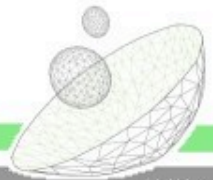


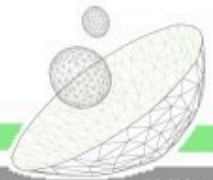
# Altimesh Hybridizer™

Enabling Accelerators in .Net and more

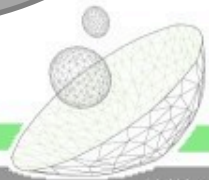
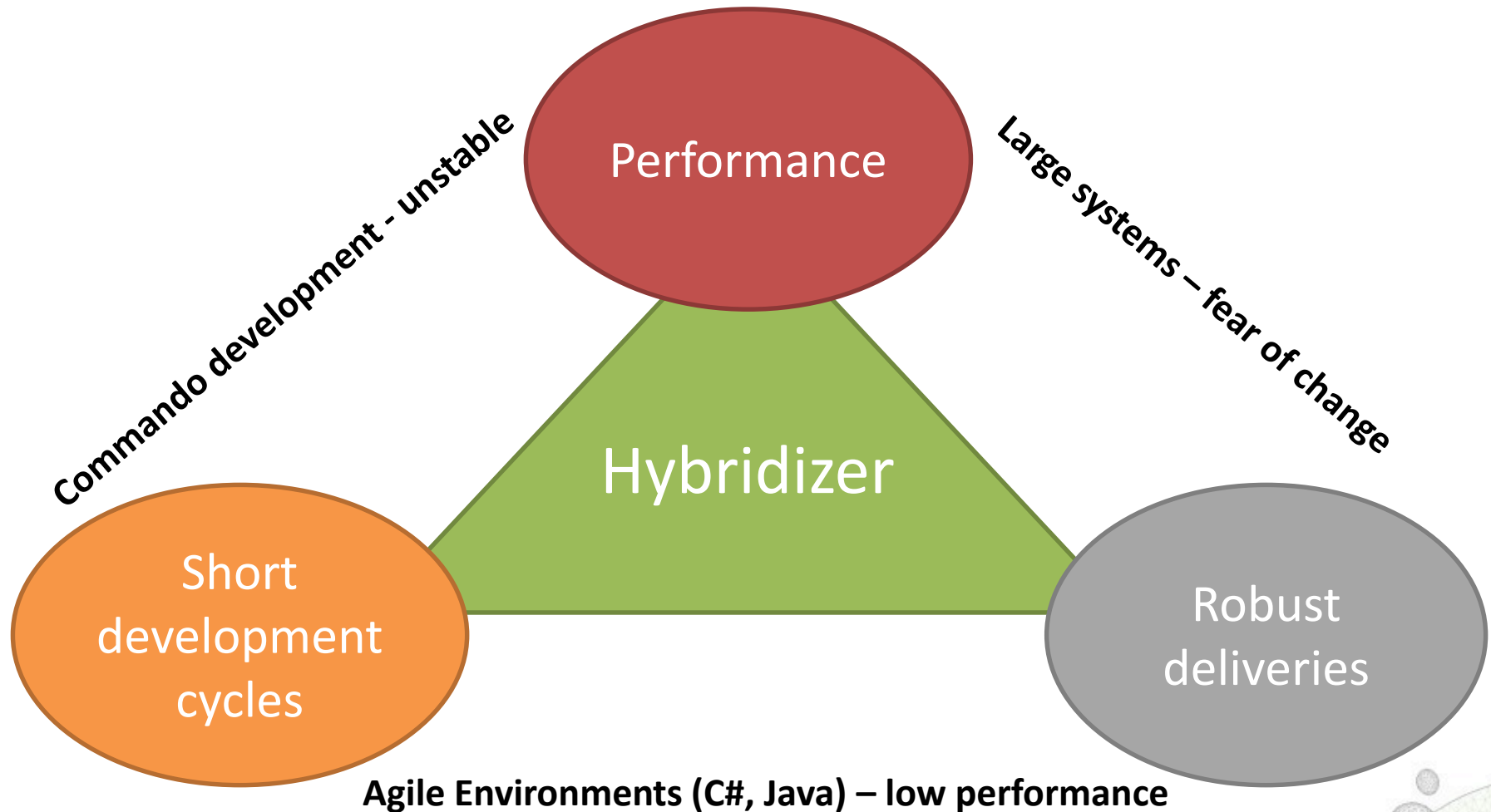


So many platforms, so few experts...

