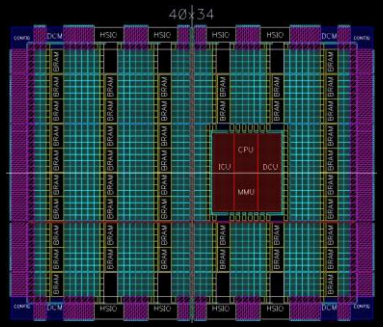
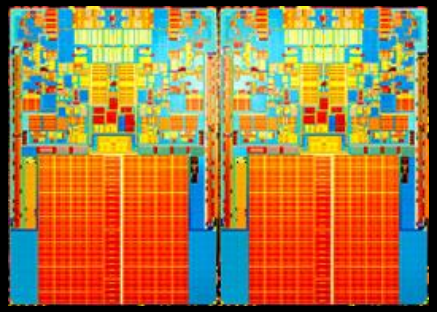
GPU TECHNOLOGY CONFERENCE

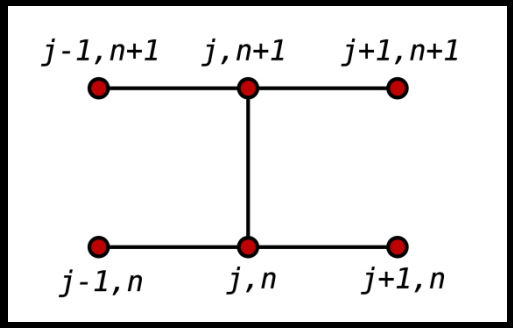
UNIVERSITY OF BIRMINGHAM

Heterogeneous Data-Parallel Programming

Satnam Singh, The University of Birmingham, UK







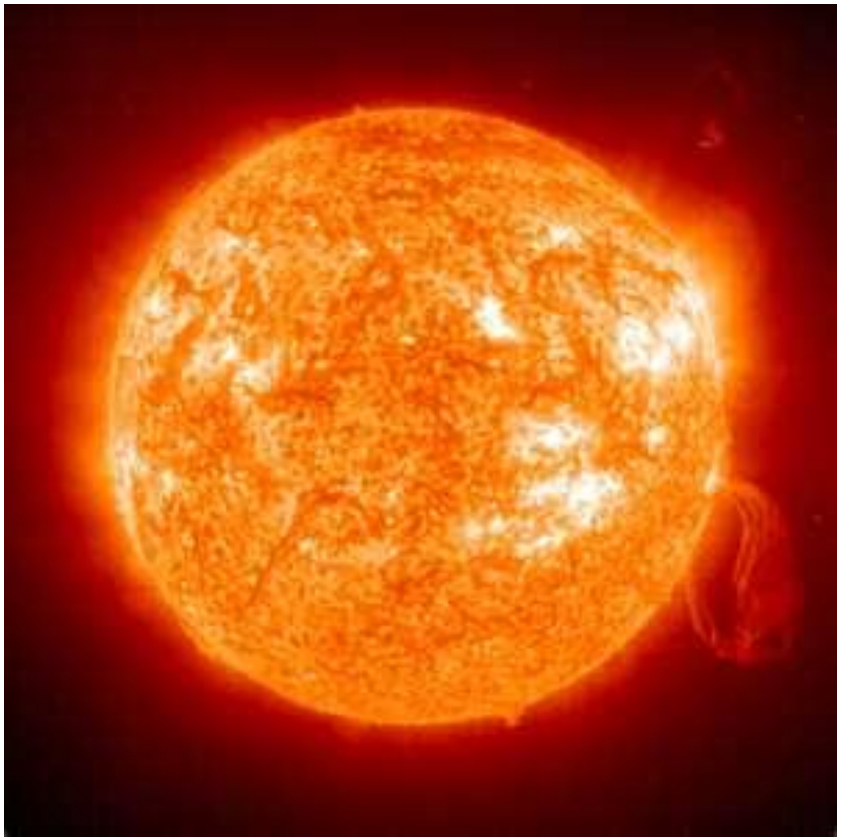
UNIVERSITY OF BIRMINGHAM

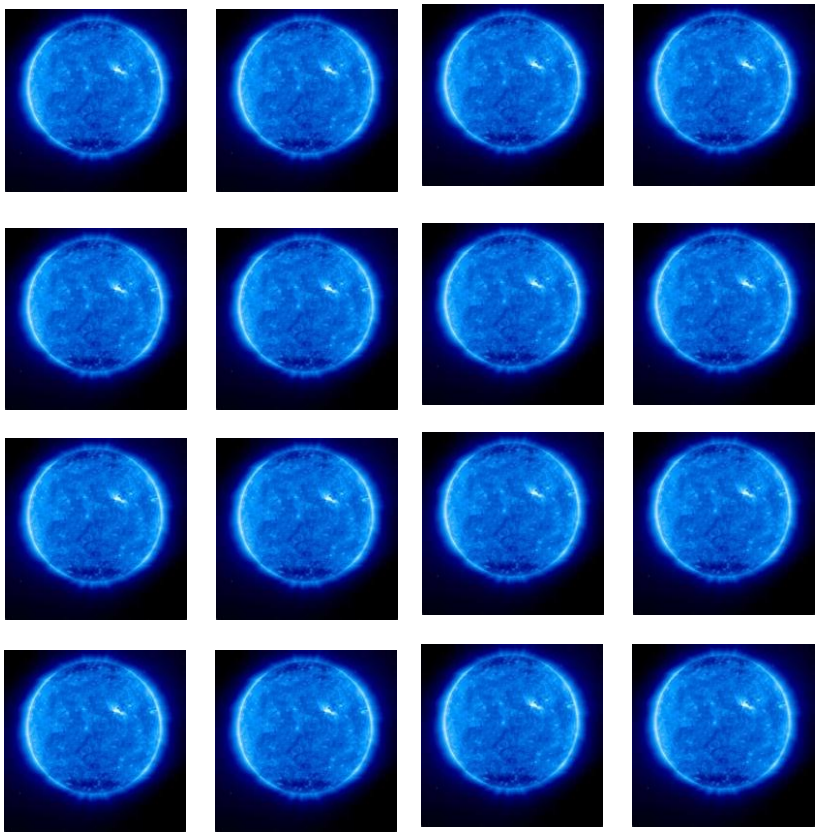
Professor Satnam Singh
Chair of Reconfigurable Systems
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham, B15 2TT
United Kingdom.

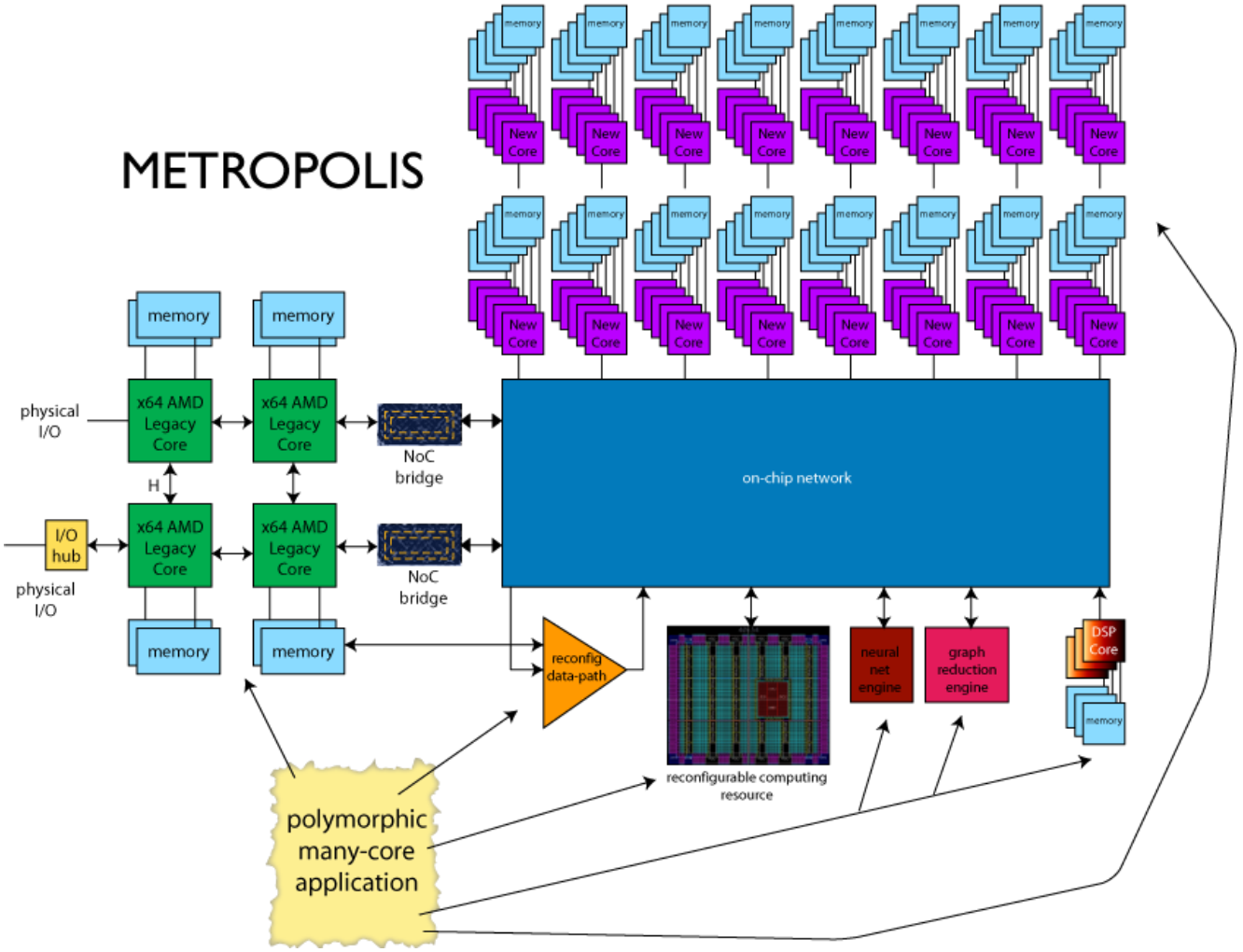
Email: singhsu@cs.bham.ac.uk or s.singh@acm.org
UK cell: +44 7979 648412.
USA cell: +1 408 656 4590

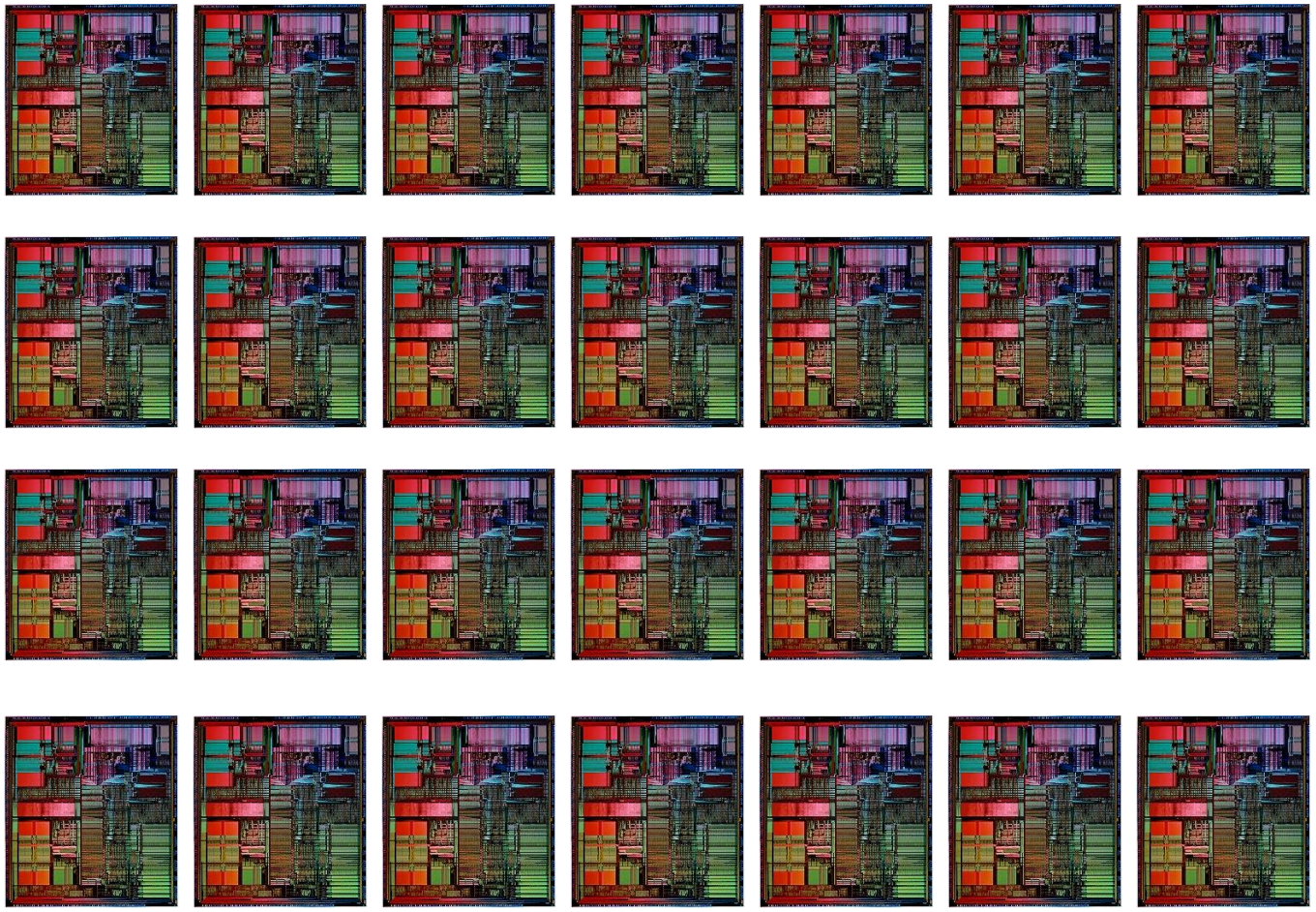
I hold the **Chair of Reconfigurable Systems** at the University of Birmingham and in January 2012 I will join Google in Mountain View, California. I currently work on two research topics:

- **Alchemy: Transmuting Programs into Circuits** for computing with reduced latency or energy consumption.
  - 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++, C# and F# 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 techniques with a focus on Bluespec and Esterel.
- **Concurrent and Parallel Programming in Haskell** for exploiting the potential of pure functional programming for exploiting parallel computing resources in a civilized manner.
  - Specifically, I have been working on the ThreadScope profiler to help understand the behavior of parallel and concurrent Haskell programs.
  - Higher level data-parallel programming models based on Accelerator.
  - The compilation of recursive Haskell funcitons into circuits.

