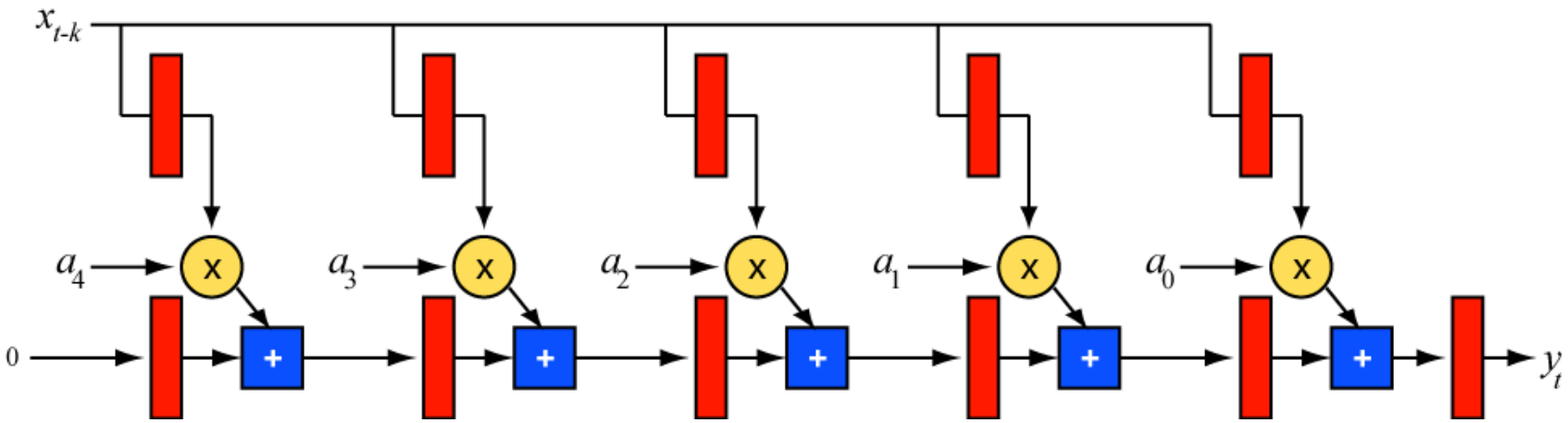
Verilog

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

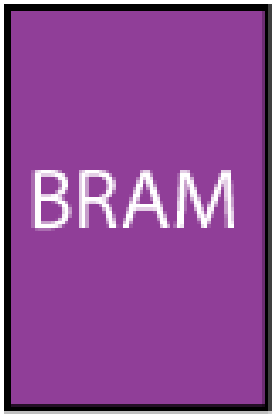
32-bit integer input data
32-bit integer coefficients
3 taps

