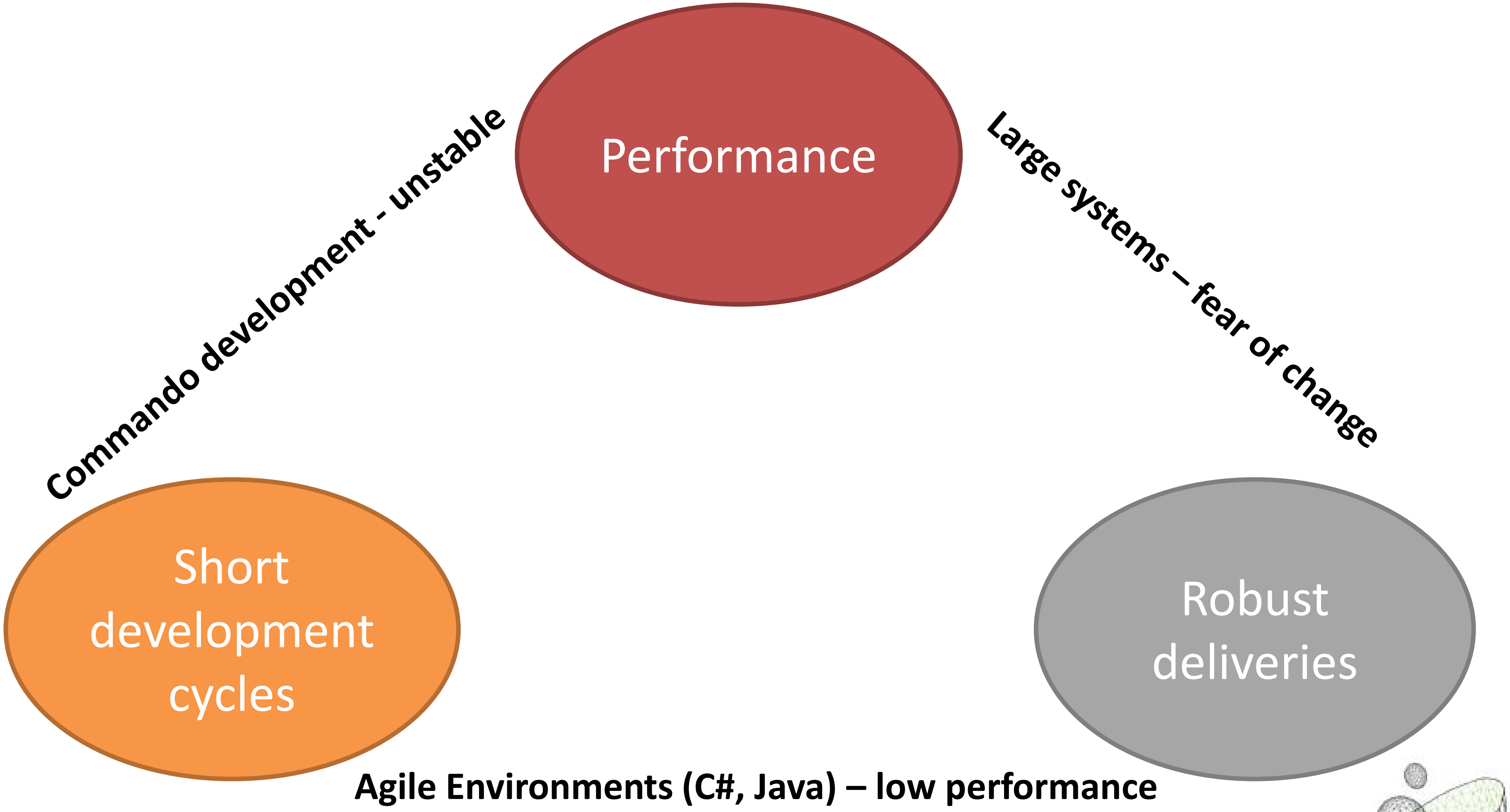
Altimesh Hybridizer™

Enabling Accelerators in .Net and more

So many platforms, so few experts…

WHY THE HYBRIDIZER ?