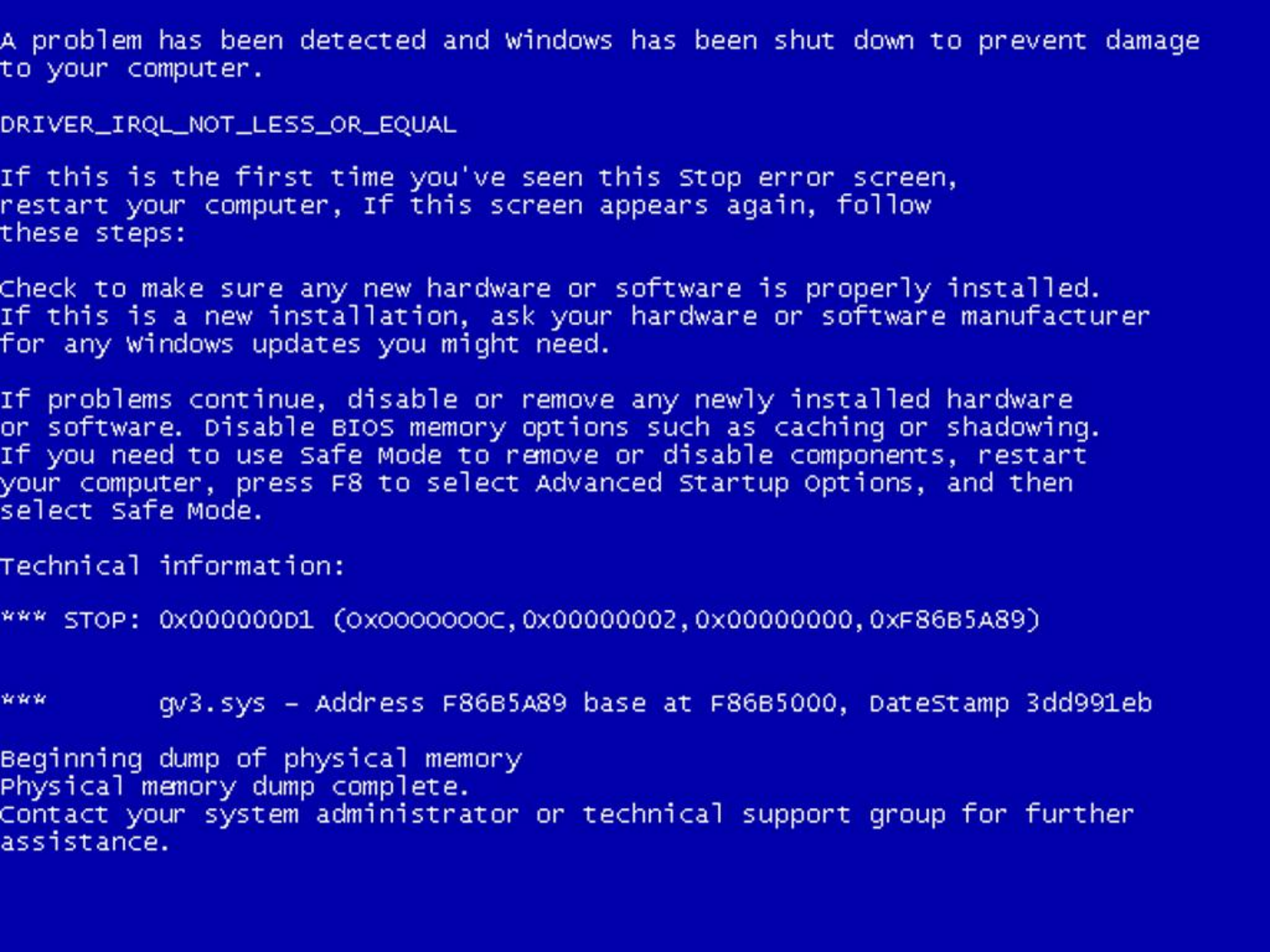





METROPOLIS
memory
New Core
physical I/O
x64 AMD Legacy Core
NoC bridge
H
I/O hub
on-chip network
reconfig data-path
neural net engine
graph reduction engine
DSP Core
reconfigurable computing resource
polymorphic many-core application



locks
monitors
condition variables
spin locks
priority inversion


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
If this is the first time you've seen this Stop error screen,
restart your computer, If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)
*** gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

News & Events

- What's New?
- Media Coverage
- Upcoming Events
- Newsletters
- AWS Blog

Related Resources

- What is AWS?
- AWS Products & Services
- AWS Solutions
- Contact Us
- Careers at AWS

What's New?

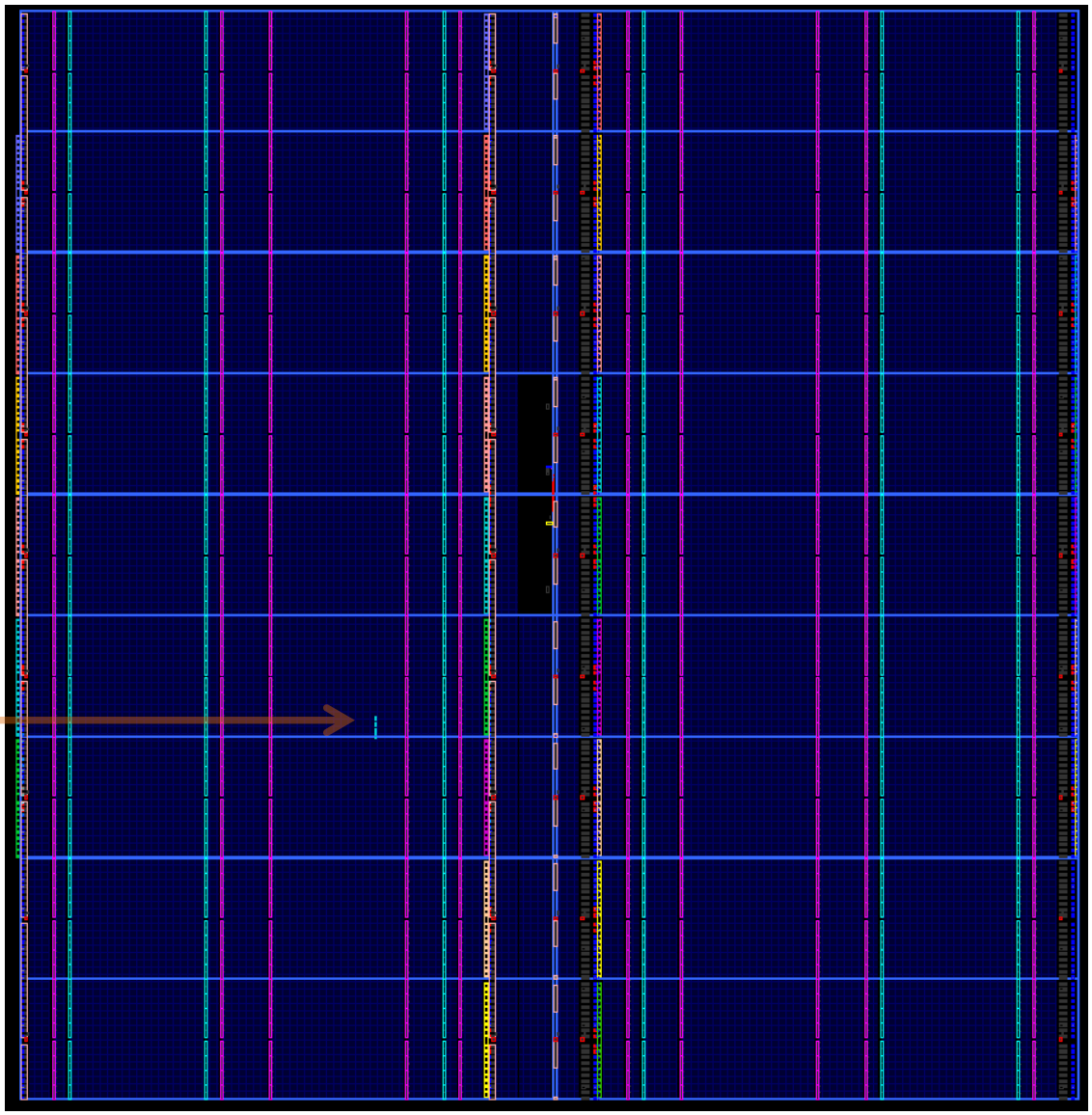
Announcing Cluster GPU Instances for Amazon EC2

We are excited to announce the immediate availability of Cluster GPU Instances for Amazon EC2, a new instance type designed to deliver the power of GPU processing in the cloud. GPUs are increasingly being used to accelerate the performance of many general purpose computing problems. However, for many organizations, GPU processing has been out of reach due to the unique infrastructural challenges and high cost of the technology. Amazon Cluster GPU Instances remove this barrier by providing developers and businesses immediate access to the highly tuned compute performance of GPUs with no upfront investment or long-term commitment.

Learn more about the new Cluster GPU instances for Amazon EC2 and their use in running HPC applications.


NVIDIA

14820 sim-adds
1,037,400,000,000
additions/second



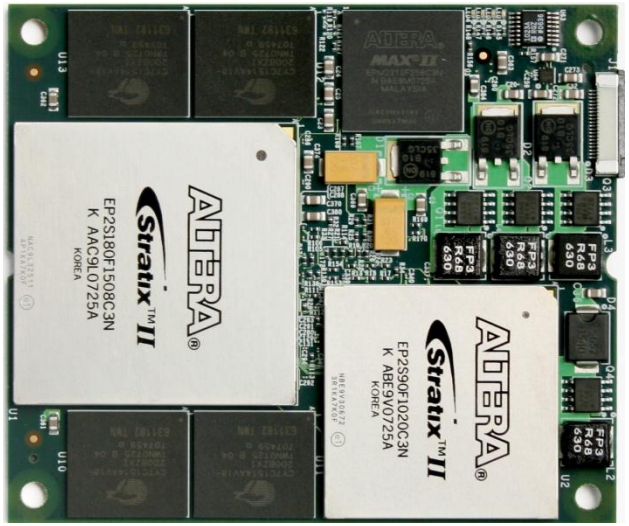
32-bit
integer
Adder
(32/474,240)
>700MHz

332x1440

XC6VLX760 758,784 logic cells, 864 DSP blocks,
1,440 dual ported 18Kb RAMs


XILINX®
XCV300ES
BG432
DE4187
9839

XD2000i FPGA in-socket accelerator for Intel FSB



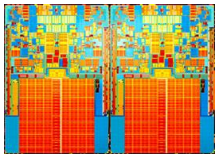
XD2000F FPGA in-socket accelerator for AMD socket F

XD1000 FPGA co-processor module for socket 940

processing with specialized processors
and heterogeneous systems

information
processing gap

Volume Of Information



processing with multi-core processors

time

2011


12
11

CUDAC Programmers

Java/Python/C#/F#/Perl Programmer

OpenMP

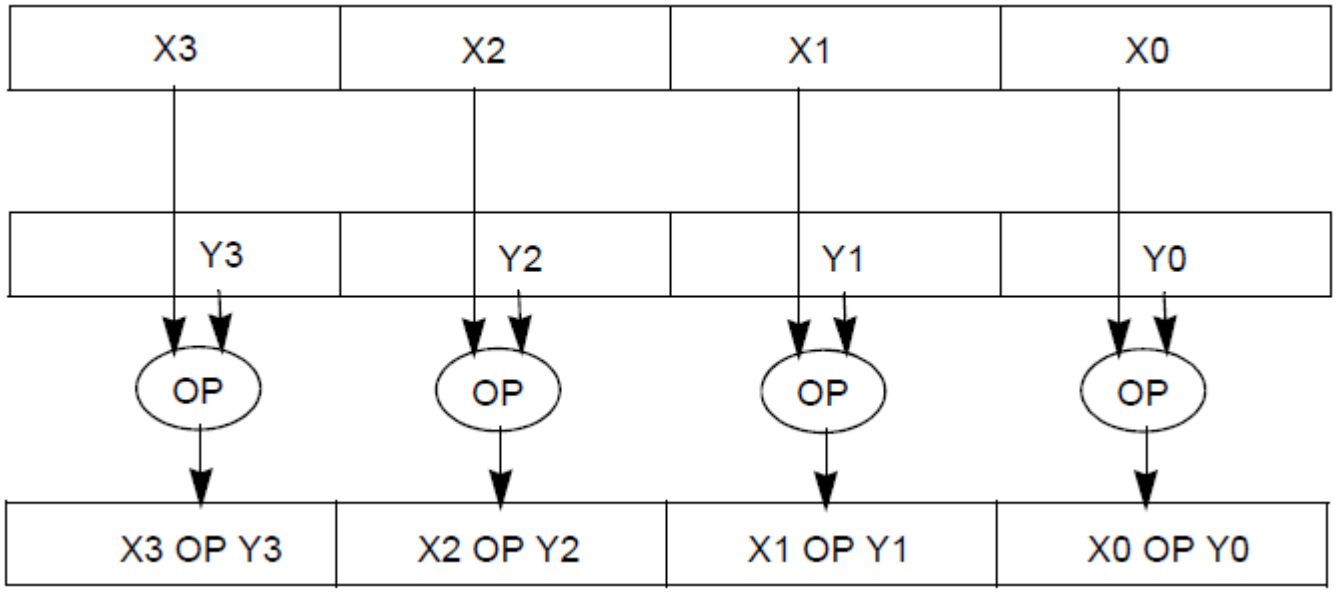
```
#include <stdio.h>
int main(int argc, char* argv[])
{
      const unsigned int n = 5000000 ;
      float *a = new float[n];
      float *b = new float[n];
      float *c = new float[n];
      int i, j ;
      #pragma omp parallel for
      for (i=0; i<n; i++)
         c[i] = a[i] + b[i] ;
      return 0;
}
```

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/



grand unification theory

polygots

Accelerator

Accelerator is a high-level data parallel library which uses parallel processors such as the GPU or multicore CPU to accelerate execution. Accelerator v1 was released to the MSR Web site in October 2006 and has been periodically updated since then. Accelerator v2 is an MSR incubation project whose goal is to to validate the architecture and API approach with high quality engineering sufficient to gather real-world usage data.