Virtex-5 FPGA

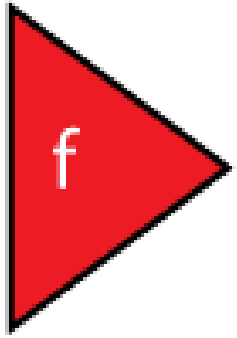
XC5VLX50T-2

175 MHz

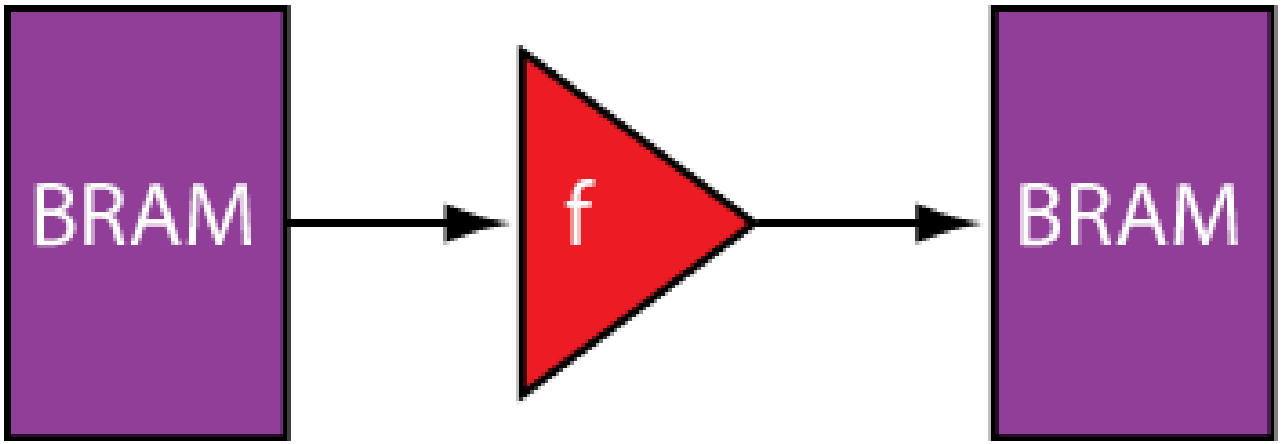
BRAM to BRAM

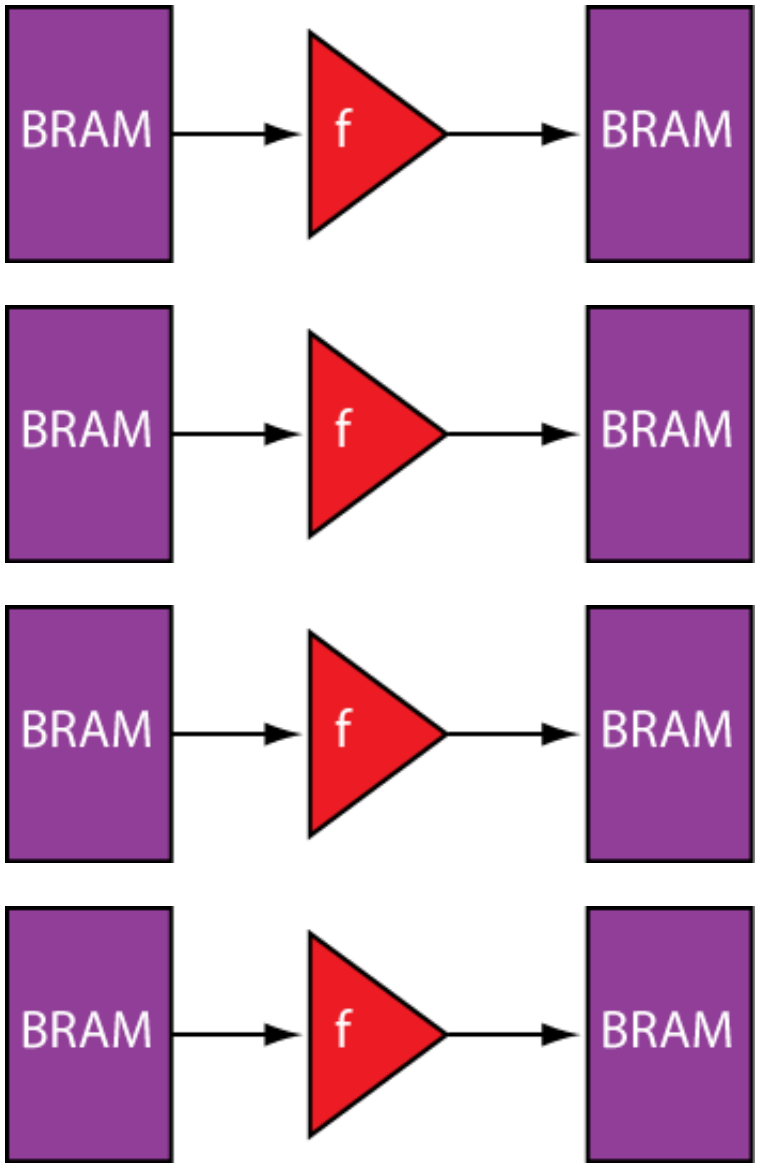