**WHY THE HYBRIDIZER ?**



# Software development teams accommodate external constraints



# Why the Hybridizer?

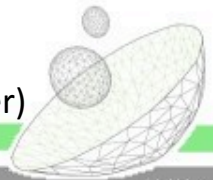
- Develop in a managed environment (C#/Java)
  - Fast developments (fast compile time, edit and continue...)
  - Testing and refactoring ecosystem
  - Glitch-safe memory management
  - Embrace Change

HIGHER PRODUCTIVITY   REDUCE TCO   OF APPLICATION DEVELOPMENT

- Benefit from manycore architectures
  - ***With single version of the source code***
  - Obtain first grade performances (use >80% of peak)
  - Fine tune optimizations with debugger/profiler integration
  - Variety of execution platforms
  - Change execution target without rewriting code

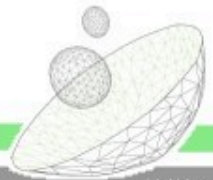
MORE EFFICIENT HARDWARE   REDUCE TCO   OF DATA CENTERS

IT spending : approx 30% in hardware and approx 20% in application development (Source : Gartner)



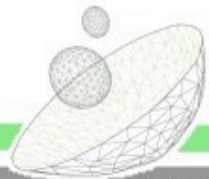
# What the Hybridizer is not

- Hybridizer is not a magic wand: some hints have to be given
  - Memory management is performed either in a naïve way, or needs to be done by hand
  - Memory level usages need to be defined
  - Some execution behaviors cannot be guessed
- Work distribution needs to be explicit
  - Loop parallelization is not automatic
  - Concurrency needs to be handled by hand
  - Code patterns need to be changed from sequential to parallel

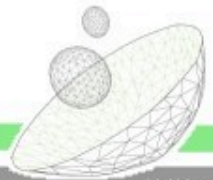
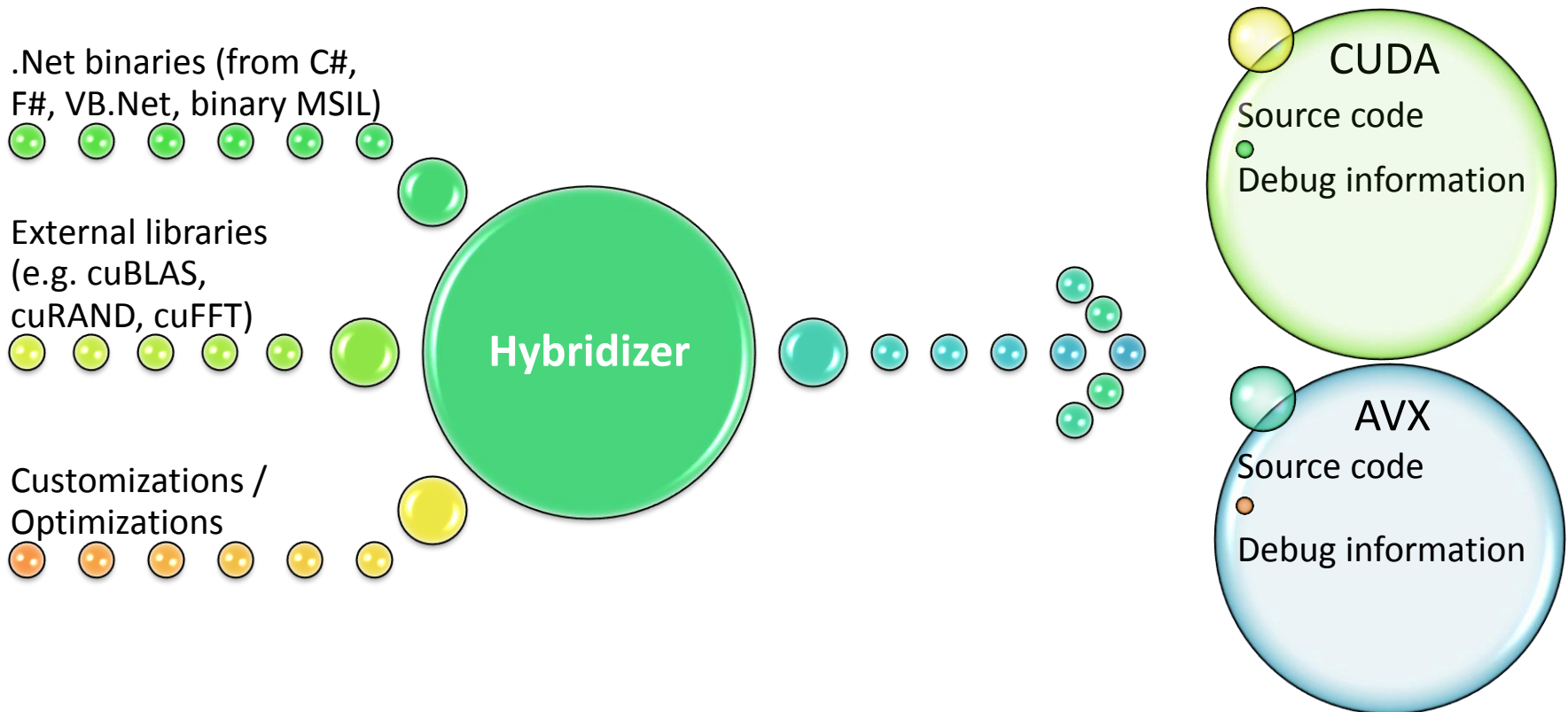