Software development teams accommodate external constraints



Performance

Short development cycles

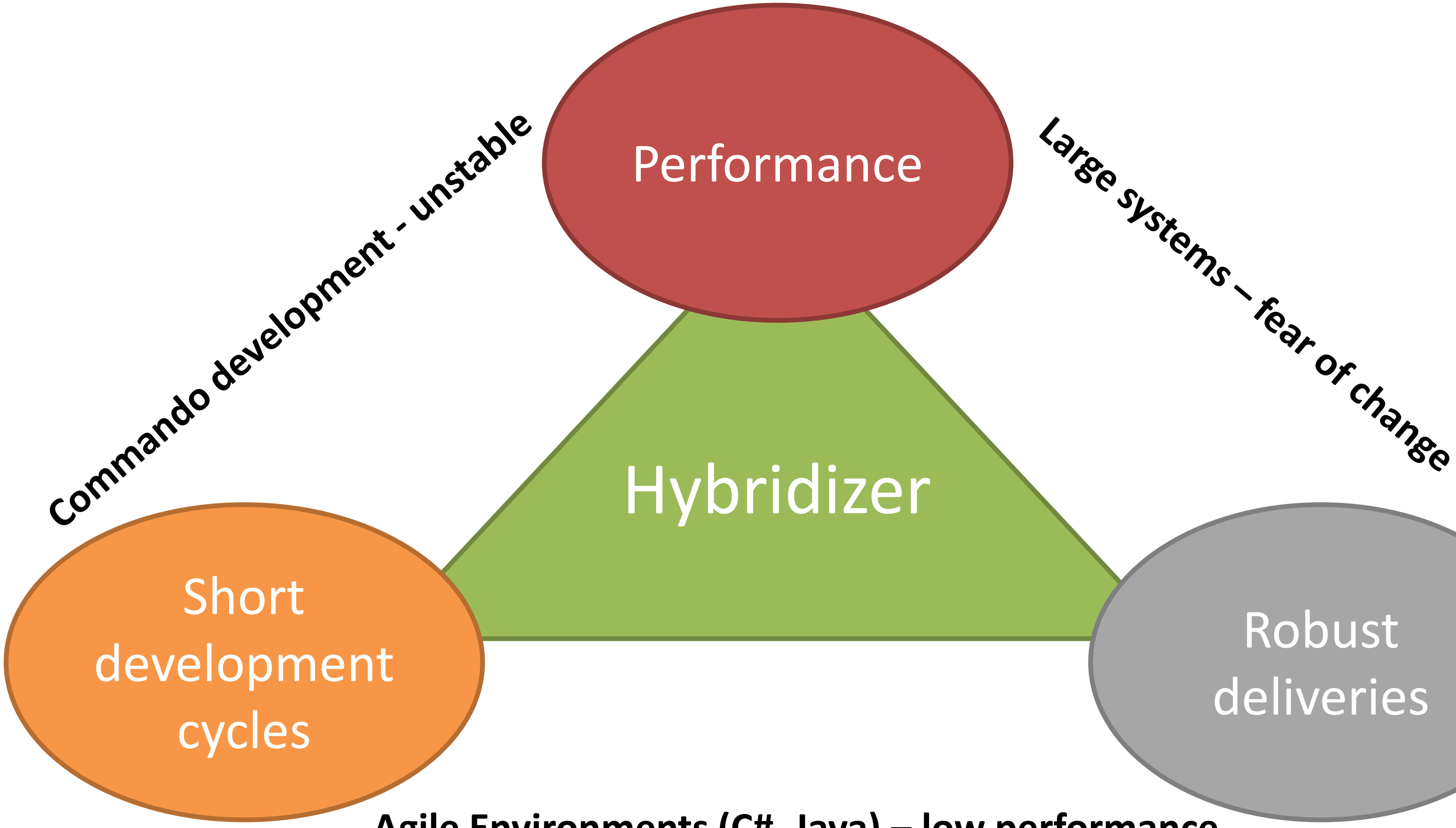
Robust deliveries

Commando development - unstable

Large systems – fear of change

Agile Environments (C#, Java) – low performance

Software development teams accommodate external constraints



Commando development - unstable

Large systems – fear of change

Agile Environments (C#, Java) – low performance