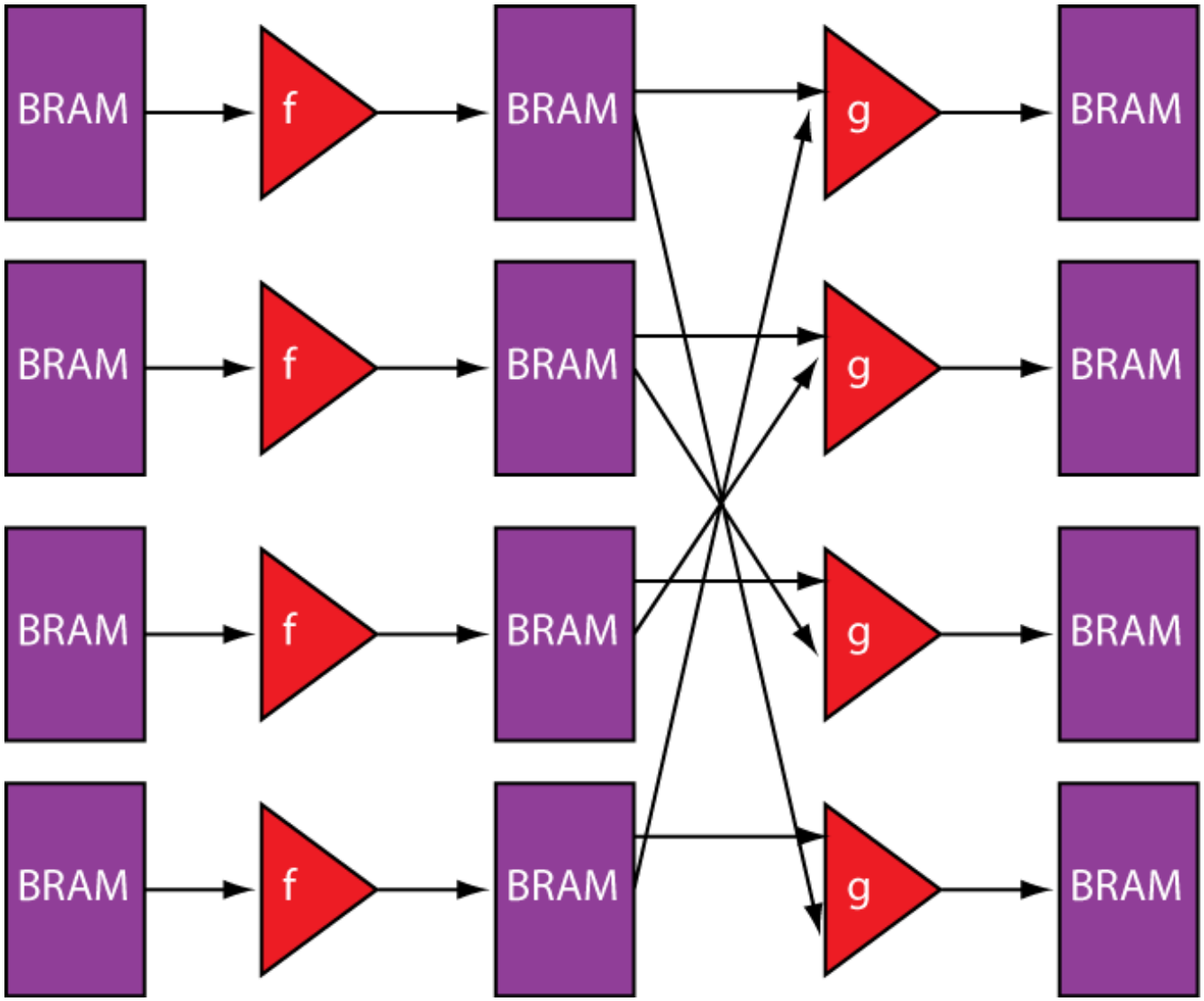
36Kbits
38,304
dual-ported



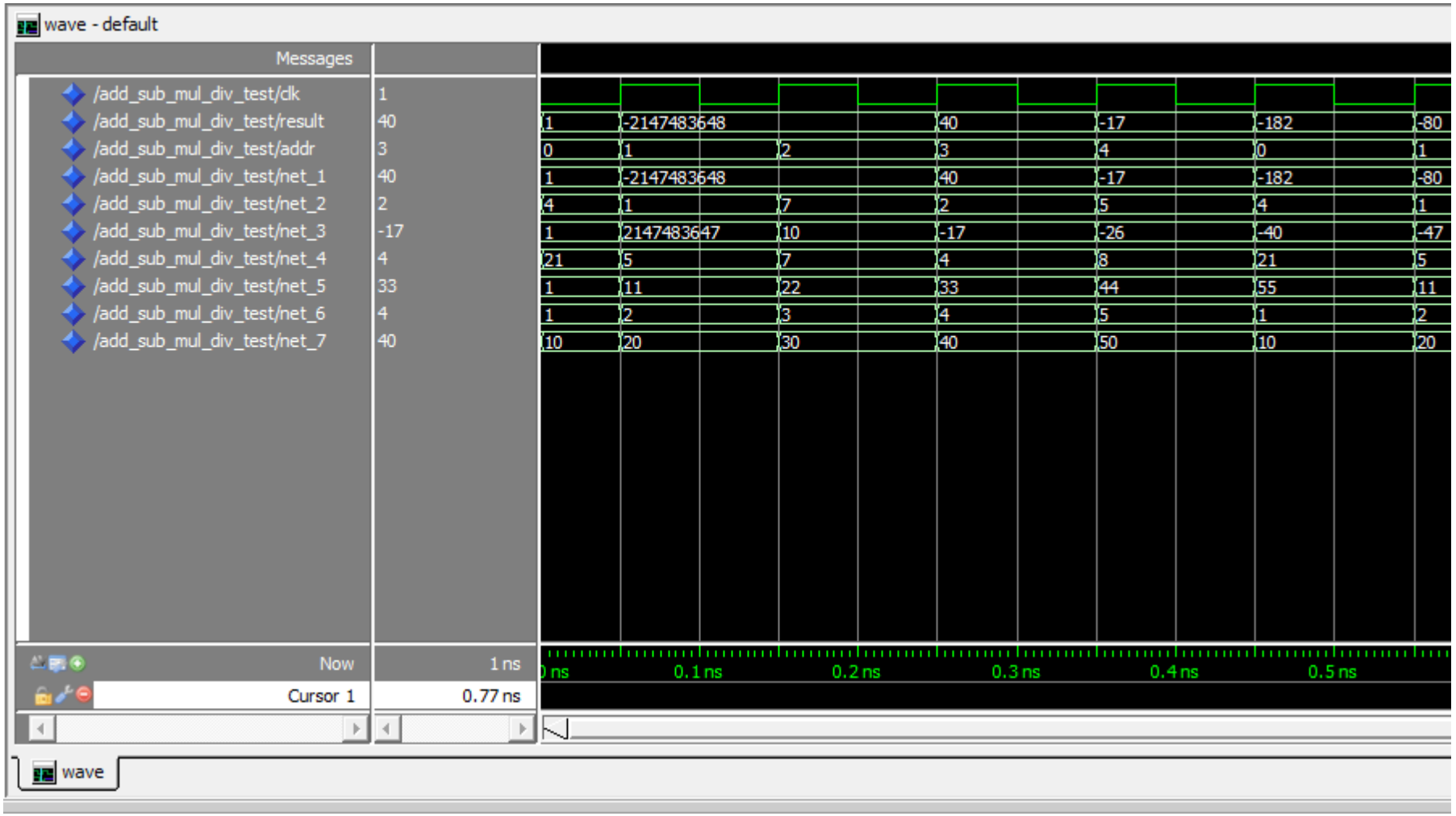
FPGA basic logic (LUTS)
DSP blocks (current max 2016)