# What the Hybridizer does

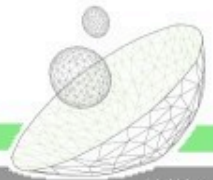
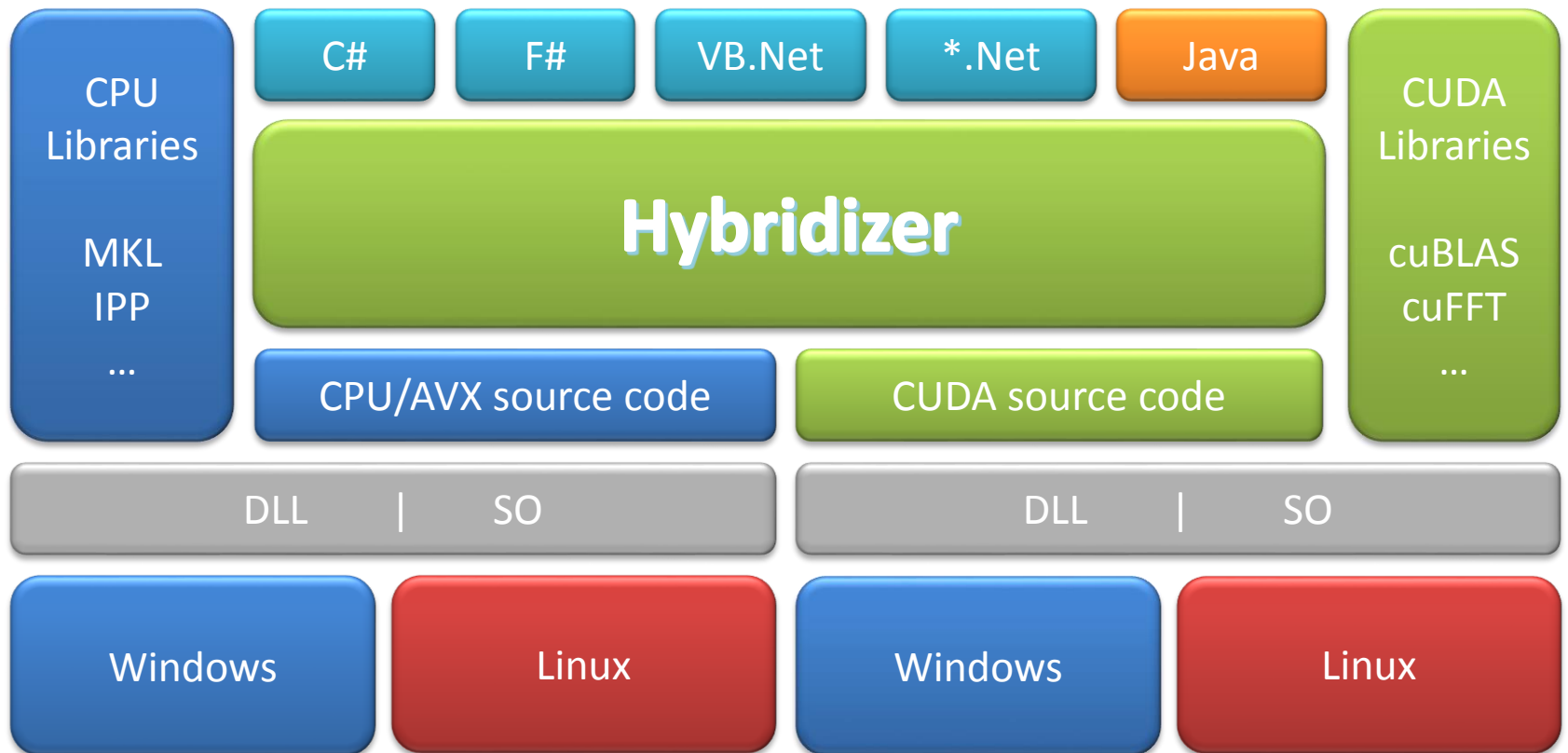
- Generates **source code** from **binaries**
  - Input is dot net binary (C#, VB.Net, Managed C++, other MSIL languages, Java)
  - Output is source code that can be used in various environments (plain C/C++ projects, CUDA projects, Windows/Linux, DotNet / Java runtimes)
- Supports the following language constructs
  - Virtual functions, generic types
  - Use of external libraries with seamless integration (e.g. CUBLAS, CURAND for CUDA environment) – user-extensible
  - Perform debugging within original source code – say C#. (this feature needs pdb)