What's in Accelerator v2?

Accelerator v2 builds on Accelerator v1's programming model and adds features that were commonly requested by Accelerator v1 users. New functionality includes:

- Accelerator v2 is written as a native-code C++ library with a managed API wrapper
- Execution on multicore CPUs, both 32 and 64 bit, in addition to DX9 GPUs and CUDA.
- Extensible HW target interface enabling support for execution on new devices
- Ability to execute on multiple devices within a single Accelerator instance
- Asynchronous evaluation of parallel arrays
- Reusable expression graphs: Across different devices and on the same device with different leaf-node data

Download the Accelerator v2 Preview today to try it out. The package includes the Accelerator SDK, extensive documentation and several sample applications to help you get started.

Microsoft Redmond Accelerator Team

Barry Bond
Kerry Hammil
Lubomir Litchev
<anonymous other person>

Effort vs. Reward (Productivity)

CUDAC
OpenCL
HLSL
DirectCompute

Thurst
Accelerator

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

CENTERS & PARTNERS

Academic Partners
CUDA Centers of Excellence
CUDA Fellows
CUDA Research Centers
CUDA Teaching Centers

PROGRAMS

Academic Partnership
CUDA Center of Excellence
CUDA Research Center
CUDA Teaching Center
Graduate Fellowship
Tegra Prototype Proposals

ADDITIONAL INFORMATION

CUDA Courses Around the World
CUDA Course Materials
CUDA Education and Training
CUDA Forums
CUDA Zone
Developer Zone
Intern & Coop Programs
Sign Up For Research News

EXTERNAL LINKS

Thrust: A Productivity-Oriented Library for CUDA

"Thrust: A Productivity-Oriented Library for CUDA"
Nathan Bell (NVIDIA), Jared Hoberock (NVIDIA), in *GPU Computing Gems, Jade Edition, Edited by Wen-mei W. Hwu*, October 2011

Research Area: Algorithms & Numerical Techniques
High Performance Computing
Tools & Libraries

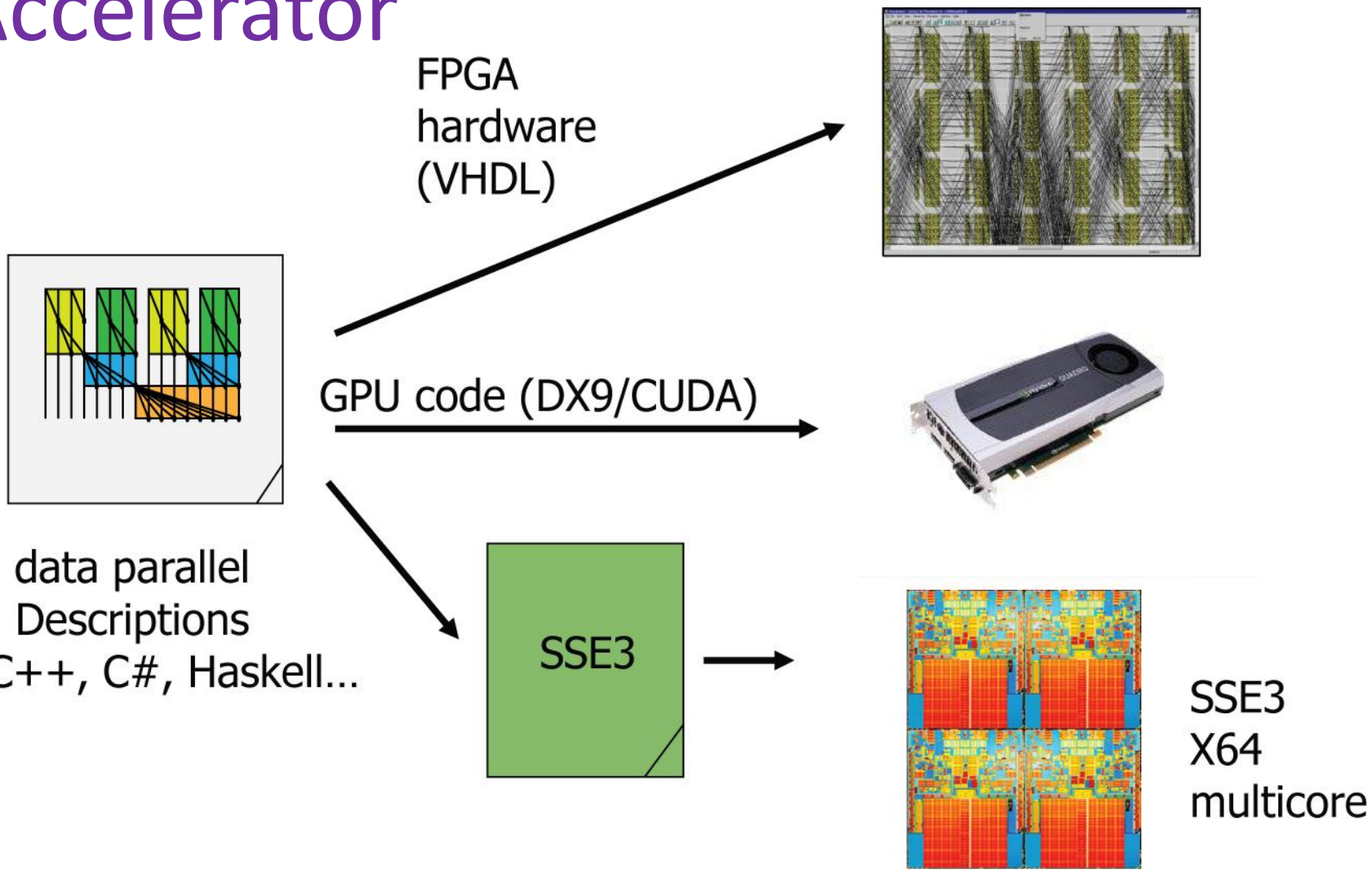
Author(s): Nathan Bell (NVIDIA), Jared Hoberock (NVIDIA)

Date: October 2011

Download(s): Thrust Website Paper (PDF)

Abstract: This chapter demonstrates how to leverage the Thrust parallel template library to implement high-performance applications with minimal programming effort. Based on the C++ Standard Template Library (STL), Thrust brings a familiar high-level interface to the realm of GPU Computing while remaining fully interoperable with

Accelerator




F2
F3
F4
F5
F6
F7
2
4
5
6
7
Q
E
R
Y
A
S
D
F
H


Kingston
KVR
KVR1066D3S7/4G
1.5V
ASSY IN KOREA
Warranty Void If Removed
GENUINE KINGSTON

DRAM

Addition	$R_{i,j} = A_{i,j} + B_{i,j}$
Multiplication	$R_{i,j} = A_{i,j} \times B_{i,j}$
Scalar Multiplication (k)	$R_{i,j} = kA_{i,j}$
Maximum	$R_{i,j} = \max(A_{i,j}, B_{i,j})$
Sine	$R_{i,j} = \sin A_{i,j}$
Square root	$R_{i,j} = \sqrt{A_{i,j}}$
And	$R_{i,j} = A_{i,j} \wedge B_{i,j}$
Equality test	$R_{i,j} = \begin{cases} \text{true} & \text{if } A_{i,j} = B_{i,j} \\ \text{false} & \text{otherwise} \end{cases}$
Greater than test	$R_{i,j} = A_{i,j} > B_{i,j}$
Select	$R_{i,j} = \begin{cases} B_{i,j} & \text{if } A_{i,j} = \text{true} \\ C_{i,j} & \text{otherwise} \end{cases}$

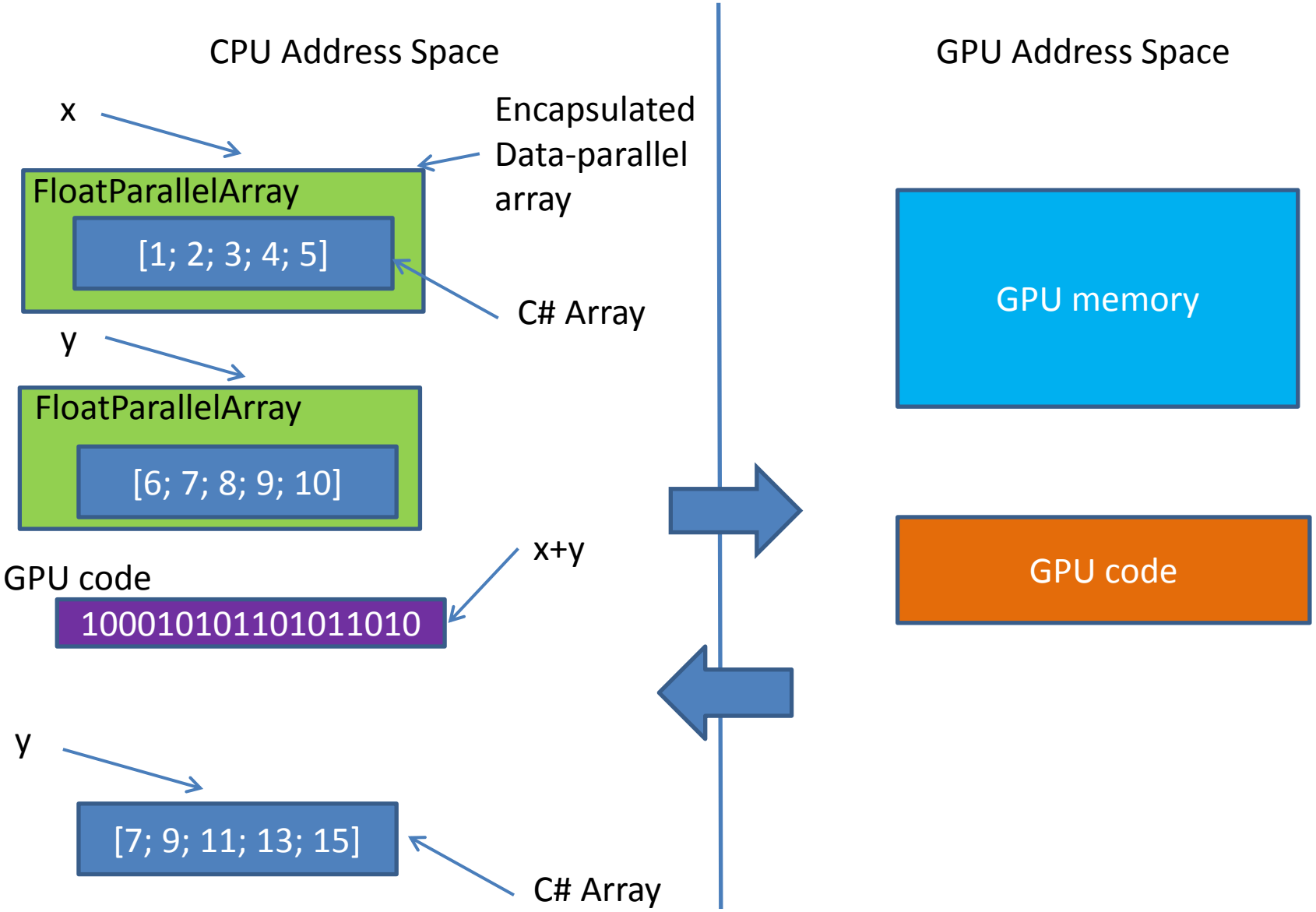
Sum(0)	$R_i = \sum_i A_{i,j}$
Sum(1)	$R_i = \sum_j A_{i,j}$
Maximum value (1)	$R_i = \max_j A_{i,j}$

Section $(b_i, c_i, s_i, b_j, c_j, s_j)$	$R_{i,j} = A_{b_i} + s_i \times i, b_j + s_j \times j$
Shift (m, n)	$R_{i,j} = A_{i-m,j-n}$
Rotate (m, n)	$R_{i,j} = A_{(i-m)\text{mod}M,(j-n)\text{mod}N}$
Replicate (m, n)	$R_{i,j} = A_{i \text{ mod } m,\ j \text{ mod } n}$
Expand (b_i, a_i, b_j, a_j)	$R_{i,j} = A_{i-b_i\text{mod}M,(j-b_j)\text{mod}N}$
Pad (m, a_i, m, a_j, c)	$R_{i,j} = \begin{cases} A_{i-m,j-n} & \text{if in bounds} \\ c & \text{otherwise} \end{cases}$
Transpose(1,0)	$R_{i,j} = A_{j,i}$

Drop Dimension (0)	$R_j = A_{0,j}$
Drop Dimension (1)	$R_i = A_{i,0}$
Add Dimension (1)	$R_{i,j,k} = A_{i,k}$

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

```
ps_3_0
dcl_2d  s0
dcl_texcoord0 v0.xy
dcl_2d  s1
texld   r0, v0, s0
texld   r1, v0, s1
add     r1,   r0,   r1
mov     oC0,   r1
```


CPU Address Space
GPU Address Space
x
Encapsulated
Data-parallel
array
FloatParallelArray
[1; 2; 3; 4; 5]
C# Array
y
FloatParallelArray
[6; 7; 8; 9; 10]
GPU memory
GPU code
x+y
GPU code
1000101011010110 10
y
[7; 9; 11; 13; 15]
C# Array

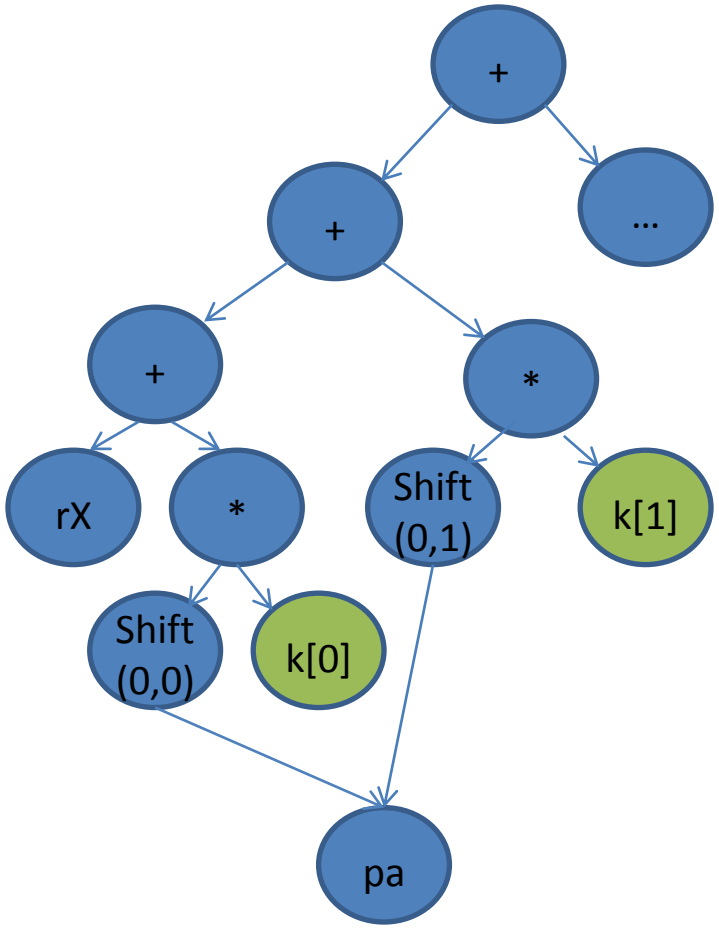
```
using System;
using Microsoft.ParallelArrays;

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) ;
        }
    }
}
```