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

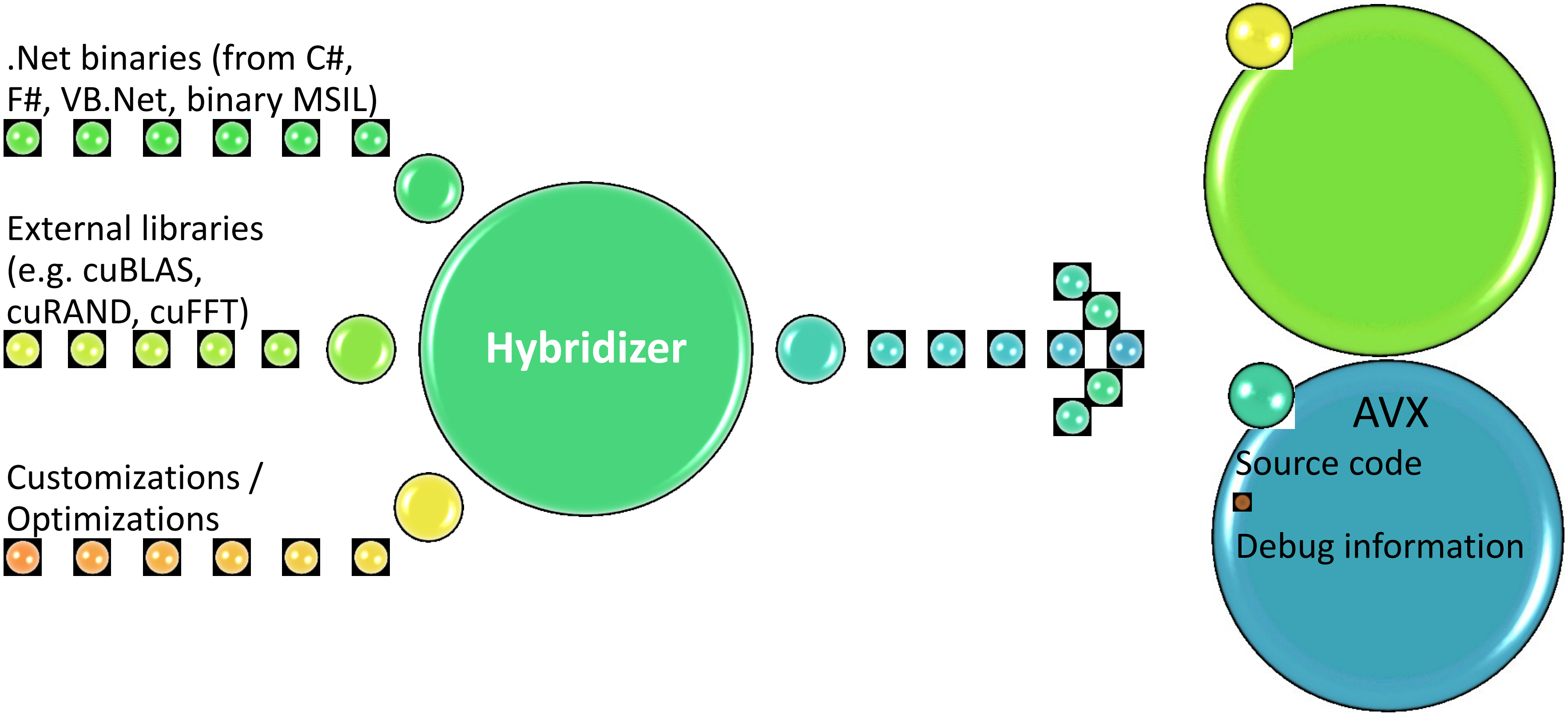
- Work distribution needs to be explicit
  - Loop parallelization is not automatic
  - Concurrency needs to be handled by hand
  - Code patterns need to be changed from sequential to parallel

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



.Net binaries (from C#, F#, VB.Net, binary MSIL)

External libraries (e.g. cuBLAS, cuRAND, cuFFT)