# What the Hybridizer does



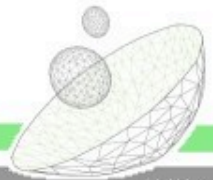
# Software Stack





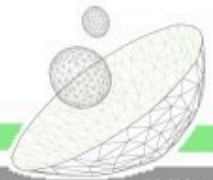
Flexibility of managed environments,  
80%+ usage of hardware

## **HYBRIDIZER IN ACTION**



# Basic features

- CUDA-style work distribution
- Seamless integration (attribute-based)
- Extensibility:
  - Usage of existing functions (erfc, hand-written, ...)
  - Usage of external libraries (cuBLAS, cuRand, ...)
  - Printf available using Console.Out / System.out
  - System.Math maps to <cmath> functions
- Customizable memory management
  - Zero copy arrays
  - Resident array (single copy for multiple kernel calls)



# Performances bandwidth & double precision



KEPLER – K20C



i7-3610 QM - AVX

Compute	GCFLOPS	usage	GFLOPS	usage
whetstone	541	92%	43.2	87%
peak	587	-	49.6	-

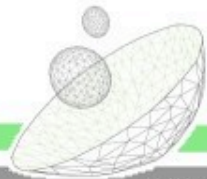
NOTE : Whetstone is our internal naive reproduction of the basic Whetstone test operating on doubles

Memory	GB/s	usage	GB/s	usage
stream	162	78%	20.4	80%
peak	208	-	25.6	-

NOTE : FMA IS COUNTED AS 1 FLOP HENCE REDUCING PEAK TO HALF  
1 CFLOP = 1e9 FMA DP – MEASURES ON K20C

1GB/s = 1e9bytes /s here – MEASURES ON K20C – ECC OFF – CUDA 5.0

CORE i7-3610 QM (HT activated) @ 2.3 GHz  
TurboBoost @ 3.1 GHz (observed using monitor)  
AVX - OpenMP with 8 threads (4 cores)



# Virtual Functions

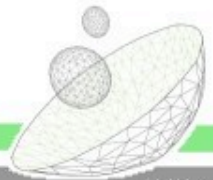
Support for Virtual functions

Function overriding :  
using inheritance

Use of Interfaces  
(single or multiple  
interfaces on classes  
or structs)

Native integration: no  
dedicated code needed.

```
public interface ISimple
{
    int f();
}
public class Answer : ISimple
{
    [Kernel]
    public int f()
    {
        return 42 ;
    }
}
public class Other : ISimple
{
    [Kernel]
    public int f()
    {
        return 12;
    }
}
```



# Performances virtual functions



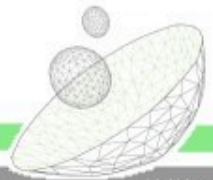
KEPLER – K20C

Expm1 <sup>2</sup>	GFLOPS	GCFLOPS	usage
Local	975	538	92%
Dispatch	478	263	45%
peak	1174	587	-

Virtual functions suffer significant performance penalty

NOTE : FMA IS COUNTED AS 1 FLOP HENCE REDUCING PEAK TO HALF  
1 GCFLOP = 1<sup>e9</sup> FMA DP – MEASURES ON K20C

<sup>2</sup>: EXPM1 IS A TAYLOR EXPANSION OF EXP(X)-1: (1 ADDITION, 13 FUSED MULTIPLY ADD, 2 MULTIPLY)



# Improve performances with Generics

Generics can be  
converted to Templates

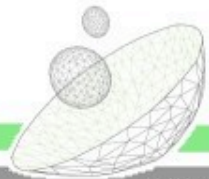
Generic constraints  
lead to usage of  
template functions (no  
virtual call)

Performances are very  
close to performance  
obtained with local  
functions (no  
inheritance/interface)

```
[HybridTemplateConcept]
public interface IMyArray {
    double this[int index] { get; set; }
}

[HybridRegisterTemplate(Specialize=typeof(MyAlgorithm<MyArray>))]
public struct MyArray : IMyArray
{
    double[] _data;
    [Kernel] public double this[int index] {
        get { return _data[index]; }
        set { _data[index] = value; }
    }
}

public class MyAlgorithm<T> where T : struct, IMyArray
{
    T a, b;
    [Kernel] public void Add(int n) {
        for (int k = threadIdx.x + blockDim.x * blockIdx.x;
            k < n; k += blockDim.x * gridDim.x)
            a[k] += b[k];
    }
}
```