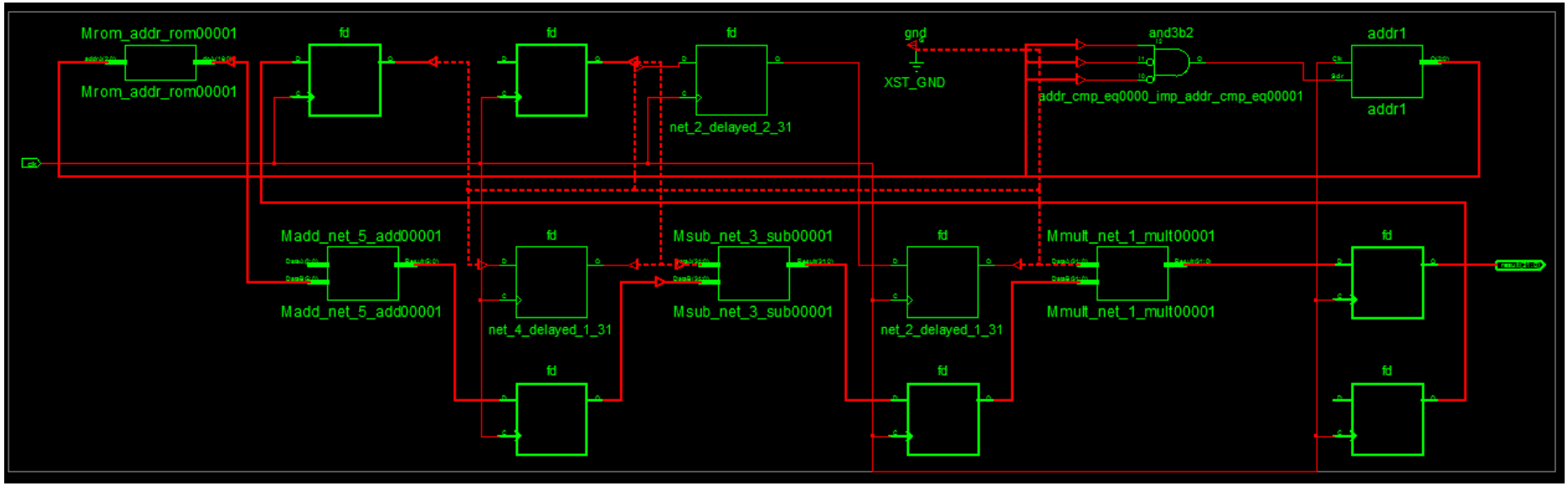

BRAM
f
BRAM


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

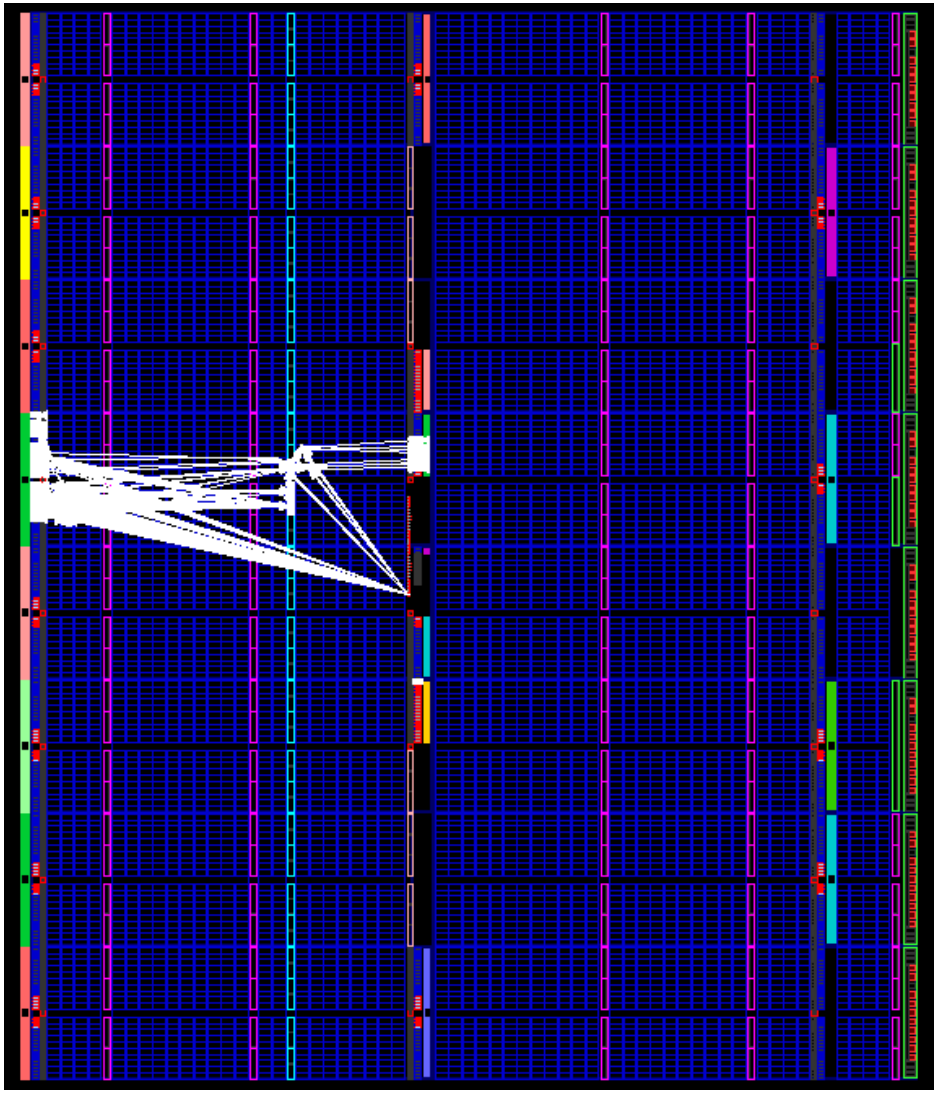

BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM
BRAM
f
BRAM
g
BRAM

```
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 ;
};
```

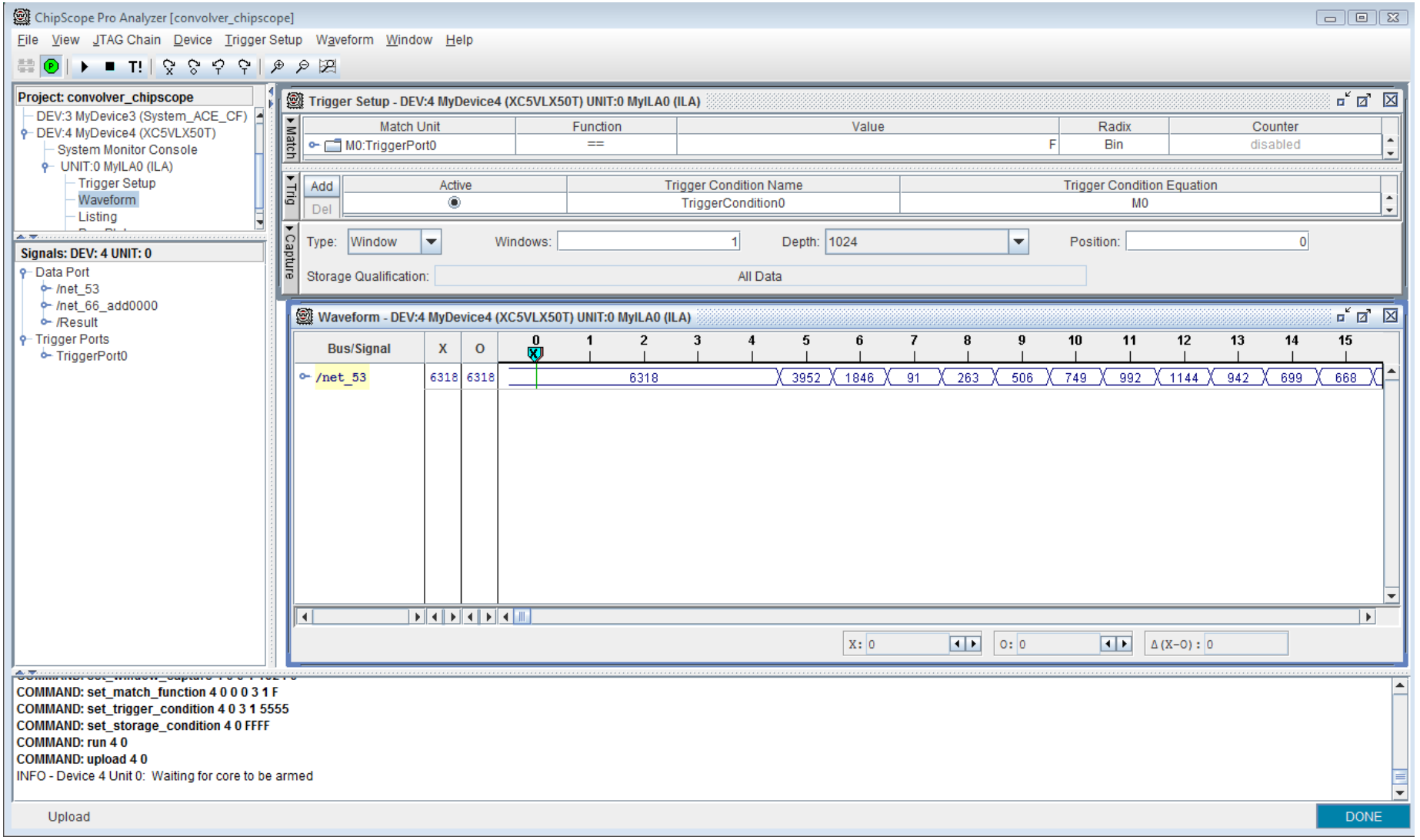

wave - default
Messages
/add_sub_mul_div_test/clk
/add_sub_mul_div_test/result
/add_sub_mul_div_test/addr
/add_sub_mul_div_test/net_1
/add_sub_mul_div_test/net_2
/add_sub_mul_div_test/net_3
/add_sub_mul_div_test/net_4
/add_sub_mul_div_test/net_5
/add_sub_mul_div_test/net_6
/add_sub_mul_div_test/net_7
1
40
3
40
2
-17
4
33
4
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
0.1 ns
0.2 ns
0.3 ns
0.4 ns
0.5 ns
wave