Customizations / Optimizations

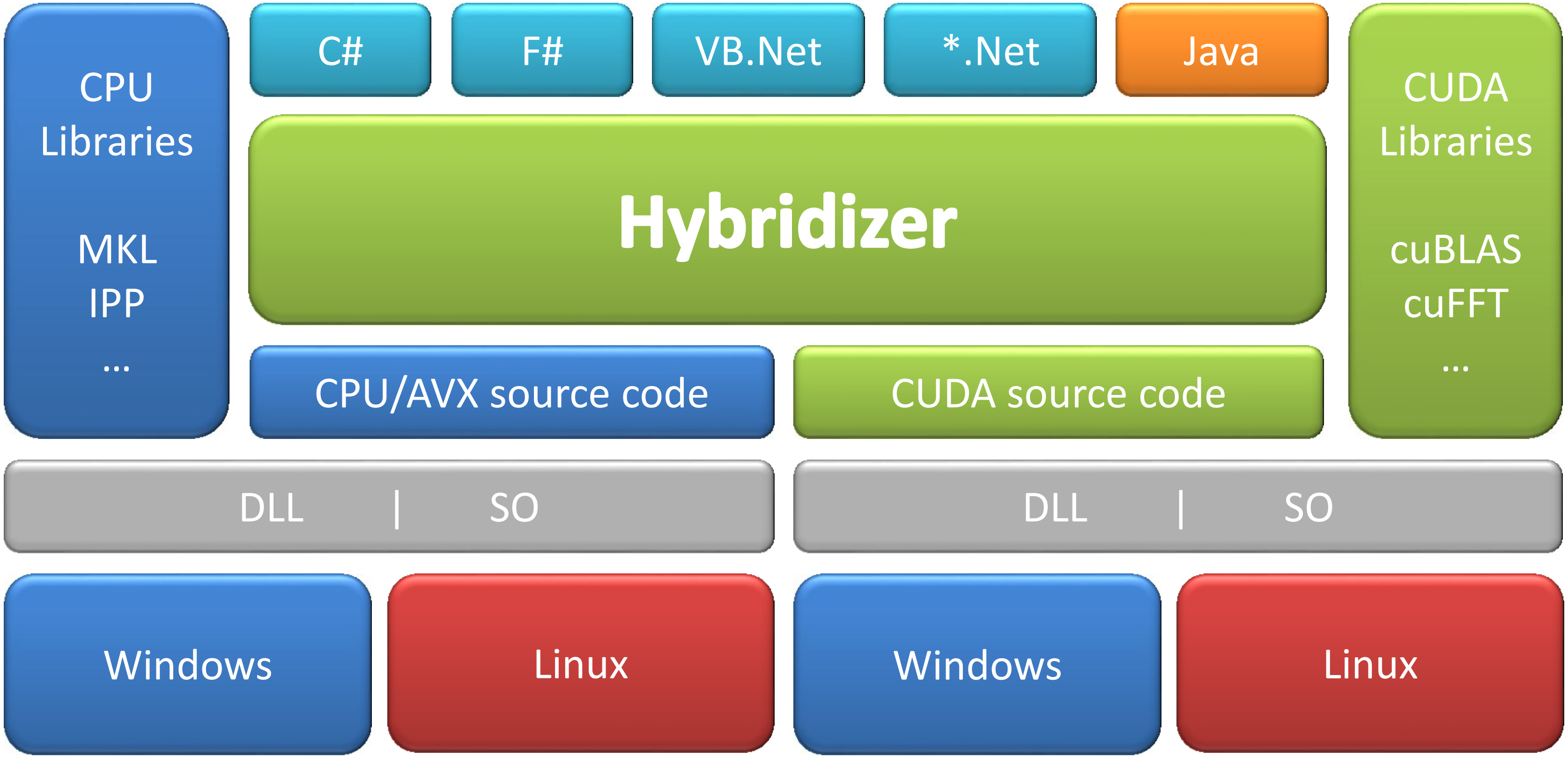
Hybridizer

AVX

Source code

Debug information

Software Stack



CPU Libraries

MKL

IPP

…

C#

F#

VB.Net

*.Net

Java

CUDA Libraries

cuBLAS

cuFFT

…

Hybridizer

CPU/AVX source code

CUDA source code

DLL | SO

DLL | SO

Windows

Linux

Windows

Linux

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

	KEPLER – K20C		i7-3610 QM - AVX	
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 = 1^{e}9 FMA DP – MEASURES ON K20C

1GB/s = 1^{e}9bytes /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[2]	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^{e}9 FMA DP – MEASURES ON K20C

[2]: 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[2]	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^{e}9 FMA DP – MEASURES ON K20C

[2]: 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[2] benchmark	GCFLOPS	Usage	GCFLOPS	usage
Local	953.6 - 1234	61% - **80%**	450.8 - 660.0	65% - **95%**
Dispatch	392.3 - 632.7	**25%** - 41%	171.0 - 343.2	**25%** - 49%
Template	958.1 - 1069	62% - **69%**	440.3 - 539.3	63% - **78%**
peak	1545	-	694.4	-