# Performances generics



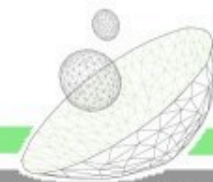
KEPLER – K20C

Expm1 <sup>2</sup>	GFLOPS	GCFLOPS	usage
Local	975	538	92%
Dispatch	478	263	45%
Generics	985	544	93%
peak	1174	587	-

Mapping generics to templates restores performances

NOTE : FMA IS COUNTED AS 1 FLOP HENCE REDUCING PEAK TO HALF  
1 GCFLOP = 1<sup>e9</sup> FMA DP – MEASURES ON K20C

<sup>2</sup>: EXPM1 IS A TAYLOR EXPANSION OF  $\exp(x)-1$ : (1 ADDITION, 13 FUSED MULTIPLY ADD, 2 MULTIPLY)



# Performances single precision



## KEPLER – GTX 680

1536 cores @ 1006 GHz = 1545 GCFLOPS

## MAXWELL – GTX 750Ti

640 cores @ 1.085 GHz = 694.4 GCFLOPS

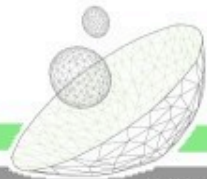
Expm1 <sup>2</sup> benchmark	GCFLOPS	Usage	GCFLOPS	usage
Local	953.6 - 1234	61% - <b>80%</b>	450.8 - 660.0	65% - <b>95%</b>
Dispatch	392.3 - 632.7	<b>25%</b> - 41%	171.0 - 343.2	<b>25%</b> - 49%
Template	958.1 - 1069	62% - <b>69%</b>	440.3 - 539.3	63% - <b>78%</b>
peak	1545	-	694.4	-

without - *with*  
vectorization

without - *with*  
vectorization

NOTE : FMA IS COUNTED AS 1 FLOP HENCE REDUCING PEAK TO HALF : 1 GCFLOP = 1<sup>e9</sup> FMA SP

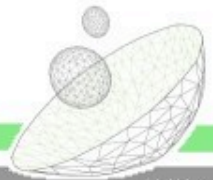
<sup>2</sup>: EXPM1 IS A TAYLOR EXPANSION OF EXP(X)-1: (1 ADDITION, 13 FUSED MULTIPLY ADD, 2 MULTIPLY)





Developers perspective

## **INTEGRATION WITH VISUAL STUDIO**



# Debugging session using NSIGHT for Visual Studio [2010]

The screenshot displays the Visual Studio IDE with the following components:

- MapReduceSample.cs**: A C# file containing a `Discount` class that implements `IMapOperator`. It includes a `F(int i)` method with a `[Kernel]` attribute. The method calculates a result based on `rates` and `dates` arrays.
- Espion 1**: A Watch window showing the state of variables during execution. The variables and their values are:

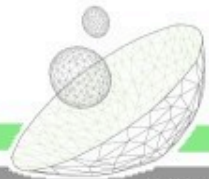
Nom	Valeur	Type
rate	0.0261487155552715	double
pself->rates[i]	0.0261487155552715	__device__ double&
pself->nbSims	0x00011800	__device__ int
threadIdx	{x = 0x00000000, y = 0x00000000, z = 0x00000000}	const uint3
x	0x00000000	unsigned int
y	0x00000000	unsigned int
z	0x00000000	unsigned int
- CUDA Info 1**: A table showing the execution status of GPU threads. The first five threads are in a 'Breakpoint' status.

Current	Frozen	CUcontext	Grid ID	blockIdx	Warp Index	threadIdx	PC	Active Mask	Status	Exception	Exception Details	Global Status Details
➔		0x067b3f70	6	( 0, 0, 0)	0	( 0, 0, 0)	0x00050bf8	0xffffffff	Breakpoint	None	None	None
		0x067b3f70	6	( 0, 0, 0)	1	( 32, 0, 0)	0x00050bf8	0xffffffff	Breakpoint	None	None	None
		0x067b3f70	6	( 0, 0, 0)	2	( 64, 0, 0)	0x00050bf8	0xffffffff	Breakpoint	None	None	None
		0x067b3f70	6	( 0, 0, 0)	3	( 96, 0, 0)	0x00050bf8	0xffffffff	Breakpoint	None	None	None
		0x067b3f70	6	( 0, 0, 0)	4	(128, 0, 0)	0x00050bf8	0xffffffff	Breakpoint	None	None	None