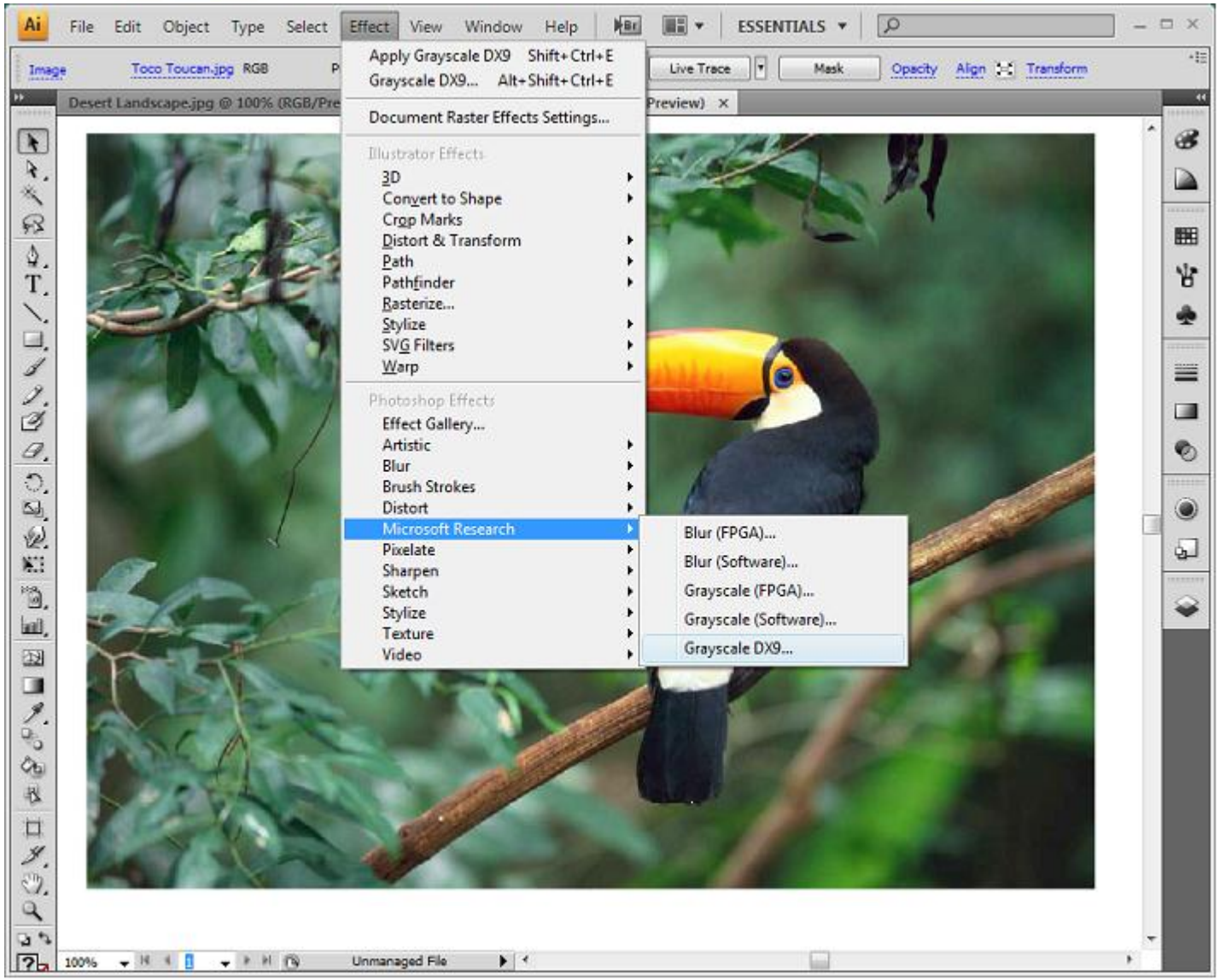
8.249ns max delay
3 x DSP48Es
63 slice registers
24 slice LUTs