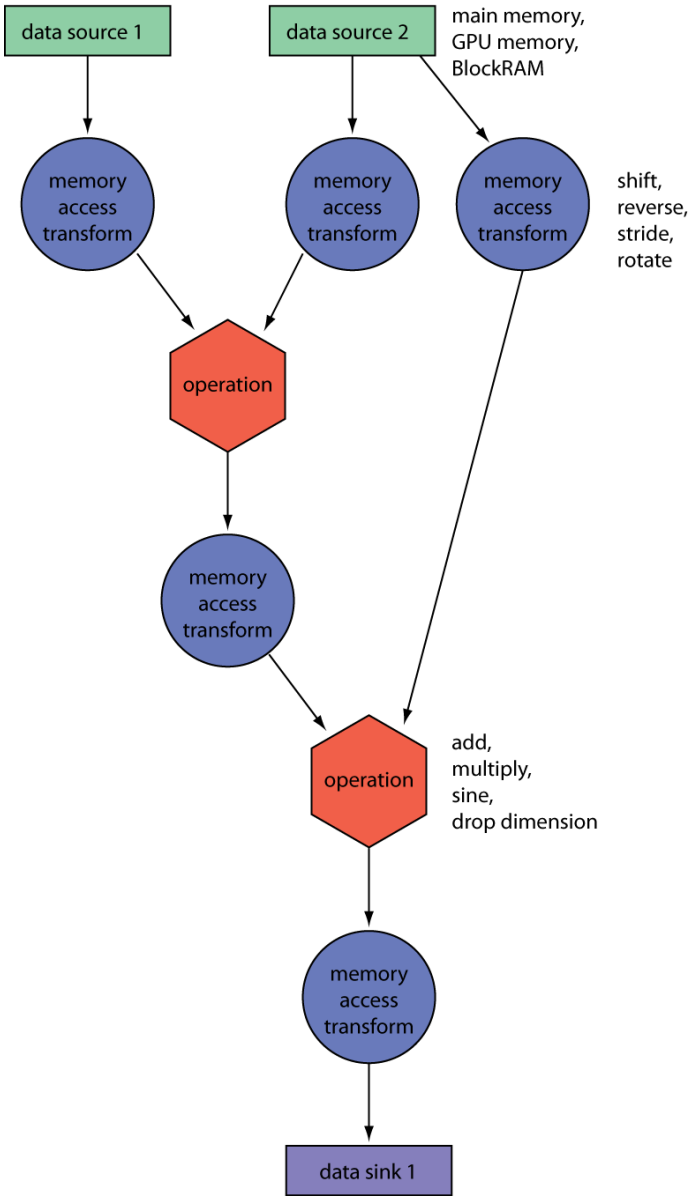
```
library ieee ;
use ieee.std_logic_1164.all ;
use work.addarrays_package.all ;
entity addarrays is
  port (signal clk, en, rst : in std_logic ;
        signal result : out float) ;
end entity addarrays ;

library ieee ;
use ieee.std_logic_unsigned.all ;
architecture accelerator of addarrays is
  attribute rom_style: string ;
  attribute ram_style: string ;
  attribute keep : string ;
  type net_1_array_type is array (0 to 4) of float ;
  signal net_1_array : net_1_array_type ; -- result  (*)
  attribute ram_style of net_1_array : signal is "block";
  signal net_1 : float ; -- Array input signal
  attribute keep of net_1 : signal is "true" ;
  type ext_1_array_type is array (0 to 4) of float ;
  signal ext_1_array : ext_1_array_type := (X"3f800000", X"40000000", X"40400000", X"40800000", X"40a00000") ;
  attribute rom_style of ext_1_array : signal is "block";
  signal ext_1_row_major : float := (others => '0') ; -- 1D array array output signal
  attribute keep of ext_1_row_major : signal is "true" ;
  signal net_2 : float ; -- Reference to array with external ID 1
  type ext_2_array_type is array (0 to 4) of float ;
  signal ext_2_array : ext_2_array_type := (X"40c00000", X"40e00000", X"41000000", X"41100000", X"41200000") ;
  attribute rom_style of ext_2_array : signal is "block";
  signal ext_2_row_major : float := (others => '0') ; -- 1D array array output signal
  attribute keep of ext_2_row_major : signal is "true" ;
  signal net_3 : float ; -- Reference to array with external ID 2
  signal net_4 : float := (others => '0') ;
  signal float_4_a : float := (others => '0') ;
  signal float_4_b : float := (others => '0') ;
  type ext_1_delayed_type is array (0 downto 0, 0 downto 0) of float ;
  signal ext_1_delayed : ext_1_delayed_type := (others => (others => (others => '0'))) ;
  type ext_2_delayed_type is array (0 downto 0, 0 downto 0) of float ;
  signal ext_2_delayed : ext_2_delayed_type := (others => (others => (others => '0'))) ;
```

```
    -- dimensions = (5) rank = 1
    variable col : integer := 0 ;
    variable col_shifted : integer ;
  begin
    wait until clk'event and clk='1' and en='1' ;
    if rst = '1' then
      col := 0 ;
    else
      col_shifted := col ;
      if col_shifted < 0 then
        col_shifted := 0 ;
      elsif col_shifted > 4 then
        col_shifted := 4 ;
      end if ;
      ext_2_delayed(0, 0) <= ext_2_array(col_shifted) ;
      if col < 4 then -- Advance along col
        col := col + 1 ;
      end if ; -- 1D array case
    end if ;
  end process gen_addr_ext_net_2_row_0 ;

  net_3 <= ext_2_delayed(0, 0) ;

  net_1_expr : process
  begin
    wait until clk'event and clk='1' and en='1' ;
    float_4_a <= net_2 ;
    float_4_b <= net_3 ;
  end process net_1_expr ;
  net_1 <= net_4 ;
  -- Section delay: 1 cycles
  result <= net_1 ;
  float_add_4 : floating_point_ieee_single_add port map (clk => clk, a => float_4_a, b => float_4_b, result => net_4) \
;

end architecture accelerator ;
```

```
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

```
namespace AddArrays1D
{
    class AddArrays1D
    {
        static void Main(string[] args)
        {
            FloatParallelArray a = new FloatParallelArray(new float[] {1.0f, 2.0f, 3.0f, 4.0f});
            FloatParallelArray b = new FloatParallelArray(new float[] {5.0f, 6.0f, 7.0f, 8.0f });
            FloatParallelArray c = a + b;
            Target gpuTarget = new DX9Target();
            float[] result = gpuTarget.ToArray1D(c);
            foreach (float f in result)
                Console.Write(f + " ");
            Console.WriteLine();
        }
    }
}
```

```
ps_3_0
dcl_2d s0
dcl_texcoord0 v0.xy
dcl_2d s1
texld   r0, v0, s0
texld   r1, v0, s1
add    r1,   r0,   r1
mov oC0,   r1
```

```
using System;
using Microsoft.ParallelArrays;
using FPA = Microsoft.ParallelArrays.FloatParallelArray;
namespace MultiplyAdd1D
{
    class MultiplyAdd1D
    {
        static void Main(string[] args)
        {
            FPA a = new FPA(new float[] { 1.0f, 2.0f, 3.0f, 4.0f });
            FPA b = new FPA(new float[] { 5.0f, 6.0f, 7.0f, 8.0f });
            FPA c = new FPA(new float[] { 9.0f, 10.0f, 11.0f, 12.0f });
            FPA d = ParallelArrays.MultiplyAdd(a, b, c);
            Target gpuTarget = new DX9Target();
            float[] result = gpuTarget.ToArray1D(d);
            foreach (float f in result)
                Console.Write(f + " ");
            Console.WriteLine();
        }
    }
}
```

```
ps_3_0
dcl_2d s0
dcl_texcoord0 v0.xy
dcl_2d s1
dcl_2d s2
texld   r0, v0, s0
texld   r1, v0, s1
texld   r2, v0, s2
mad    r2,   r0,   r1,   r2
mov oC0,   r2
```

```
static void Main(string[] args)
{
    Random random = new Random(42);
    FPA a = MakeRandomArray(3, 4, random);
    FPA b = MakeRandomArray(3, 4, random);
    FPA c = a + b;
    Target gpuTarget = new DX9Target();
    float[,] result = gpuTarget.ToArray2D(c);
    WriteArray(result);
}
```

```
ps_3_0
dcl_2d s0
dcl_texcoord0 v0.xy
dcl_2d s1
texld   r0, v0, s0
texld   r1, v0, s1
add    r1,   r0,   r1
mov oC0,   r1
```

```
int main()
{
     // Create a GPGPU computing resource
     DX9Target *tgtDX9 = CreateDX9Target() ;

     // Declare some sample input arrays
     float xvalues[5] = {1, 2, 3, 4,  5} ;
     float yvalues[5] = {6, 7, 8, 9, 10} ;

     // Create data-parallel versions of inputs
     FPA x = FPA(xvalues, 5) ;
     FPA y = FPA(yvalues, 5) ;

     // Specify data-parallel computation
     FPA z = x + y ; // Computation does not occur here...

     // Allocate space for the result array
     float* zvalues = (float*) malloc (5 * sizeof(float)) ;

     // Execute the data-parallel computation on the GPU
     tgtDX9->ToArray(z, zvalues, 5) ; // z = x + y is now evaluated

     // Write out the result
     for (int i = 0; i < 5; i++)
               cout << zvalues[i] << " " ;
     cout << endl ;
}
```

$$
\begin{aligned}
cnd(d) &= 1 - x \quad \text{if} \quad x < 0 \\
&= x \quad \text{otherwise} \\
x &= 1/\sqrt{2\pi} e^{-d^2/2} poly \\
poly &= horner(a, k) \\
horner(a, k) &= k(a_1 + k(a_2 + k(a_3 + k(a + 4 + ka_5)))) \\
k &= 1/(1 + 0.2316419\,|d|) \\
a &= [0.31938153, -0.356563782, 1.781477937, \\
&\quad -1.821255978, 1.330274429]
\end{aligned}
$$

$$
\begin{aligned}
V_{call} &= S \cdot cnd(d_1) - X \cdot e^{-rT} \cdot cnd(d2) \\
V_{put} &= X \cdot e^{-rT} \cdot cnd(-d_2) - S \cdot cnd(-d_1)
\end{aligned}
$$

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

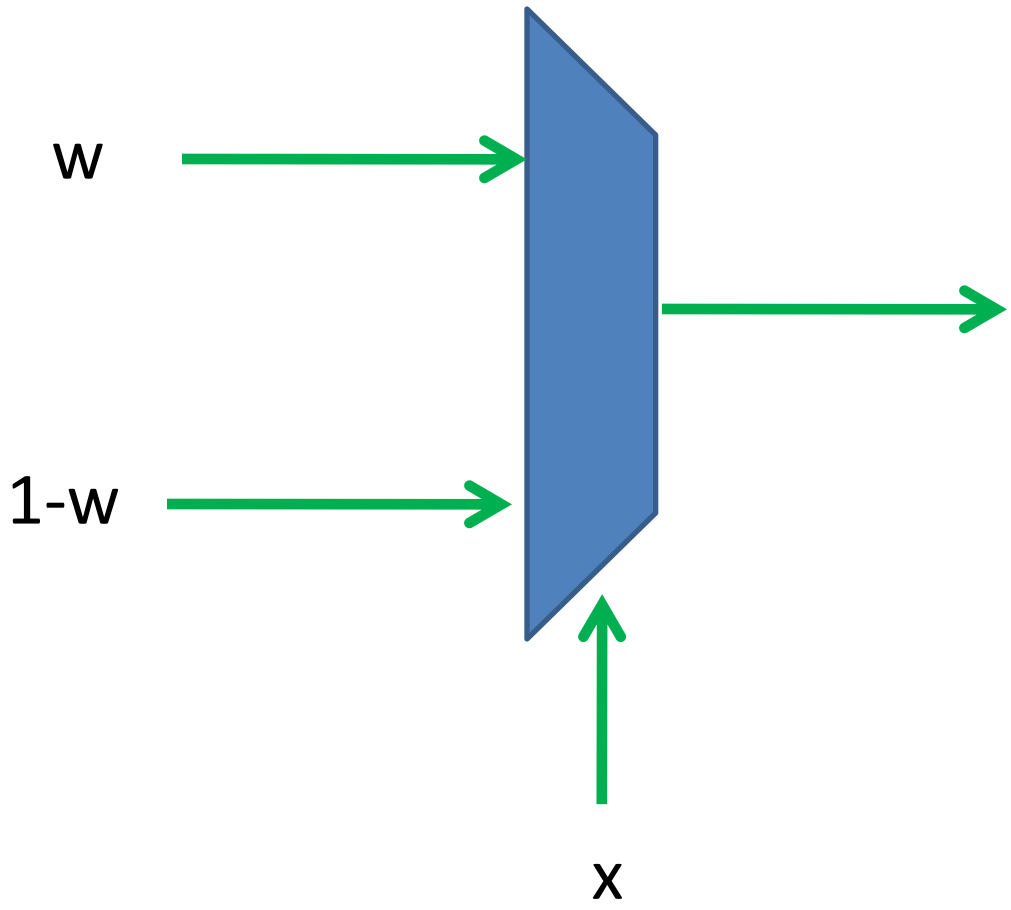
```
static FloatParallelArray Horner(float[] coe, FloatParallelArray x)
{
    FloatParallelArray result = new FloatParallelArray(0.0f, x.Shape);
    foreach (var c in coe)
    {
        result = result + x * c;
    }
    return result;
}
```