without - *with* vectorization

without - *with* vectorization

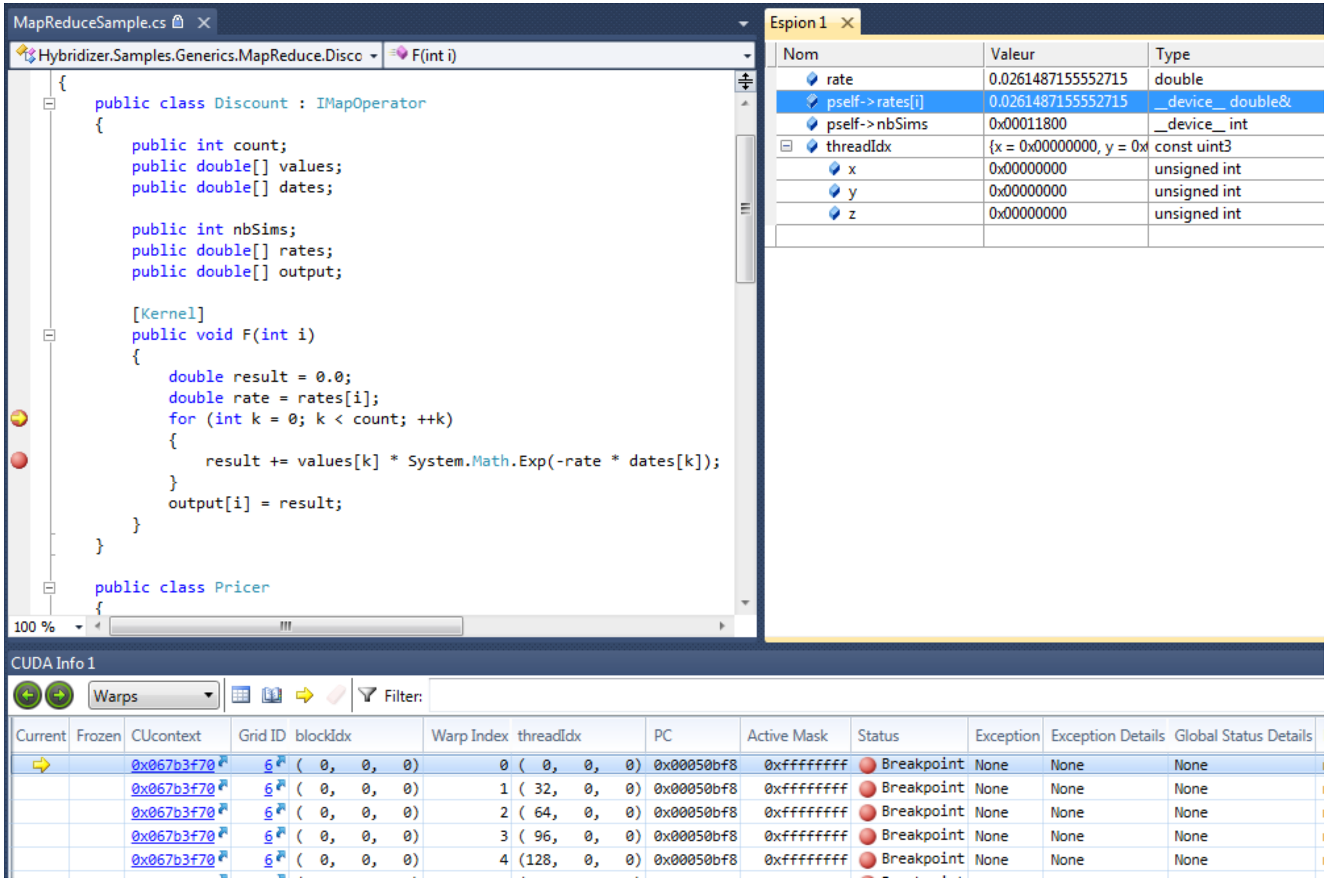
NOTE : FMA IS COUNTED AS 1 FLOP HENCE REDUCING PEAK TO HALF : 1 GCFLOP = 1^{e}9 FMA SP

[2]: 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]



```
{
    public class Discount : IMapOperator
    {
        public int count;
        public double[] values;
        public double[] dates;

        public int nbSims;
        public double[] rates;
        public double[] output;

        [Kernel]
        public void F(int i)
        {
            double result = 0.0;
            double rate = rates[i];
            for (int k = 0; k < count; ++k)
            {
                result += values[k] * System.Math.Exp(-rate * dates[k]);
            }
            output[i] = result;
        }
    }

    public class Pricer
    {
```

Nom	Valeur	Type
rate	0.0261487155552715	double
pself->rates[i]	0.0261487155552715	__device__ double&
pself->nbSims	0x00011800	__device__ int
threadIdx	{x = 0x00000000, y = 0x	const uint3
x	0x00000000	unsigned int
y	0x00000000	unsigned int
z	0x00000000	unsigned int