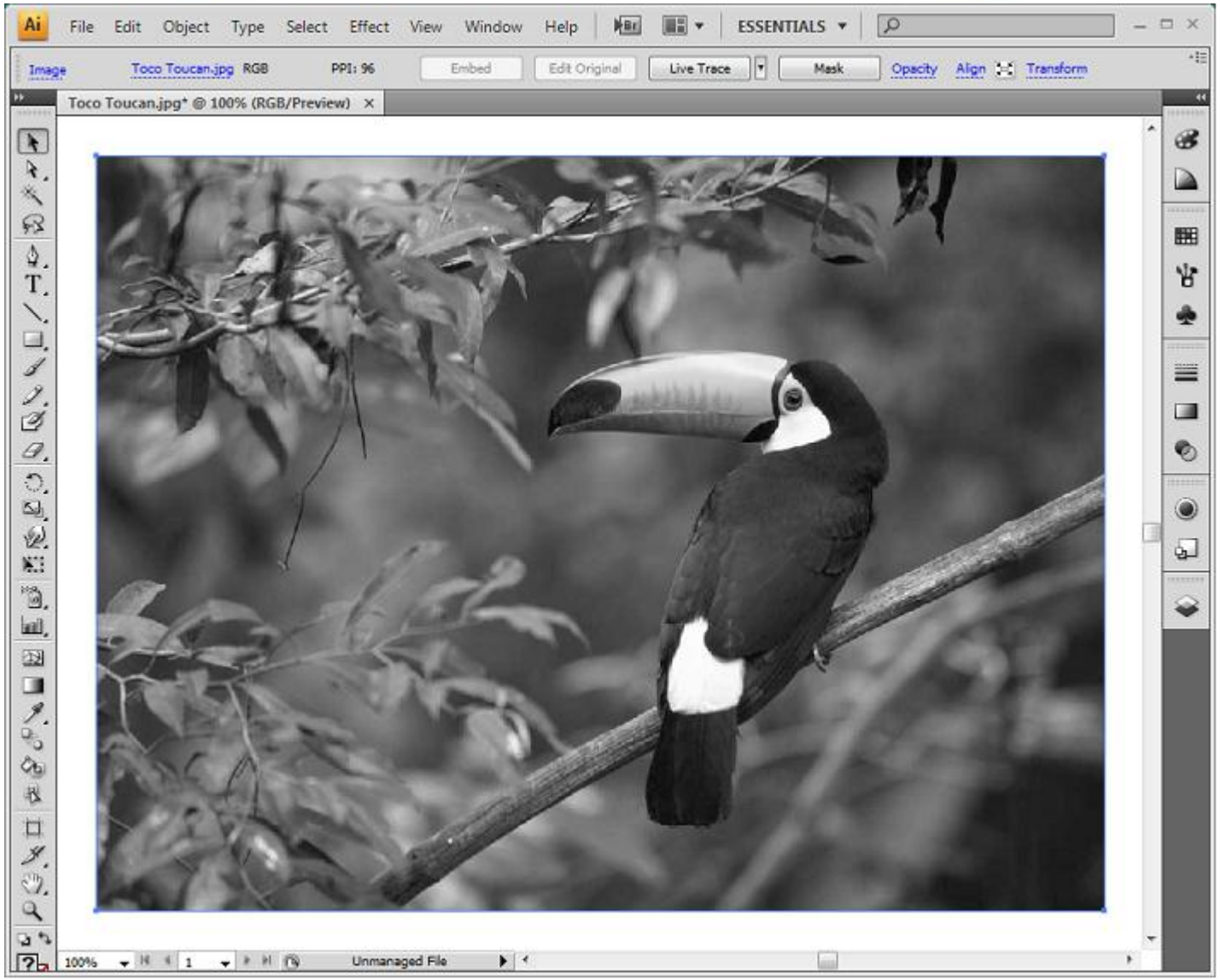
Microsoft Research



Microsoft Research

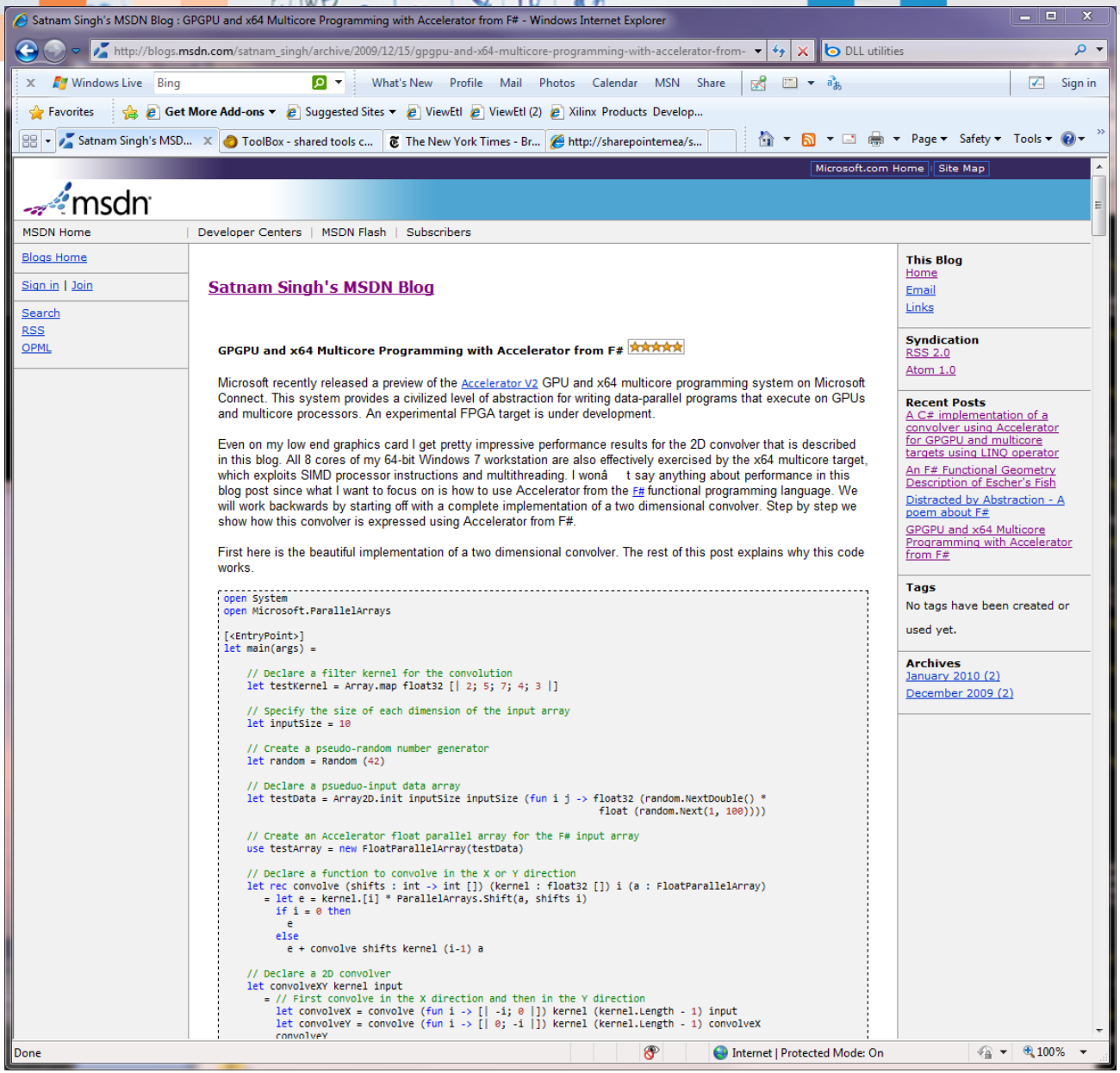


Microsoft Research



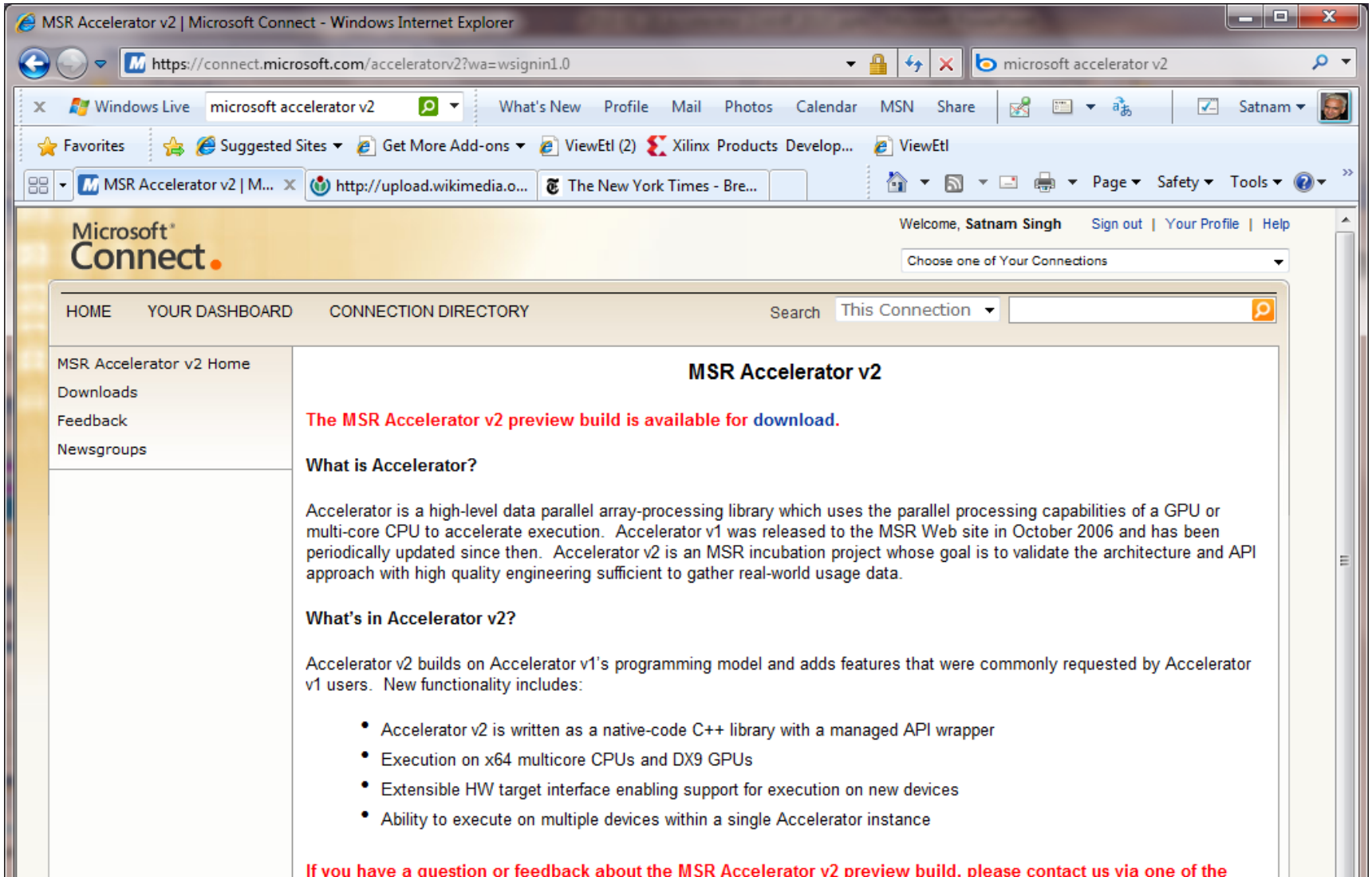
```
// Compute grayscale
Target &tgt = CreateDX9Target();
float* grayF = (float*) malloc(sizeof(float) * pixels) ;
FPA red   = FPA(redF,   rectHeight, rectWidth) ;
FPA green = FPA(greenF, rectHeight, rectWidth);
FPA blue  = FPA(blueF,  rectHeight, rectWidth);
FPA sum = Add (77 * red, Add (151 * green, 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);
  }
```