```
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);
}
```

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

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

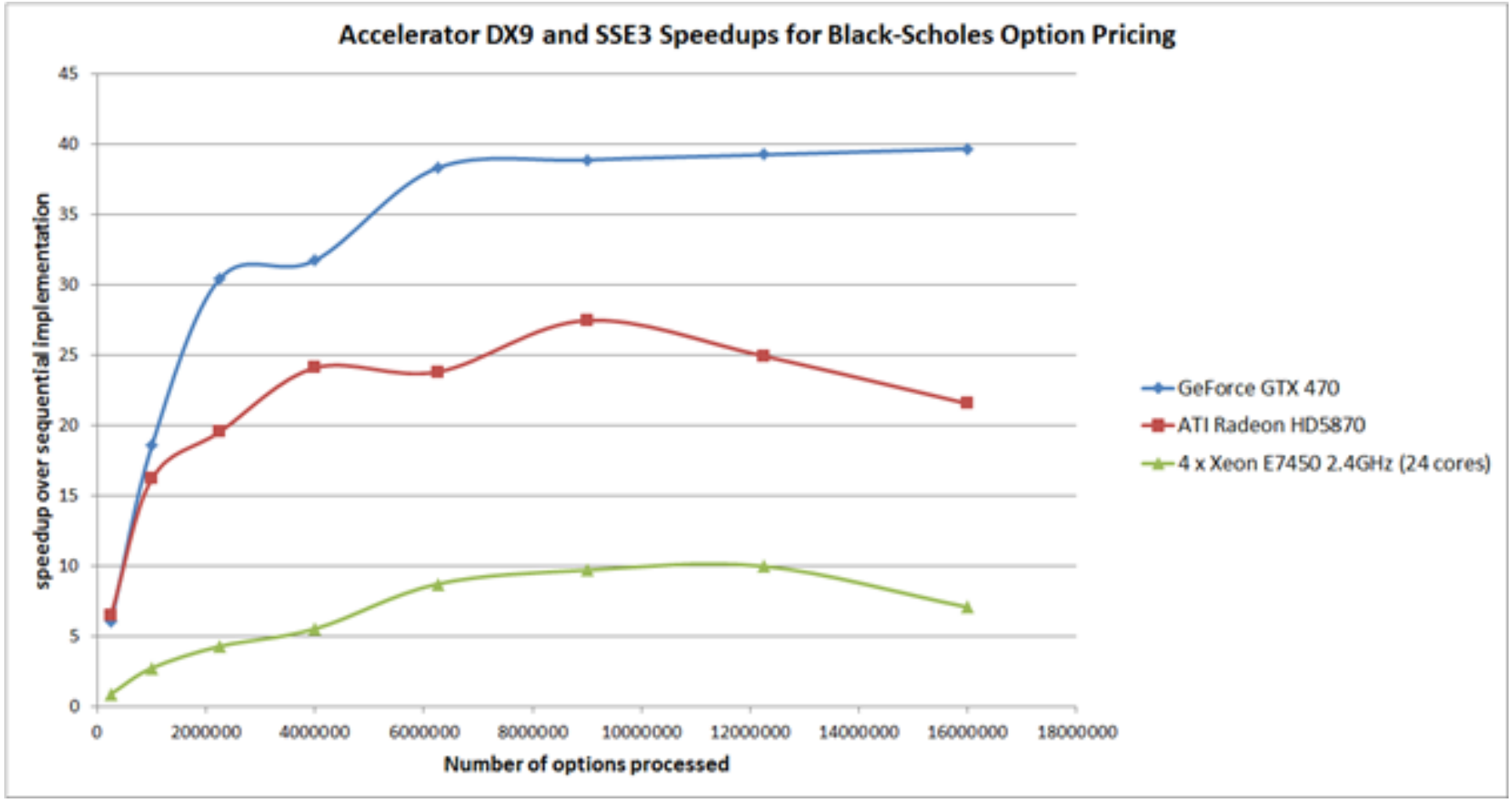

w
1-w
x

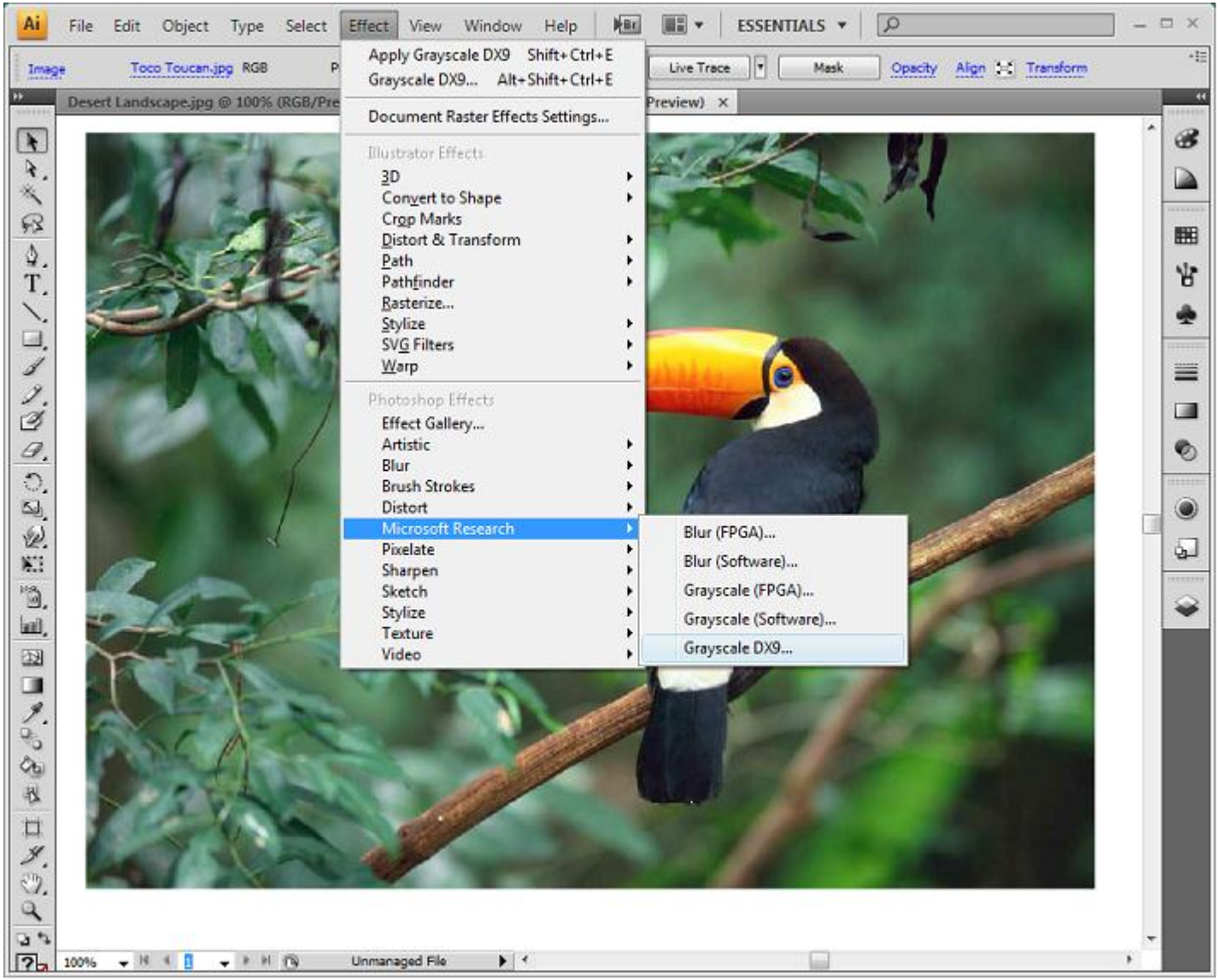
```
static float BlackScholes1(float s, float x, float t, float r, float v)
{
    float d1 = (float)((Math.Log(s / x) +
                        (r + v * v / 2) * t) / (v * Math.Sqrt(t)));
    float d2 = (float)(d1 - v * Math.Sqrt(t));
    return (float)(s * NormCdf(d1) - x * Math.Exp(-r * t) * NormCdf(d2));
}
```

```
static FloatParallelArray BlackScholes1(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));
}
```

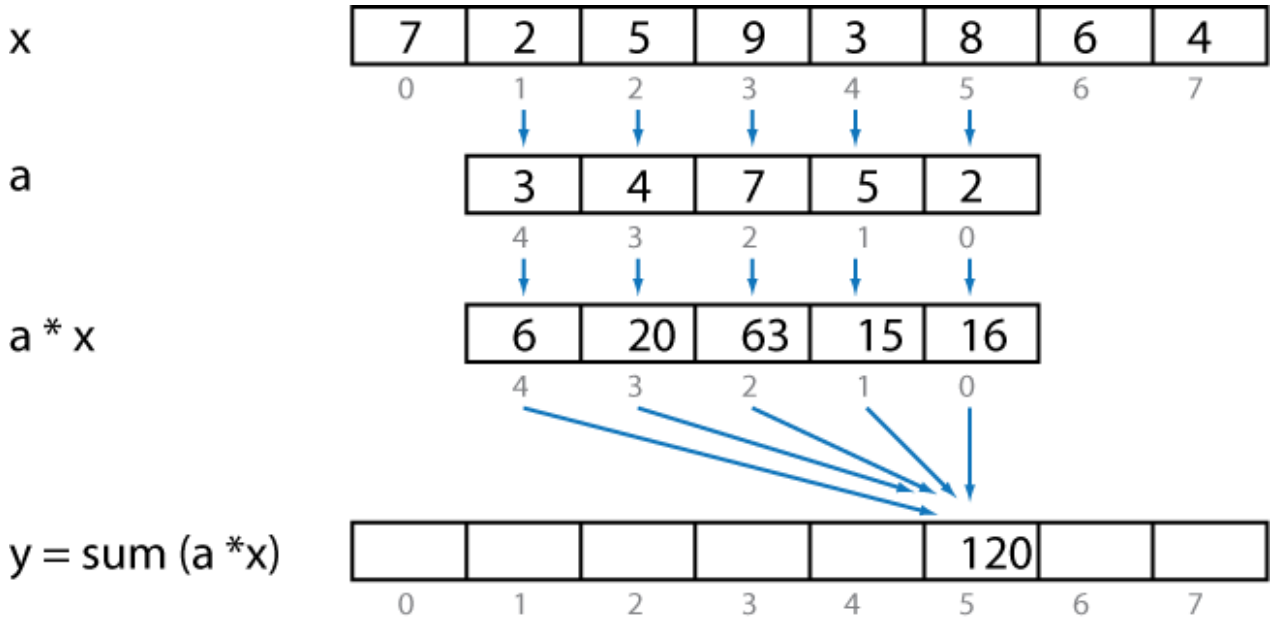
```
static float[] BlackScholes(float[] ss, float[] xs, float[] ts)
{
    float r = 1.3f;
    float v = 2.5f;
    var result = new float[ss.GetLength(0)];
    for (int i = 0; i < ss.GetLength(0); i++)
    {
        result[i] = BlackCholes1(ss[i], xs[i], ts[i], r, v);
    }
    return result;
}
```

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


Accelerator DX9 and SSE3 Speedups for Black-Scholes Option Pricing
speedup over sequential implementation
45
40
35
30
25
20
15
10
5
0
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
16000000
18000000
Number of options processed
GeForce GTX 470
ATI Radeon HD5870
4 x Xeon E7450 2.4GHz (24 cores)


File Edit Object Type Select Effect View Window Help
ESSENTIALS
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...
Live Trace
Mask
Opacity
Align
Transform
100%
Unmanaged File

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