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

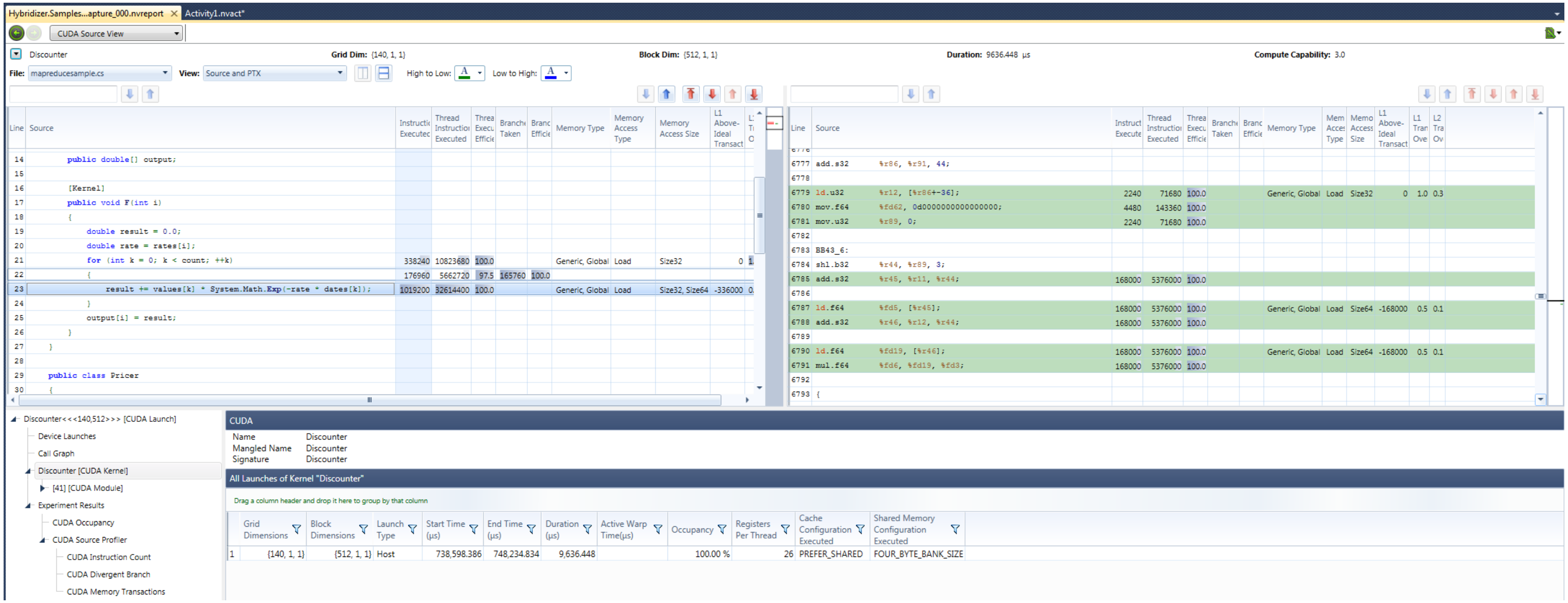
Breakpoint is set and hit in C# code

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)



Discounter **Grid Dim:** {140, 1, 1} **Block Dim:** {512, 1, 1} **Duration:** 9636.448 µs **Compute Capability:** 3.0

File: mapreducesample.cs **View:** Source and PTX High to Low: A Low to High: A

Line	Source	Instructic Executec	Thread Instructior Executed	Threa Execu Efficie	Branche Taken	Branc Efficie	Memory Type	Memory Access Type	Memory Access Size	L1 Above-Ideal Transact	L: Ti O
14	`public double[] output;`										
15											
16	`[Kernel]`										
17	`public void F(int i)`										
18	`{`										
19	`double result = 0.0;`										
20	`double rate = rates[i];`										
21	`for (int k = 0; k < count; ++k)`	338240	10823680	100.0			Generic, Global	Load	Size32	0	1.
22	`{`	176960	5662720	97.5	165760	100.0					
23	`result += values[k] * System.Math.Exp(-rate * dates[k]);`	1019200	32614400	100.0			Generic, Global	Load	Size32, Size64	-336000	0.
24	`}`										
25	`output[i] = result;`										
26	`}`										
27	`}`										
28											
29	`public class Pricer`										
30	`{`										

Line	Source	Instruct Execute	Thread Instructior Executed	Threa Execu Efficie	Branche Taken	Branc Efficie	Memory Type	Mem Acces Type	Memo Access Size	L1 Above-Ideal Transact	L1 Tran Ove	L2 Tra Ov
6777	`add.s32 %r86, %r91, 44;`											
6778												
6779	`ld.u32 %r12, [%r86+-36];`	2240	71680	100.0			Generic, Global	Load	Size32	0	1.0	0.3
6780	`mov.f64 %fd62, 0d0000000000000000;`	4480	143360	100.0								
6781	`mov.u32 %r89, 0;`	2240	71680	100.0								
6782												
6783	`BB43_6:`											
6784	`shl.b32 %r44, %r89, 3;`											
6785	`add.s32 %r45, %r11, %r44;`	168000	5376000	100.0								
6786												
6787	`ld.f64 %fd5, [%r45];`	168000	5376000	100.0			Generic, Global	Load	Size64	-168000	0.5	0.1
6788	`add.s32 %r46, %r12, %r44;`	168000	5376000	100.0								
6789												
6790	`ld.f64 %fd19, [%r46];`	168000	5376000	100.0			Generic, Global	Load	Size64	-168000	0.5	0.1
6791	`mul.f64 %fd6, %fd19, %fd3;`	168000	5376000	100.0								
6792												
6793	`{`											