Breakpoint  
is set and  
hit in C#  
code

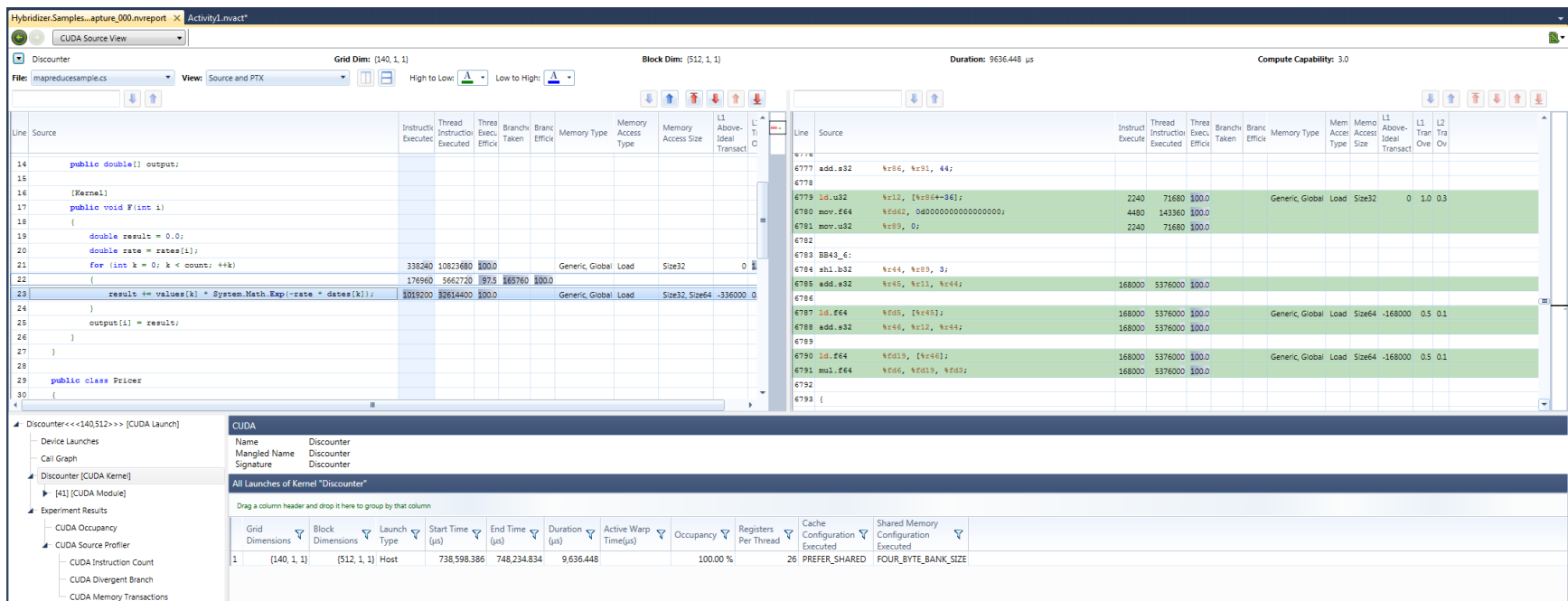
Values can  
be explored  
using Watch

Execution  
is on GPU



# Profiling session using NSIGHT for Visual Studio [2010]

Compilation with line-info allows dot net source-level  
profiling (also in release mode)



The screenshot displays the NSIGHT for Visual Studio 2010 interface, showing a CUDA profiling session. The top pane displays the source code of a kernel, with line numbers 14 to 30. The middle pane shows the assembly code for the kernel, with instructions and their corresponding metrics. The bottom pane shows the 'All Launches of Kernel' table, which lists the launch configuration and performance metrics for the kernel.

**Source Code (Left Pane):**

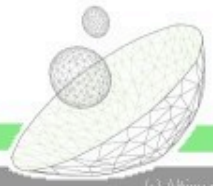
```
14 public double[] output;  
15 [Kernel]  
16 public void F(int i)  
17 {  
18     double result = 0.0;  
19     double rate = rates[i];  
20     for (int k = 0; k < count; ++k)  
21     {  
22         result += values[k] * System.Math.Exp(-rate * dates[k]);  
23     }  
24     output[i] = result;  
25 }  
26 }  
27 }  
28 }  
29 public class Prices  
30 {
```