```
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;
        }
```

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

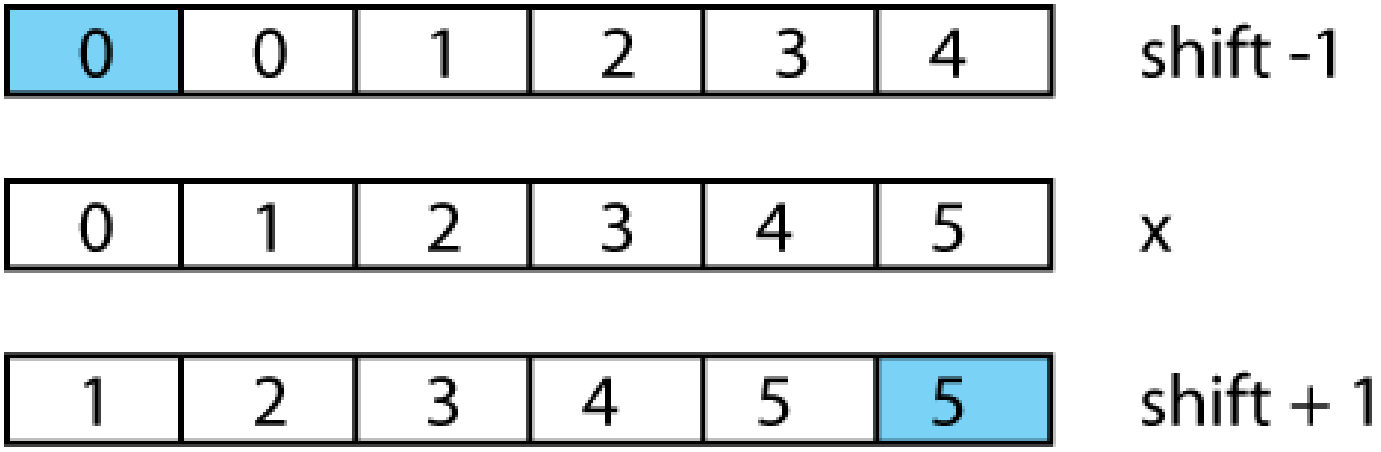
The Accidental Semi-colon

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

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

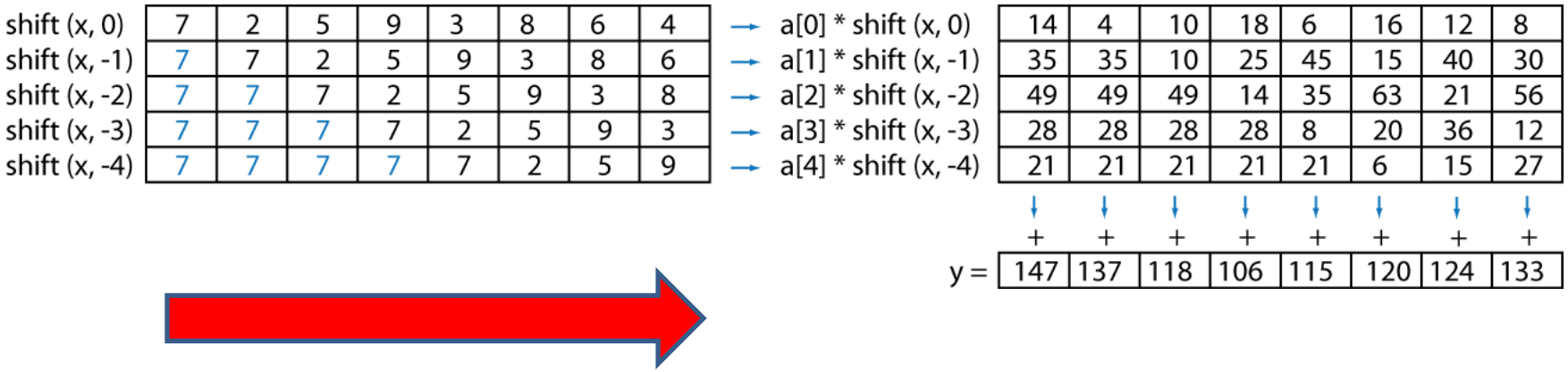
$$\begin{aligned}
y &= [y[0], y[1], y[2], y[3], y[4], y[5], y[6], y[7]] \\
&= a[0] * [x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]] + \\
&\quad a[1] * [x[-1], x[0], x[1], x[2], x[3], x[4], x[5], x[6]] + \\
&\quad a[2] * [x[-2], x[-1], x[0], x[1], x[2], x[3], x[4], x[5]] + \\
&\quad a[3] * [x[-3], x[-2],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

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]

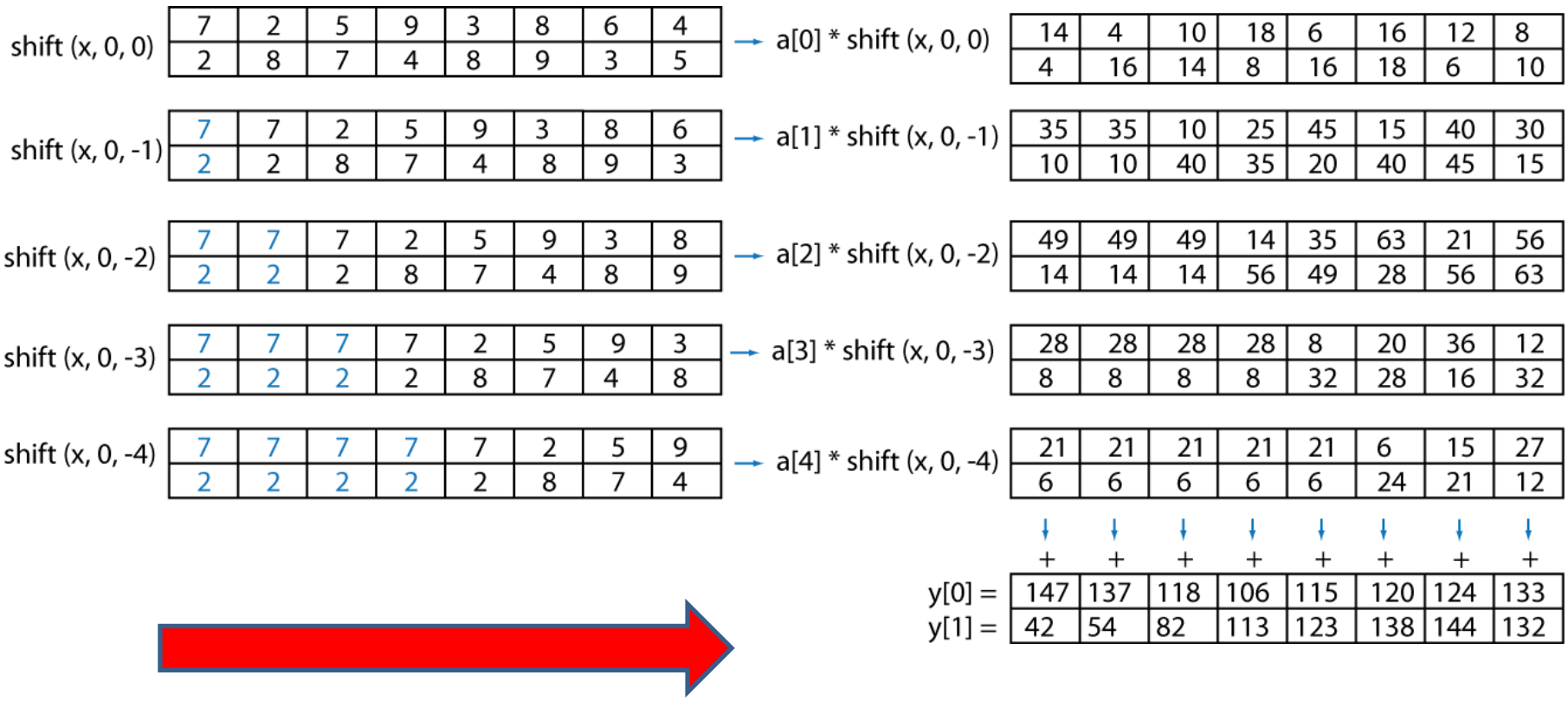


$$
\begin{aligned}
y &= [y[0], y[1], y[2], y[3], y[4], y[5], y[6], y[7]] \\
&= a[0] * [x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]] + \\
&\quad a[1] * [x[-1], x[0], x[1], x[2], x[3], x[4], x[5], x[6]] + \\
&\quad a[2] * [x[-2], x[-1], x[0], x[1], x[2], x[3], x[4], x[5]] + \\
&\quad a[3] * [x[-3],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

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


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 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;
        }
    }
}
```

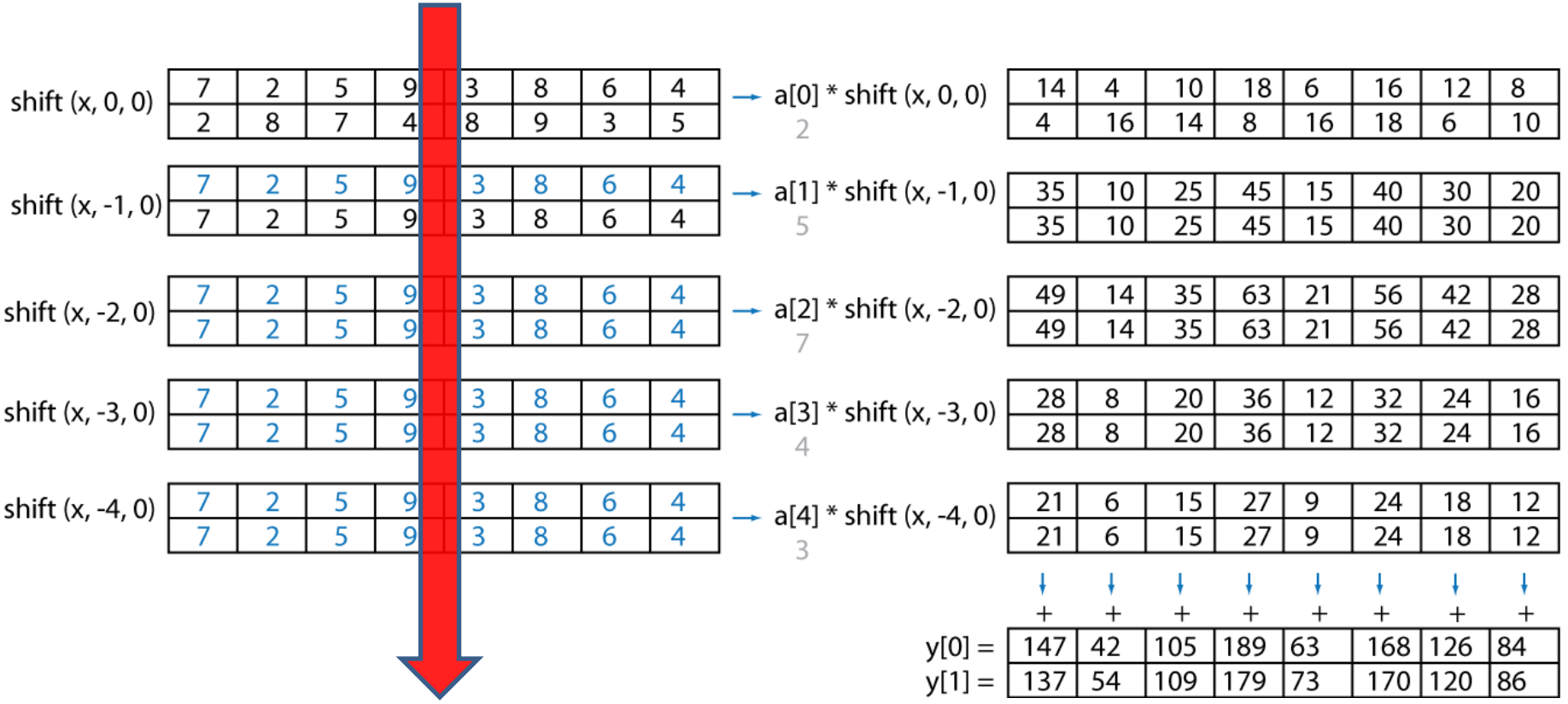

shift (x, 0, 0)
7 2 5 9 3 8 6 4
2 8 7 4 8 9 3 5
a[0] * shift (x, 0, 0)
14 4 10 18 6 16 12 8
4 16 14 8 16 18 6 10
shift (x, 0, -1)
7 7 2 5 9 3 8 6
2 2 8 7 4 8 9 3
a[1] * shift (x, 0, -1)
35 35 10 25 45 15 40 30
10 10 40 35 20 40 45 15
shift (x, 0, -2)
7 7 7 2 5 9 3 8
2 2 2 8 7 4 8 9
a[2] * shift (x, 0, -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);
}
```

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

```
ps_3_0
dcl_2d s0
dcl_texcoord0 v0.xy
dcl_texcoord1 v1.xy
dcl_texcoord2 v2.xy
dcl_texcoord3 v3.xy
dcl_texcoord4 v4.xy
def c0, 0.054489, 0.054489, 0.054489, 0.054489
def c1, 0.000000, 0.000000, 0.000000, 0.000000
def c2, 0.244201, 0.244201, 0.244201, 0.244201
def c3, 0.402620, 0.402620, 0.402620, 0.402620
texld   r0, v0, s0
mul       r0,   r0,   c0
add       r0,   c1,   r0
texld   r1, v1, s0
mul       r1,   r1,   c2
add       r1,   r0,   r1
texld   r2, v2, s0
mul       r2,   r2,   c3
add       r2,   r1,   r2
texld   r3, v3, s0
mul       r3,   r3,   c2
add       r3,   r2,   r3
texld   r4, v4, s0
mul       r4,   r4,   c0
add       r4,   r3,   r4
mov oC0,   r4
```


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();
        }
    }
  }
}
```