- Discounter<<<140,512>>> [CUDA Launch]
  - Device Launches
  - Call Graph
  - Discounter [CUDA Kernel]
    - [41] [CUDA Module]
  - Experiment Results
    - CUDA Occupancy
    - CUDA Source Profiler
      - CUDA Instruction Count
      - CUDA Divergent Branch
      - CUDA Memory Transactions

CUDA

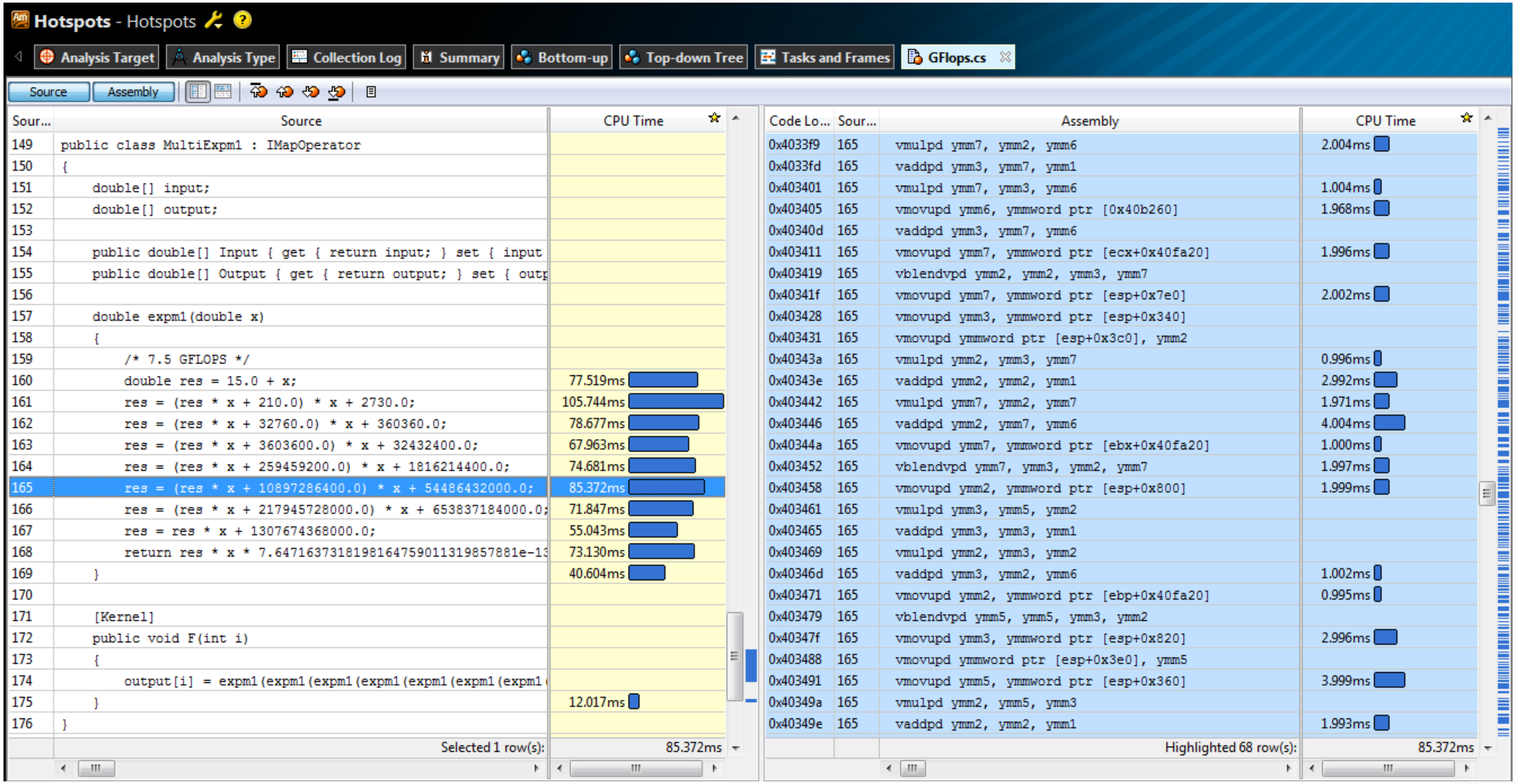
Name Discounter
Mangled Name Discounter
Signature Discounter