**Assembly Code (Middle Pane):**

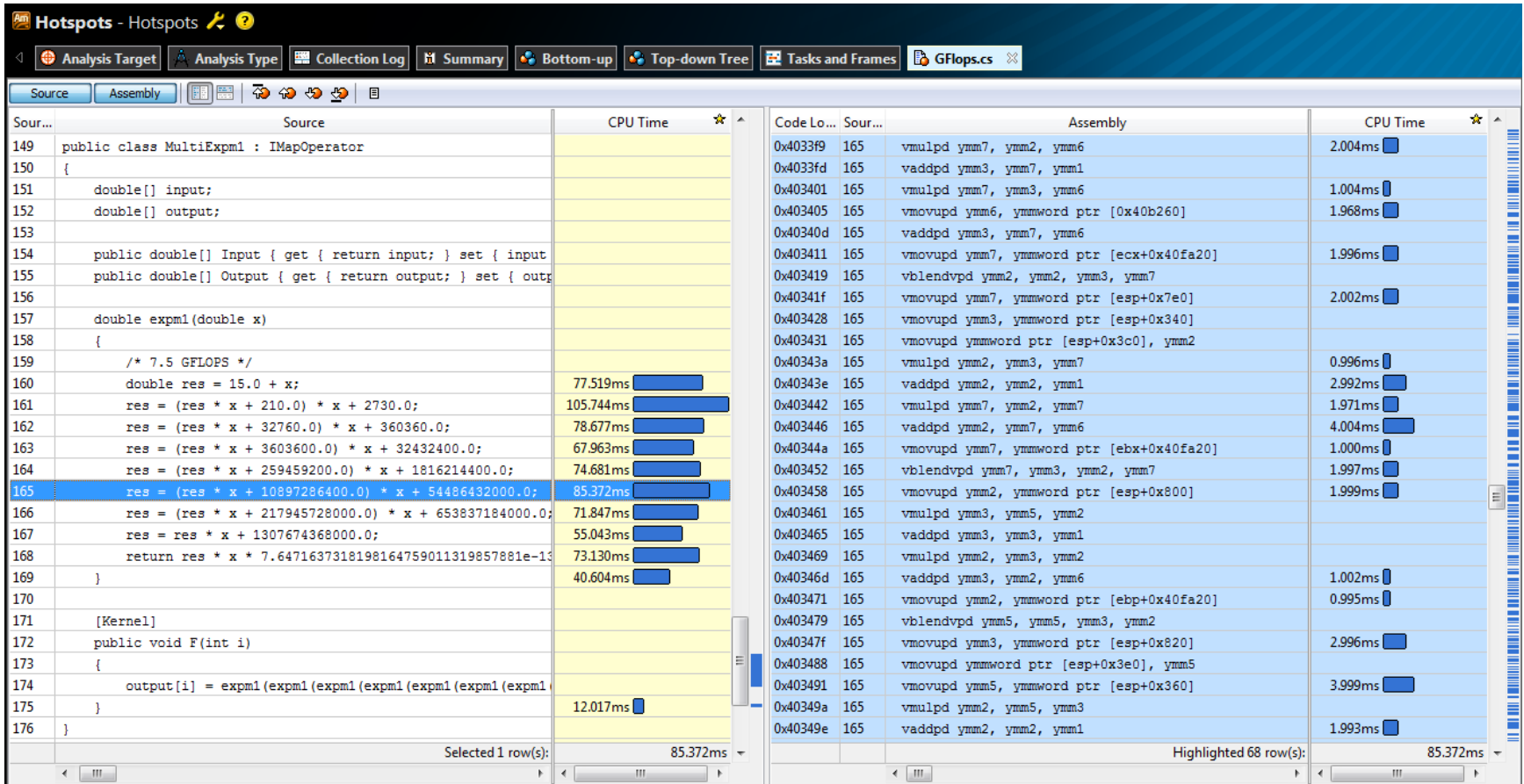
```
6777 add.s32 %r86, %r91, 44;  
6778  
6779 ld.u32 %r12, [%r86+36];  
6780 mov.f64 %fd62, 0a00000000000000;  
6781 mov.u32 %r89, 0;  
6782  
6783 BB43_6:  
6784 shl.b32 %r44, %r89, 3;  
6785 add.s32 %r45, %r11, %r44;  
6786  
6787 ld.f64 %fd5, [%r45];  
6788 add.s32 %r46, %r12, %r44;  
6789  
6790 ld.f64 %fd19, [%r46];  
6791 mul.f64 %fd6, %fd19, %fd3;  
6792  
6793 {
```

**All Launches of Kernel "Discounter" (Bottom Pane):**

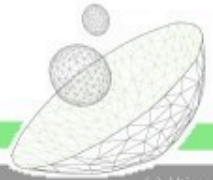
Grid Dimensions	Block Dimensions	Launch Type	Start Time (µs)	End Time (µs)	Duration (µs)	Active Warp Time (µs)	Occupancy	Registers Per Thread	Cache Configuration Executed	Shared Memory Configuration Executed
1	[140, 1, 1]	[512, 1, 1] Host	738,598,386	748,234,834	9,636,448		100.00 %	26	PREFER_SHARED	FOUR_BYTE_BANK_SIZE



# Profiling session using VTune Amplifier for Visual Studio [2010]



See line association between original sequential C# code and vectorized x86/AVX assembly instructions