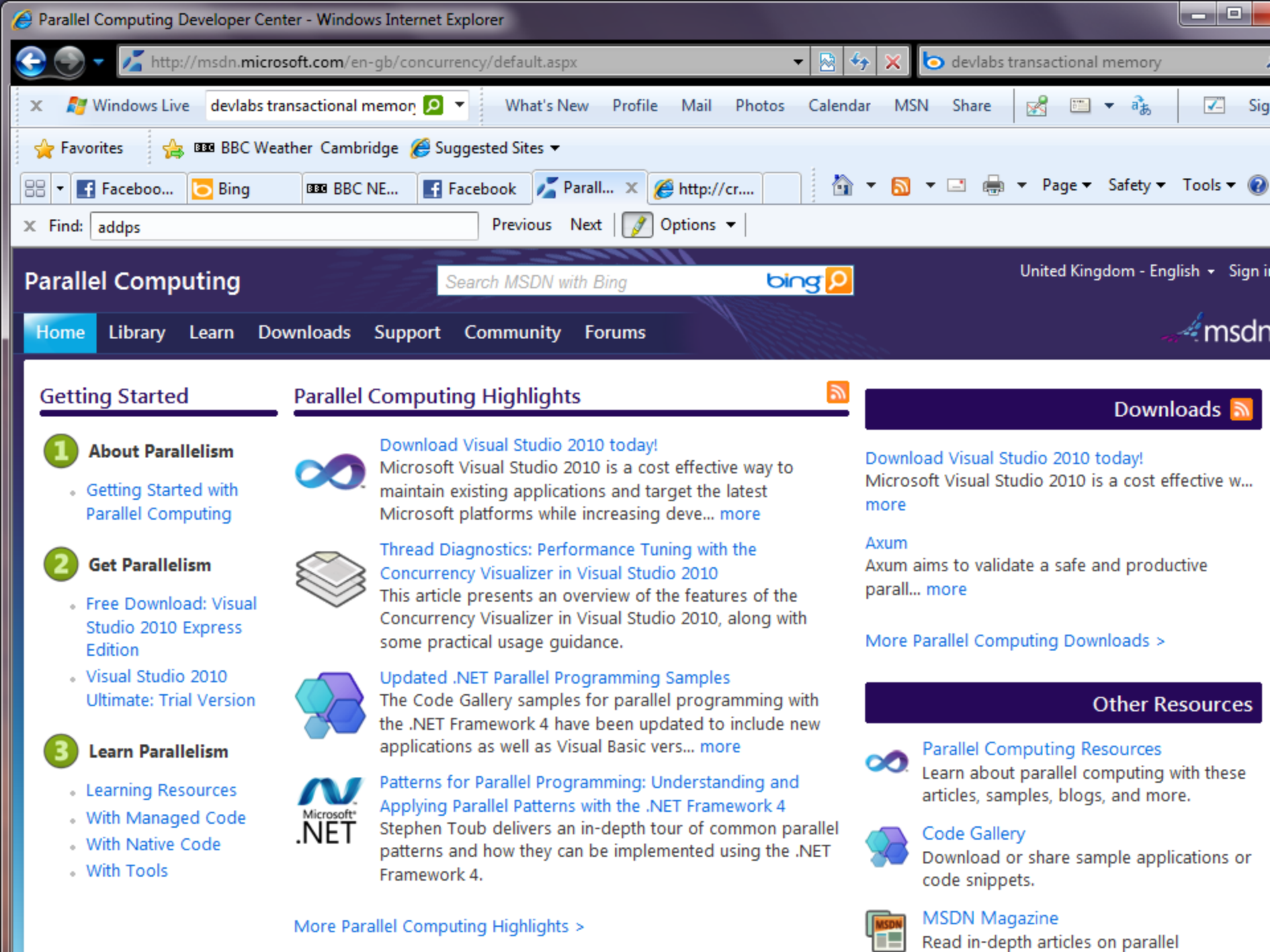
Microsoft Research



Microsoft Research

Search for “Microsoft Accelerator V2”