All Launches of Kernel "Discounter"

Drag a column header and drop it here to group by that column

	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]



Sour...	Source	CPU Time
149	public class MultiExpm1 : IMapOperator	
150	{	
151	double[] input;	
152	double[] output;	
153		
154	public double[] Input { get { return input; } set { input	
155	public double[] Output { get { return output; } set { outp	
156		
157	double expm1(double x)	
158	{	
159	/* 7.5 GFLOPS */	
160	double res = 15.0 + x;	77.519ms
161	res = (res * x + 210.0) * x + 2730.0;	105.744ms
162	res = (res * x + 32760.0) * x + 360360.0;	78.677ms
163	res = (res * x + 3603600.0) * x + 32432400.0;	67.963ms
164	res = (res * x + 259459200.0) * x + 1816214400.0;	74.681ms
165	res = (res * x + 10897286400.0) * x + 54486432000.0;	85.372ms
166	res = (res * x + 217945728000.0) * x + 653837184000.0;	71.847ms
167	res = res * x + 1307674368000.0;	55.043ms
168	return res * x * 7.6471637318198164759011319857881e-13	73.130ms
169	}	40.604ms
170		
171	[Kernel]	
172	public void F(int i)	
173	{	
174	output[i] = expm1(expm1(expm1(expm1(expm1(expm1(expm1	
175	}	12.017ms
176	}	
	Selected 1 row(s):	85.372ms

Code Lo...	Sour...	Assembly	CPU Time
0x4033f9	165	vmulpd ymm7, ymm2, ymm6	2.004ms
0x4033fd	165	vaddpd ymm3, ymm7, ymm1	
0x403401	165	vmulpd ymm7, ymm3, ymm6	1.004ms
0x403405	165	vmovupd ymm6, ymmword ptr [0x40b260]	1.968ms
0x40340d	165	vaddpd ymm3, ymm7, ymm6	
0x403411	165	vmovupd ymm7, ymmword ptr [ecx+0x40fa20]	1.996ms
0x403419	165	vblendvpd ymm2, ymm2, ymm3, ymm7	
0x40341f	165	vmovupd ymm7, ymmword ptr [esp+0x7e0]	2.002ms
0x403428	165	vmovupd ymm3, ymmword ptr [esp+0x340]	
0x403431	165	vmovupd ymmword ptr [esp+0x3c0], ymm2	
0x40343a	165	vmulpd ymm2, ymm3, ymm7	0.996ms
0x40343e	165	vaddpd ymm2, ymm2, ymm1	2.992ms
0x403442	165	vmulpd ymm7, ymm2, ymm7	1.971ms
0x403446	165	vaddpd ymm2, ymm7, ymm6	4.004ms
0x40344a	165	vmovupd ymm7, ymmword ptr [ebx+0x40fa20]	1.000ms
0x403452	165	vblendvpd ymm7, ymm3, ymm2, ymm7	1.997ms
0x403458	165	vmovupd ymm2, ymmword ptr [esp+0x800]	1.999ms
0x403461	165	vmulpd ymm3, ymm5, ymm2	
0x403465	165	vaddpd ymm3, ymm3, ymm1	
0x403469	165	vmulpd ymm2, ymm3, ymm2	
0x40346d	165	vaddpd ymm3, ymm2, ymm6	1.002ms
0x403471	165	vmovupd ymm2, ymmword ptr [ebp+0x40fa20]	0.995ms
0x403479	165	vblendvpd ymm5, ymm5, ymm3, ymm2	
0x40347f	165	vmovupd ymm3, ymmword ptr [esp+0x820]	2.996ms
0x403488	165	vmovupd ymmword ptr [esp+0x3e0], ymm5	
0x403491	165	vmovupd ymm5, ymmword ptr [esp+0x360]	3.999ms
0x40349a	165	vmulpd ymm2, ymm5, ymm3	
0x40349e	165	vaddpd ymm2, ymm2, ymm1	1.993ms
		Highlighted 68 row(s):	85.372ms

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

Usages – runtimes – execution environments

Generated From

Microsoft .net

Java

Target

NVIDIA CUDA

Intel AVX

Use From

Java

Microsoft .net

C

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

Guillaume.de-Roujoux@altimesh.com

THANK YOU