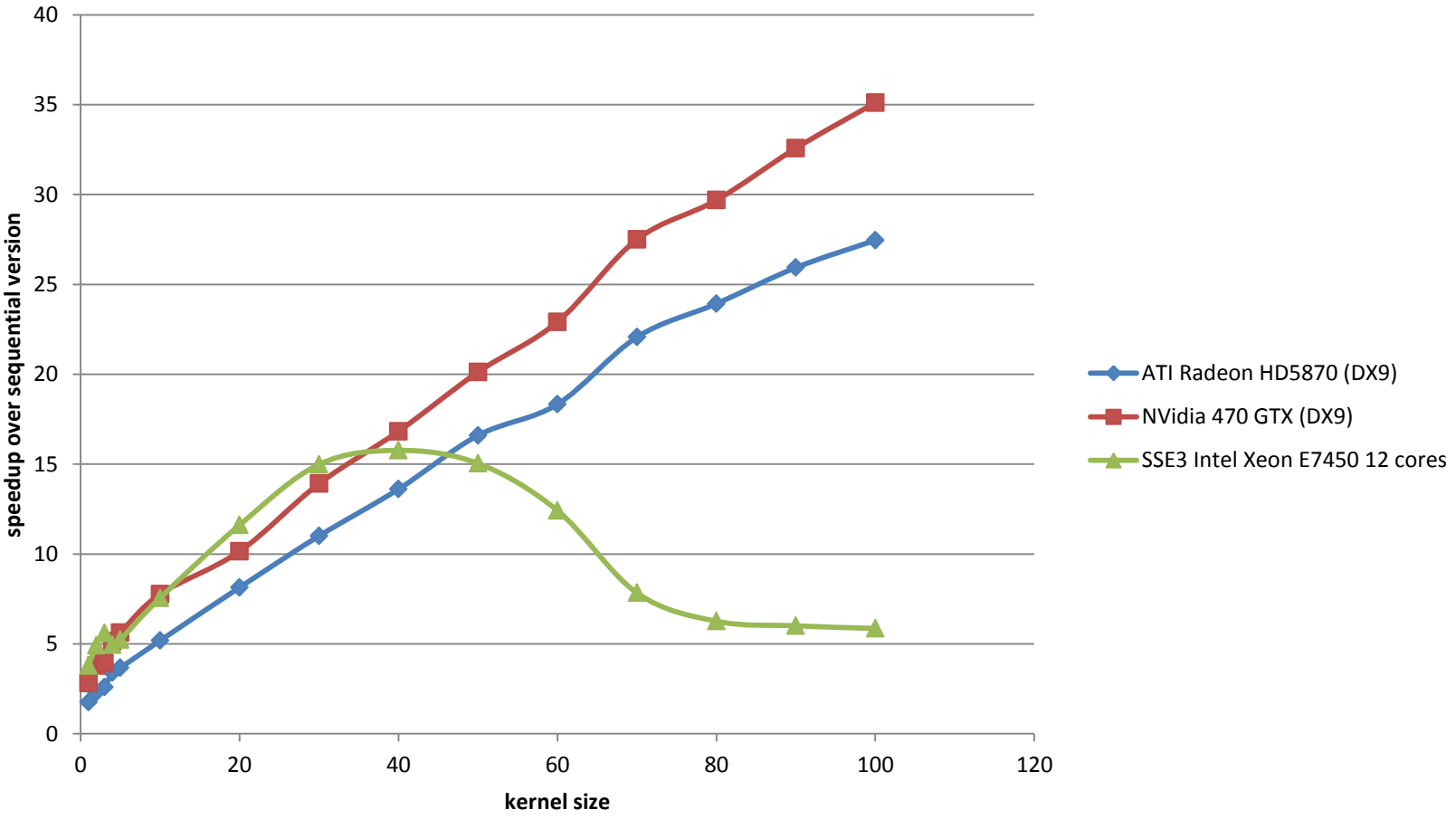
```
FPA ConvolveXY(Target &tgt, int height, int width, int filterSize, float filter[],
               FPA input, float *resultArray)
{
    // Convolve in X (row) direction.
    size_t dims[] = {height,width};
    FPA smoothX = FPA(0,dims, 2);
    intptr_t counts[] = {0,0};
    int filterHalf = filterSize/2;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

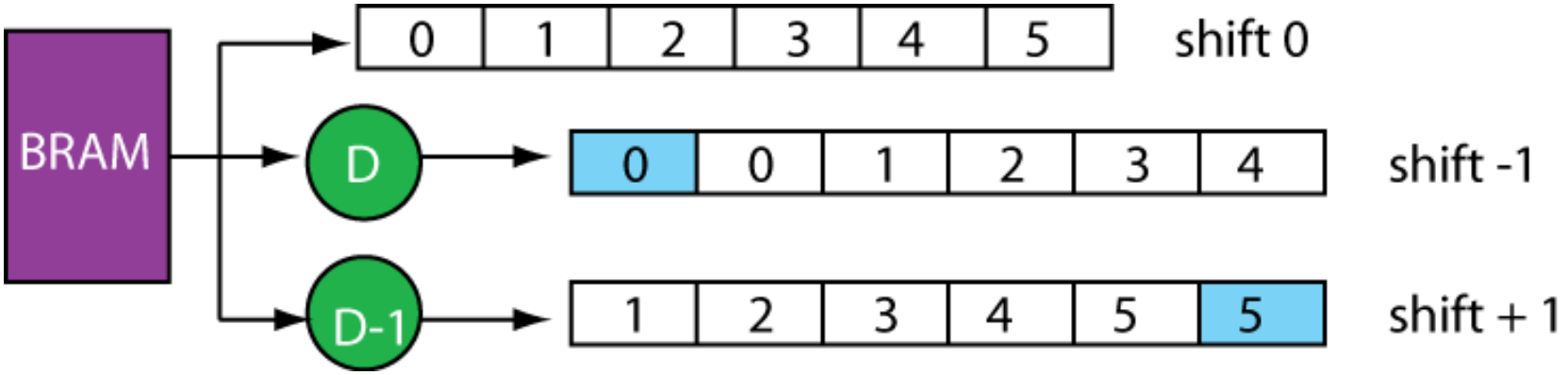
```
open System
open Microsoft.ParallelArrays
[<EntryPoint>]
let main(args) =
    // Declare a filter kernel for the convolution
    let testKernel = Array.map float32 [| 2; 5; 7; 4; 3 |]
    // Specify the size of each dimension of the input array
    let inputSize = 10
    // Create a pseudo-random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