Parallel Computing Developer Center - Windows Internet Explorer
http://msdn.microsoft.com/en-gb/concurrency/default.aspx
devlabs transactional memory
Windows Live
devlabs transactional memor
What's New
Profile
Mail
Photos
Calendar
MSN
Share
Favorites
BBC Weather Cambridge
Suggested Sites
Faceboo...
Bing
BBC NE...
Facebook
Parall...
http://cr....
Page
Safety
Tools
Find:
addps
Previous
Next
Options
Parallel Computing
Search MSDN with Bing
bing
United Kingdom - English
Sign i
Home
Library
Learn
Downloads
Support
Community
Forums
msdn
Getting Started
1 About Parallelism
Getting Started with Parallel Computing
2 Get Parallelism
Free Download: Visual Studio 2010 Express Edition
Visual Studio 2010 Ultimate: Trial Version
3 Learn Parallelism
Learning Resources
With Managed Code
With Native Code
With Tools
Parallel Computing Highlights
Download Visual Studio 2010 today!
Microsoft Visual Studio 2010 is a cost effective way to maintain existing applications and target the latest Microsoft platforms while increasing deve... more
Thread Diagnostics: Performance Tuning with the Concurrency Visualizer in Visual Studio 2010
This article presents an overview of the features of the Concurrency Visualizer in Visual Studio 2010, along with some practical usage guidance.
Updated .NET Parallel Programming Samples
The Code Gallery samples for parallel programming with the .NET Framework 4 have been updated to include new applications as well as Visual Basic vers... more
Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4
Stephen Toub delivers an in-depth tour of common parallel patterns and how they can be implemented using the .NET Framework 4.
Microsoft .NET
More Parallel Computing Highlights >
Downloads
Download Visual Studio 2010 today!
Microsoft Visual Studio 2010 is a cost effective w... more
Axum
Axum aims to validate a safe and productive parall... more
More Parallel Computing Downloads >
Other Resources
Parallel Computing Resources
Learn about parallel computing with these articles, samples, blogs, and more.
Code Gallery
Download or share sample applications or code snippets.
MSDN Magazine
Read in-depth articles on parallel

Project Tuva Enhanced Video Player
Watch the Feynman Lectures

> People > Satnam Singh

Satnam Singh

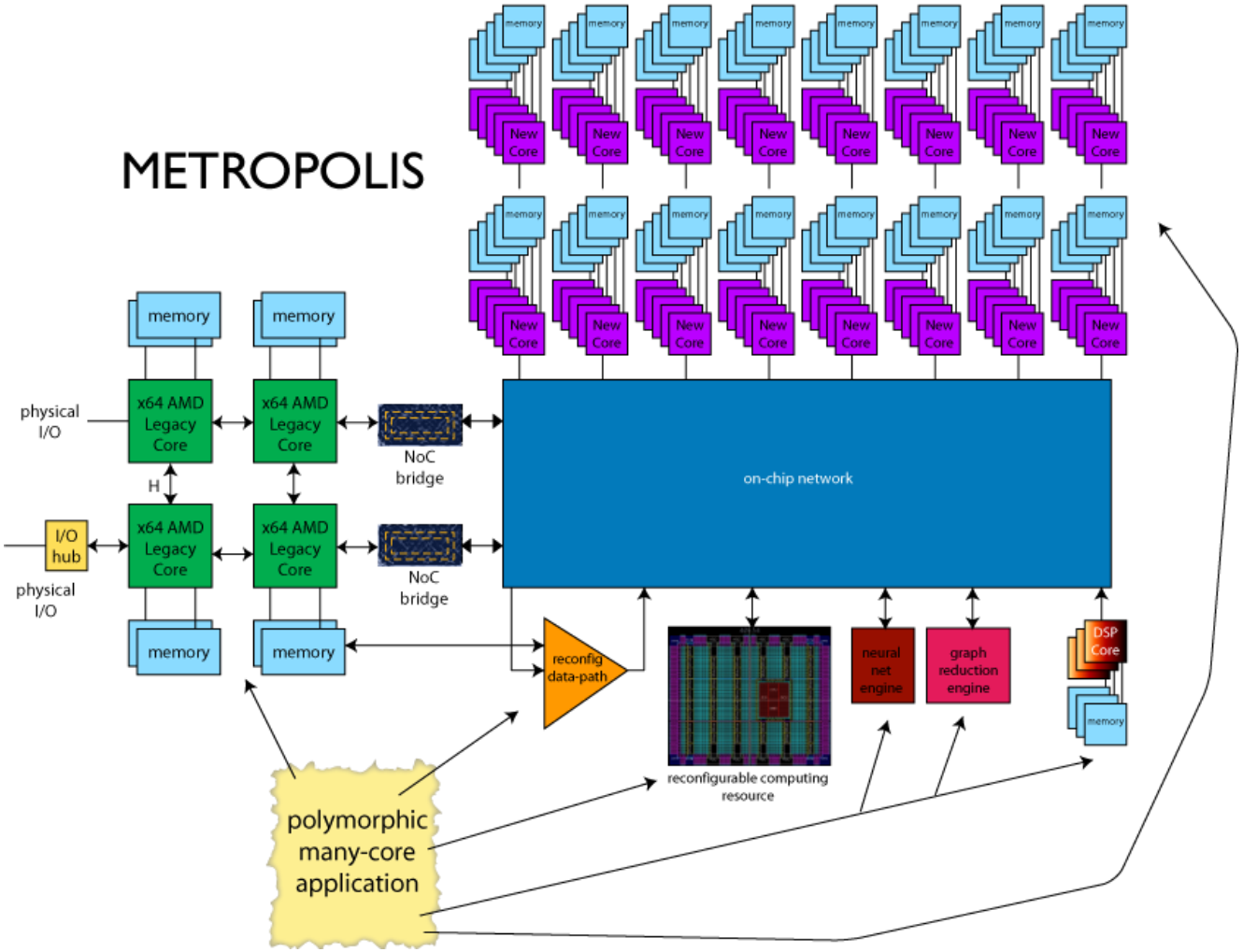
SENIOR RESEARCHER

I currently work on two research topics:

- **Alchemy: Transumtation of Programs into Circuits.**
  - The compilation of data-parallel programs written in C++ and .NET langauges like C# into FPGA circuits.
  - The Kiwi project with David Greaves which synthesizes circuits from high level circuit models written using regular multi-threaded code in languages like C#.
  - Synthesis of C programs that manipulate dynamic data structurs in the heap.
  - Synthesis of data-parallel programs in C++ and C# written with using the Microsoft Accelerator V2 library into FPGAs for co-processing.
  - High level techniques for designing low level circuits. I have re-implemented my Lava system in C# and F# and I have made HLINQ which is a circuit generator for LINQ queries.
  - Evaluation and experimentation with alternative hardware description

Some Recent Talks

- C-to-Gates Stnthesis of Dynamic Data Structures. Practical Synthesis for Concurrent Systems (PSY) 2009, Grenoble (CAV workshop). 28 June 2009.
- Three Approaches to High Level Synthesis. IBM Zurich Research Lab. 17 June 2009.
- GPU Programming with Microsoft Accelerator. Oxford University. 5 June 2009.
- Pointer Synthesis++. UK Design Forum '09. 8 May 2009.
- An Overview of Parallel and Concurrent Programming in Haskell. Unviersity of Leeds. 30 April 2009.
- Kiwi: Synthesis of FPGA Cicuits from


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