# Usages – runtimes – execution environments

Generated From



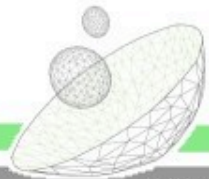
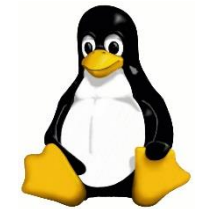
Target



Use From



Run on



“We have been using the Hybridizer for more than a year now with very satisfactory results. With **no prior knowledge of GPU programming**, we have been able to achieve **significant speedups in a large scale application** with unexcessive effort. Hybridizer enabled **rapid integration of GPU** within our development environment, with limited impact on a team of hundred programmers. It took **nine months to a handful of developers to go from early testing to production** on our first perimeter, and six more months to cover some of our most compute intensive calculations.”

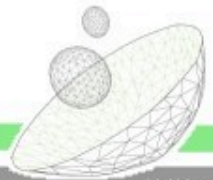
***Régis FRICKER - GPU project leader at Société Générale Investment Banking***



[Florent.Duguet@altimesh.com](mailto:Florent.Duguet@altimesh.com)

[Guillaume.de-Roujoux@altimesh.com](mailto:Guillaume.de-Roujoux@altimesh.com)

**THANK YOU**



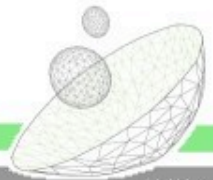
# Basics

C-style programming of functions

Use of CUDA-like constructs for work distribution

Operate on native types using standard functions

```
[EntryPoint("TestSquare")]  
public void square(int count,  
                   double[] a, double[] b)  
{  
    for (int k = threadIdx.x +  
           blockDim.x * blockIdx.x ;  
         k < count ;  
         k += blockDim.x * gridDim.x)  
    {  
        b[k] = a[k] * a[k];  
    }  
}
```





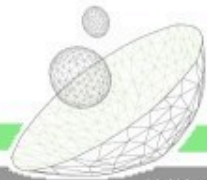
# Extensibility

Use existing functions  
from native  
developments

Use CUDA device  
functions math  
functions, and any  
custom function

```
[IntrinsicFunction("erf")]  
public double Erf(double x)  
{  
    ...  
}
```

```
[EntryPoint("TestErfc")]  
public void errorFunction(int count,  
    double[] a, double[] b)  
{  
    for (int k = threadIdx.x +  
        blockDim.x * blockIdx.x;  
        k < count ;  
        k += blockDim.x * gridDim.x)  
    {  
        b[k] = Erf (a[k]);  
    }  
}
```



# Libraries integration

Use device library  
functions such as

cuBLAS

cuRAND

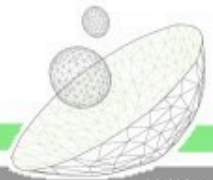
...

```
[IntrinsicType("curandStateMRG32k3a_t")]
[IntrinsicIncludeCUDA("curand_kernel.h")]
[StructLayout(LayoutKind.Sequential)]
public unsafe struct curandStateMRG32k3a_t
{
    public fixed double s1[3];
    public fixed double s2[3];
    public int boxmuller_flag;
    public int boxmuller_flag_double;
    public float boxmuller_extra;
    public double boxmuller_extra_double;

    [IntrinsicFunction("curand_init")]
    public static void curand_init(ulong seed,
        ulong subsequence, ulong offset,
        out curandStateMRG32k3a_t state)
    { throw new NotImplementedException(); }

    [IntrinsicFunction("curand")]
    public uint curand()
    { throw new NotImplementedException(); }

    [IntrinsicFunction("curand_log_normal")]
    public float curand_log_normal(float mean,
        float stdev)
    { throw new NotImplementedException(); }
```



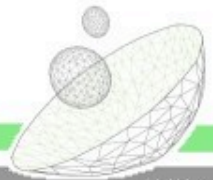
# Printf and Builtins

Seamless printf with  
Console.Out.Write()

External functions can  
be overridden for  
customization  
(builtins)

In this example,  
System.Math.Exp is  
translated into exp  
from <cmath>

```
[EntryPoint("TestPrintf")]  
public void testPrintf()  
{  
    Console.Out.WriteLine(  
        "Comment from Thread {0} Block {1}",  
        threadIdx.x, blockIdx.x);  
}  
  
[EntryPoint("TestExp")]  
public void TestExp()  
{  
    exp = System.Math.Exp(1.0);  
}
```



# Serialization, Resident Data

Managed/GC objects are serialized into native format to GPU or CPU memory

Serialization groups objects for improved performances (fewer malloc/memcpy)

Data can be flagged as resident, and remain on GPU across calls

Resident Data is customizable