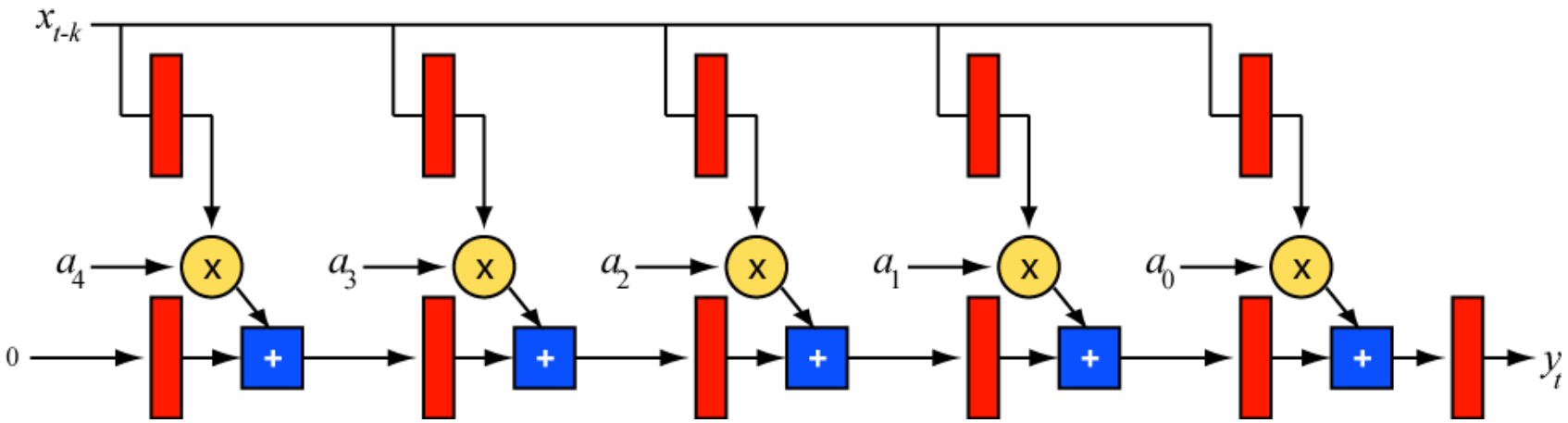
```
            e + convolve shifts kernel (i-1) a
    // Declare a 2D convolver
    let convolveXY kernel input
       = // First convolve in the X direction and then in the Y direction
         let convolveX = convolve (fun i -> [| -i; 0 |]) kernel (kernel.Length - 1) input
         let convolveY =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

Speedup using Accelerator GPU and SSE3 multicore targets for a 8000x8000 convolver

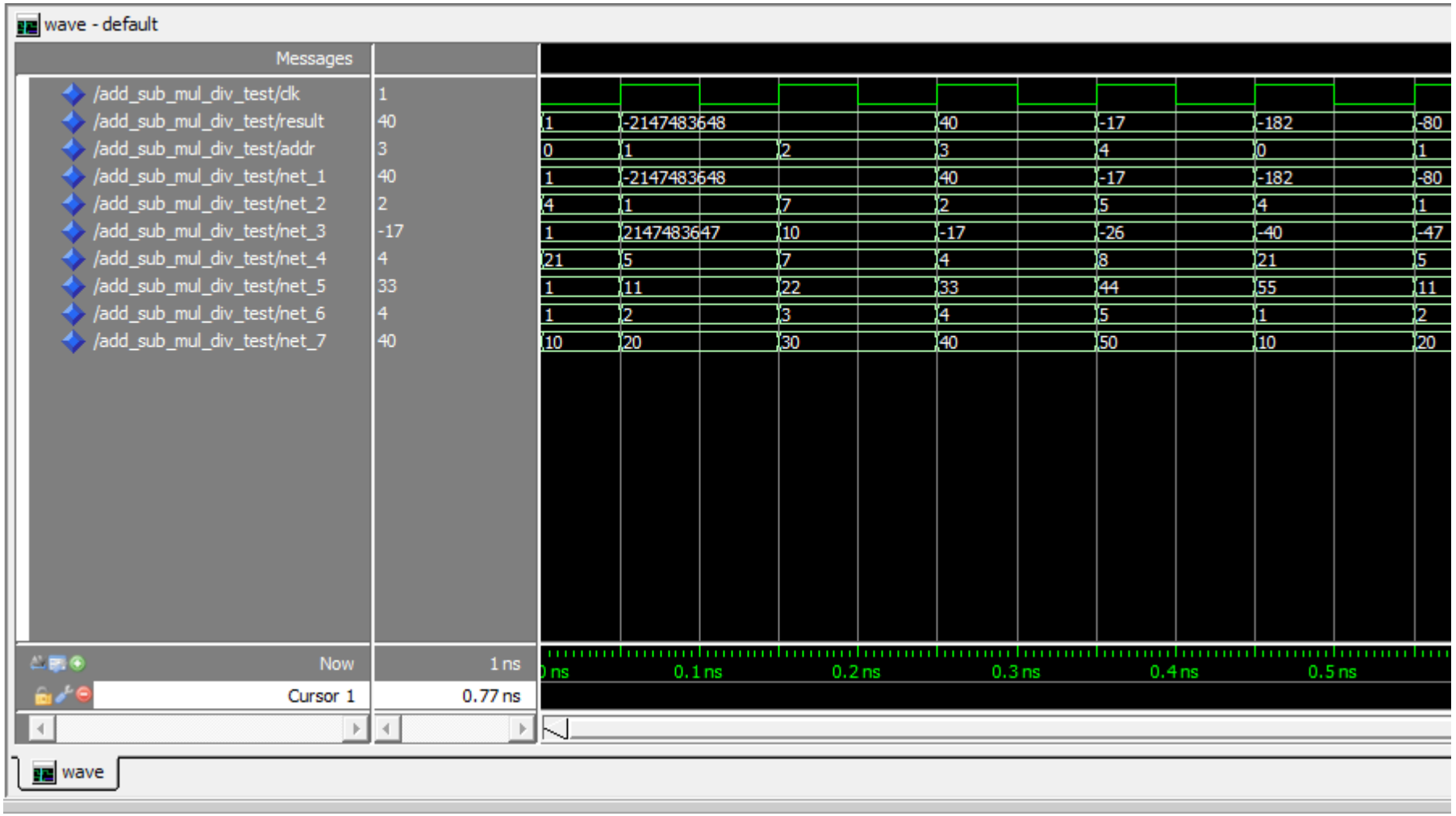


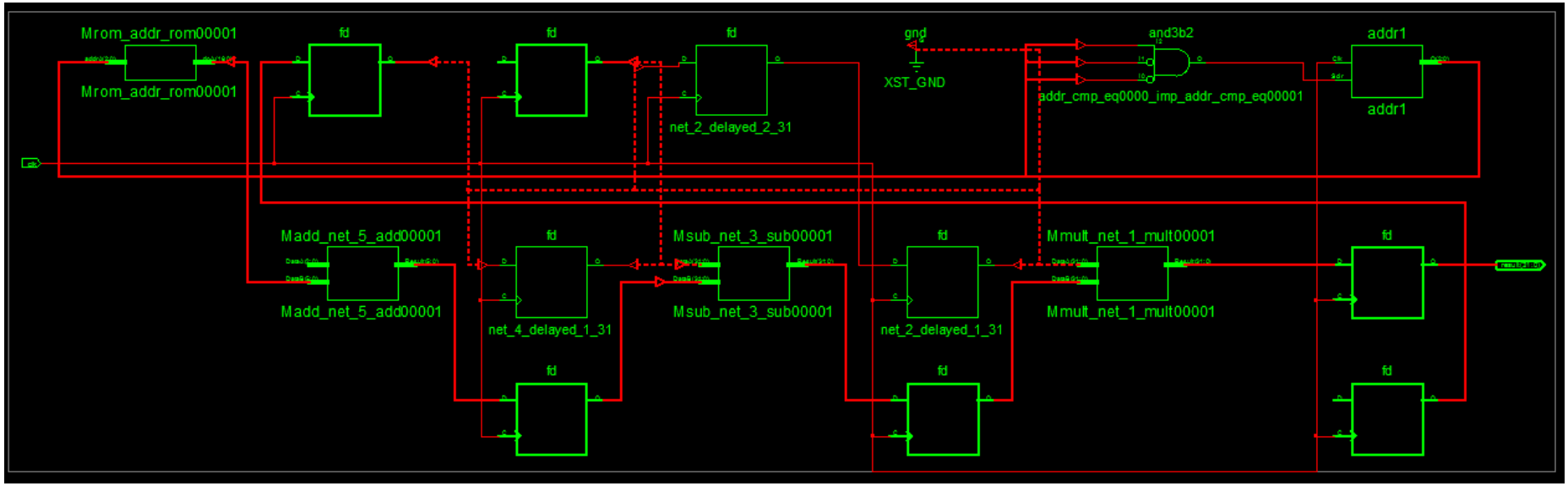

BRAM
D
D-1
0
1
2
3
4
5
shift 0
0
0
1
2
3
4
shift -1
1
2
3
4
5
5
shift + 1

Convolver

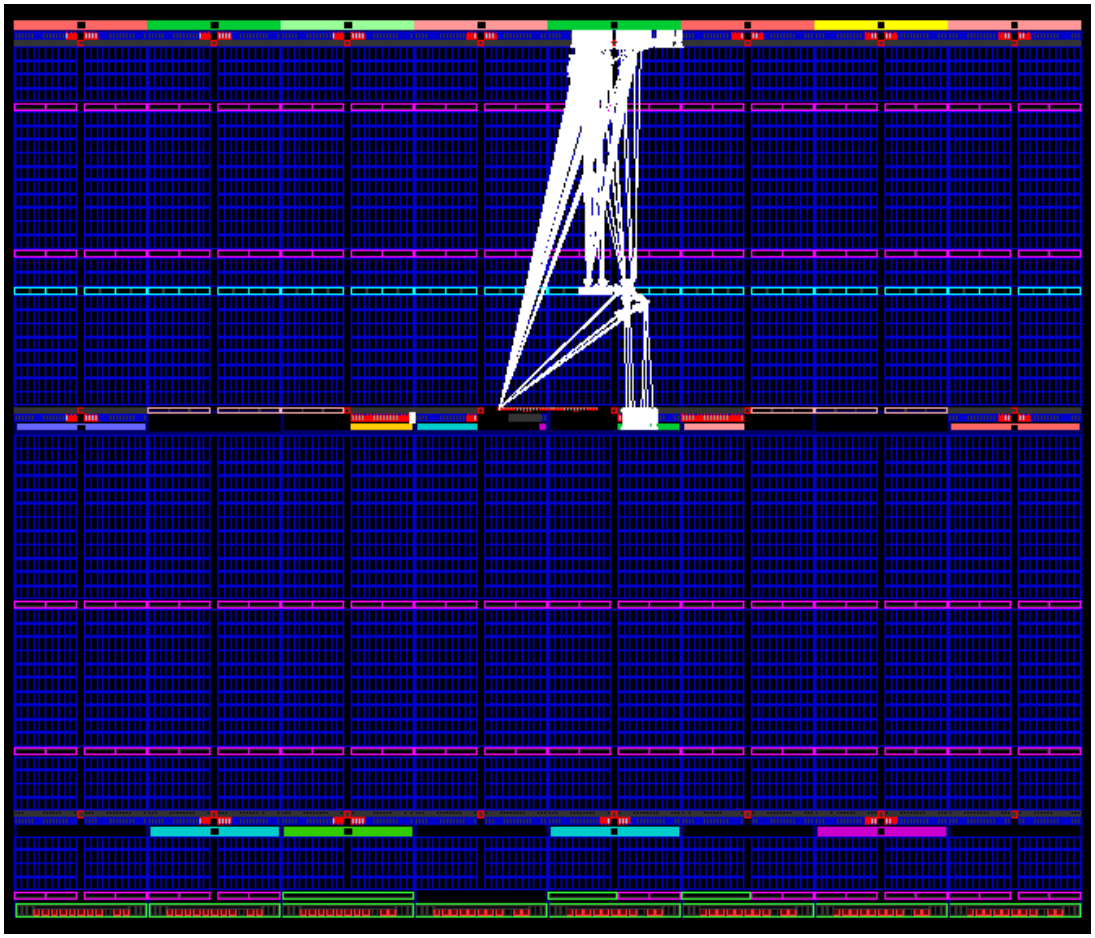


$$y_t = \sum_{k=0}^{N-1} a_k x_{t-k}$$

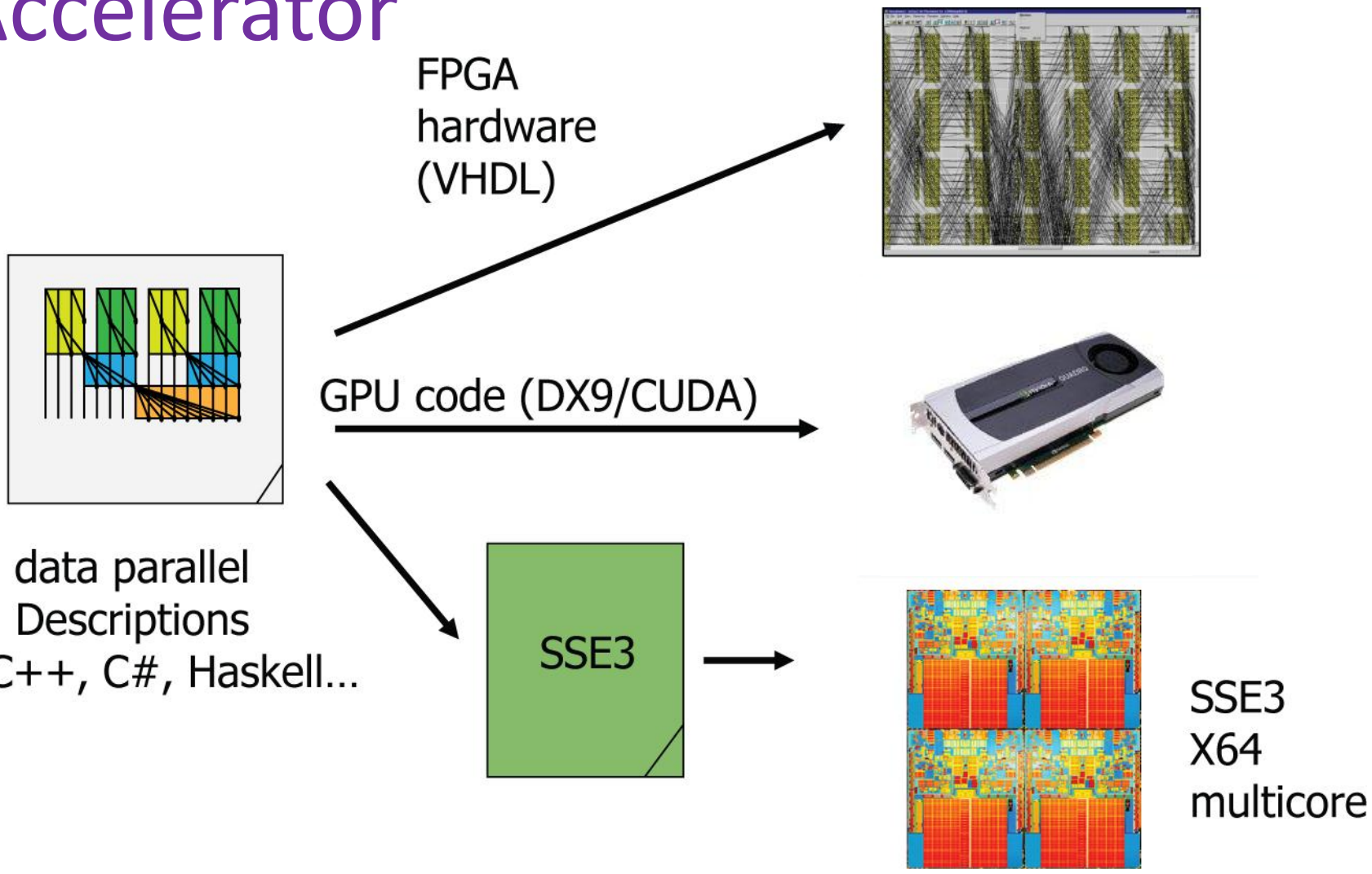

wave - default
Messages
/add_sub_mul_div_test/clk
/add_sub_mul_div_test/result
/add_sub_mul_div_test/addr
/add_sub_mul_div_test/net_1
/add_sub_mul_div_test/net_2
/add_sub_mul_div_test/net_3
/add_sub_mul_div_test/net_4
/add_sub_mul_div_test/net_5
/add_sub_mul_div_test/net_6
/add_sub_mul_div_test/net_7
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



Accelerator



DOI:10.1145/1978542.1978558

Article development led by ACM QUEUE
queue.acm.org

Heterogeneous systems allow us to target our programming to the appropriate environment.

BY SATNAM SINGH

Computing Without Processors

FROM THE PROGRAMMER'S perspective the distinction between hardware and software is being blurred. As programmers struggle to meet the performance requirements of today's systems they will face an ever increasing need to exploit alternative computing elements such as graphics processing units (GPUs), which are graphics cards subverted for data-parallel computing,[11] and field-programmable gate arrays (FPGAs), or soft hardware.

The current iteration of mainstream computing architectures is based on cache-coherent multicore processors. Variations on this theme include Intel's experimental Single-Chip Cloud Computer, which contains 48 cores that are not cache coherent. This path, however, is dictated by the end of frequency scaling rather than being driven by requirements about how programmers wish to write software.[4]

systems are largely based on abstractions developed for writing operating systems (for example, locks and monitors). However, these are not the correct abstractions to use for writing parallel applications.

There are better ways to bake all that sand. Rather than composing many elements that look like regular CPUs, a better approach, from a latency and energy-consumption perspective, is to use a diverse collection of processing elements working in concert and tuned to perform different types of computation and communication. Large coarse-grain tasks are suitable for implementation on multicore pro-

Computing without Processors

view issue

by Satnam Singh | June 27, 2011

Topic: Computer Architecture

View Comments Print +1 7 SHARE

Site Feeds
Twitter feed

BROWSE TOPICS
Programming Languages
Development
Databases
Privacy and Rights
Quality Assurance
System Evolution
Full List of Topics

COLUMNS
The Bikeshed
Kode Vicious
Curmudgeon
Opinion

DISCUSSIONS
CTO Roundtables
Case Studies
Interviews

LATEST COMMENTS
- "How pleasurable to see that good code parallels good design. Clr..." Coding Guidelines
- "Firstly. Whatever style is used why not pick an IDE that automa ..." Coding Guidelines
- "So what database or databases does Amazon

Heterogeneous systems allow us to target our programming to the appropriate environment.

SATNAM SINGH, MICROSOFT RESEARCH CAMBRIDGE, UK

From the programmer's perspective the distinction between hardware and software is being blurred. As programmers struggle to meet the performance requirements of today's systems, they will face an ever increasing need to exploit alternative computing elements such as GPUs (graphics processing units), which are graphics cards subverted for data-parallel computing,[11] and FPGAs (field-programmable gate arrays), or soft hardware.

The current iteration of mainstream computing architectures is based on cache-coherent multicore processors. Variations on this theme include Intel's experimental Single-Chip Cloud Computer, which contains 48 cores that are not cache coherent. This path, however, is dictated by the end of frequency scaling rather than being driven by requirements about how programmers wish to write software.[4] The conventional weapons available for writing concurrent and parallel software for such multicore systems are largely based on abstractions developed for writing operating systems (e.g., locks and monitors). However, these are not the right abstractions to use for writing parallel applications.

There are better ways to bake all that sand. Rather than composing many elements that look like regular CPUs, a better approach, from a latency and energy-consumption perspective, is to use a diverse collection of processing elements working in concert and tuned to perform different types of computation and communication. Large coarse-grain tasks are suitable for implementation on multicore processors. Thousands of fine-grain data-parallel computations are

NEW QUEUE CONTENT ON SLASHDOT

- OCaml for the Masses
Yaron Minsky

- The Software Industry IS the Problem
Poul-Henning Kamp

- Any Competing Whois Registry Model is Doomed
Paul Vixie

RELATED CONTENT

ON PLUG-INS AND EXTENSIBLE ARCHITECTURES
Extensible application architectures such as Eclipse offer many advantages, but one must be careful to avoid "plug-in hell."
Dorian Birsan

BROWSE THIS TOPIC:

COMPUTER ARCHITECTURE

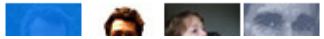
+Satnam Gmail Calendar Documents Photos Sites Web More Satnam Singh 0

Google+ Search Google+

Send a message

Send an email

In Satnam's circles (225)



Satnam Singh

Misanthrope programmer.

Edit Profile

Posts About Photos Videos +1's View profile as...

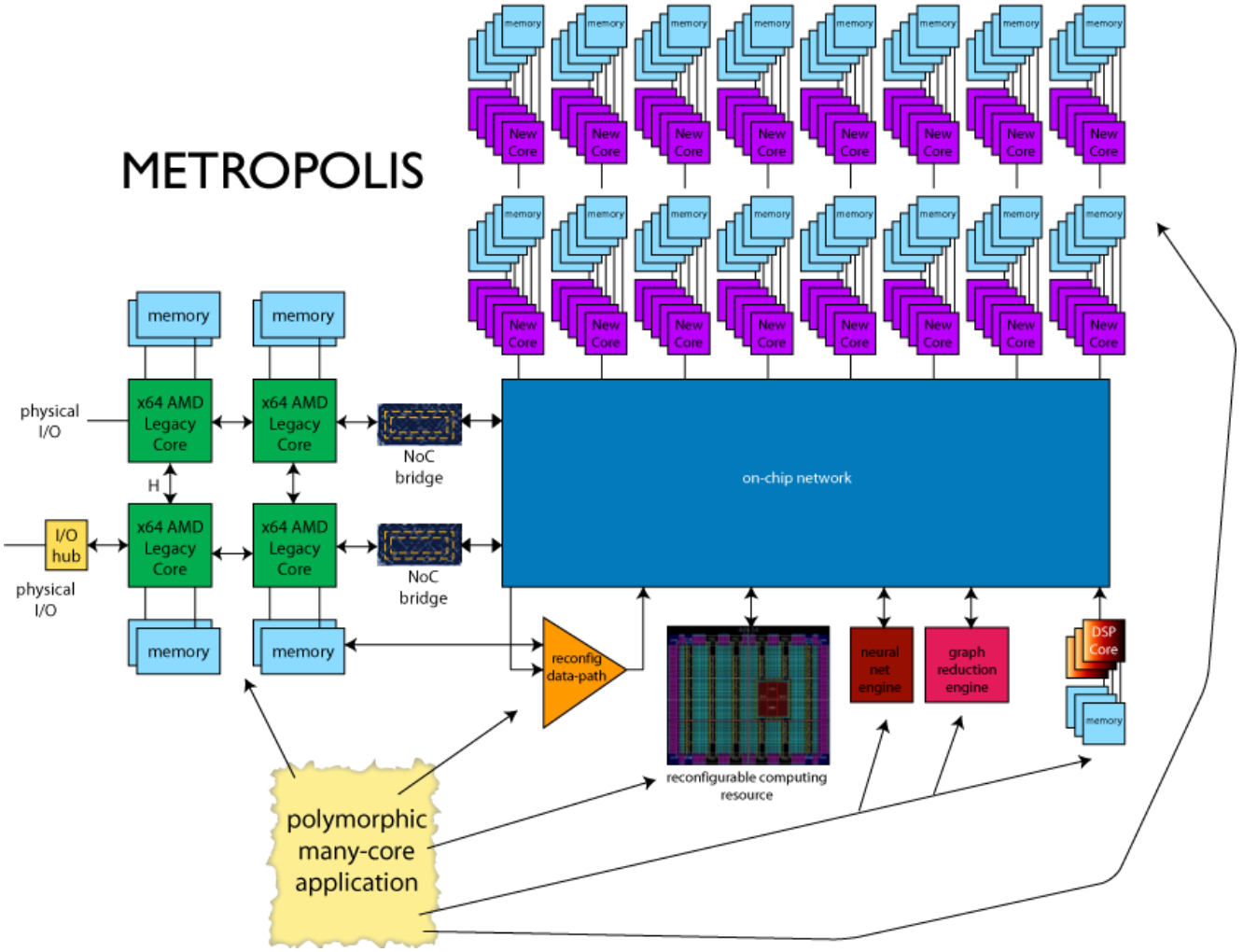
Satnam Singh - 14 Nov 2011 (edited) - Public

Several people have complained that I am not answering my Microsoft email. My Microsoft email got cut off over a month ago and email sent to satnams@microsoft.com disappears silently and the sender does not get a message delivery failure email. This is very frustrating and annoying behaviour by Microsoft and there is nothing I can do about it. Please send email to satnam@raintown.org instead (or s.singh@acm.org or s.singh@ieee.org).

+1 - Comment - Share

Suhaib Fahmy - It's a feature of Exchange ;)
Yesterday 09:22

Add a comment...


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

GPU TECHNOLOGY CONFERENCE

Register for the Next GTC Express

GPU-enabled Macromolecular Simulation: Challenges and Opportunities

Michela Taufer,, Assistant Professor, Department of Computer and Information Sciences, University of Delaware

Thursday, December 1, 2011, 9:00 AM PDT

Register at www.gputechconf.com