Dynamic Cuda with F#

GTC 2013

March 21

San Jose, California

Dr. Daniel Egloff
Xiang Zhang
+41 44 520 01 17
+41 79 430 03 61

Today

QuantAlea

- Build awareness for F# and CUDA with F#
- Briefly present what we have achieved so far
- Show the potential of F# as a future core language for CUDA
- Do some live coding examples including CUDA scripting in Excel
- Point you to more resources and free licenses

About Us

QuantAlea

- Software development and consulting company
- Based in Zurich
- Core competence
  - Quantitative finance and risk management
  - Derivative pricing and modeling
  - Numerical computing
  - High performance computing (clusters, grid, GPUs)
  - Software engineering (C++, F#, Scala, …)
- Early adopter of GPUs
  - First project with GPUs in finance back in 2007

Why should you care about F#?

QuantAlea

- F# is now a first class CUDA language

- F# is a strongly typed functional first language
- Design goals of the F# language
  - High productivity
  - Development of robust and correct code
  - Efficiency
- F# is highly flexible and extensible
  - Monads, type providers
  - DSLs
- F# is ideal for numerical computing and therefore for CUDA
- F# has potential to play an important role in computational finance and big data
- Vast .NET ecosystem

What is

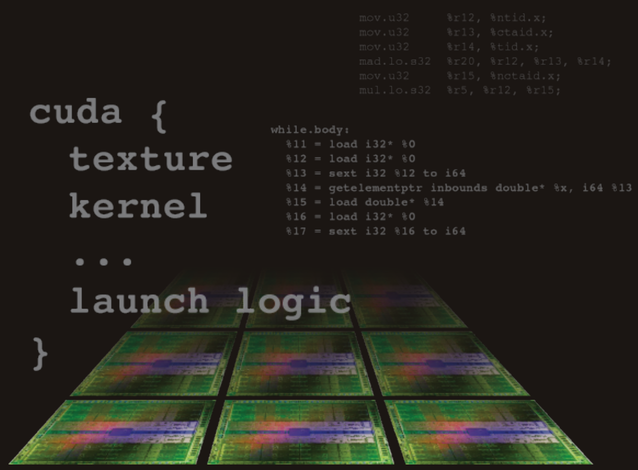
Alea.cuBase

?

Alea.cuBase

QuantAlea

- Alea.cuBase extends F# to a first class CUDA language
  - Based on LLVM and CUDA 5 technology
  - Noninvasive single language solution for host and GPU programming
  - No additional language additions required, in particular no <<<…>>>
  - Extensible
  - Basis for creating higher level GPU aware DSLs



Dynamic code generation

GPU algorithm scripting

Industry grade performance

Rapid development

Solid framework for reusability

Advanced CUDA programming

Benefits

QuantAlea

Dynamic code generation

- Generate GPU code programmatically at run-time
- Use .NET generics and F# code quotation splicing for flexible kernels
- Foundation to develop GPU aware domain specific languages

```
let init = <@ fun () -> 0.0 @>
```

```
let op = <@ (+) @>
```

```
let transf = <@ fun x -> x*x @>
```

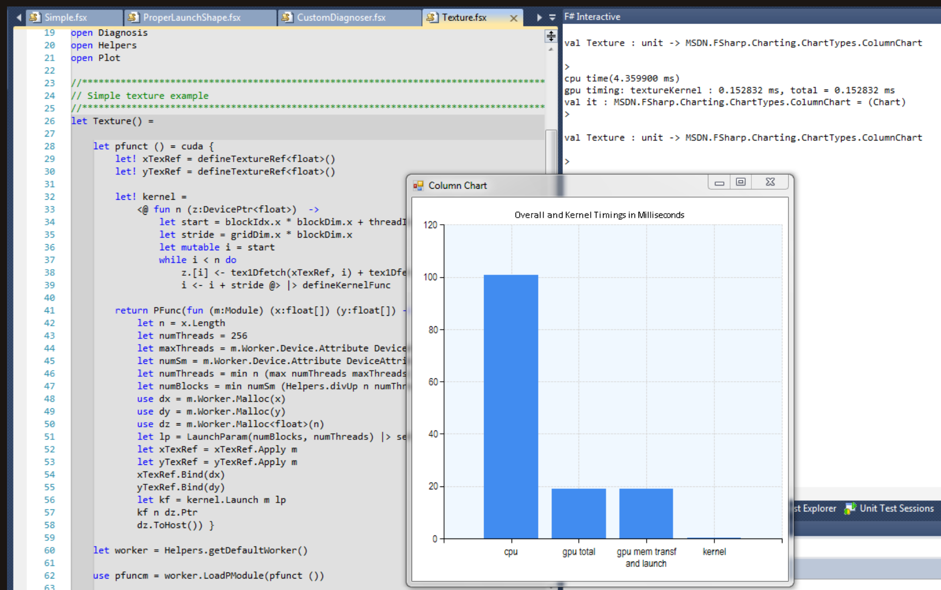
```
let inline reduceKernel (initExpr:Expr<unit -> 'T>)
                        (opExpr:Expr<'T -> 'T -> 'T>)
                        (transfExpr:Expr<'T -> 'T>)
                        blockSize nIsPow2 =
    <@ fun level n (inp:DevicePtr<'T>) (outp:DevicePtr
        let init = %initExpr
        let op = %opExpr
        let transf = %transfExpr
```

Benefits

QuantAlea

Rapid development

- Easy and quick setup of development environment, no need to install NVIDIA nvcc compiler tools
- Rapid prototyping in F# interactive
- Iteratively improve CUDA kernel algorithms without time consuming build cycles
- Simple deployment

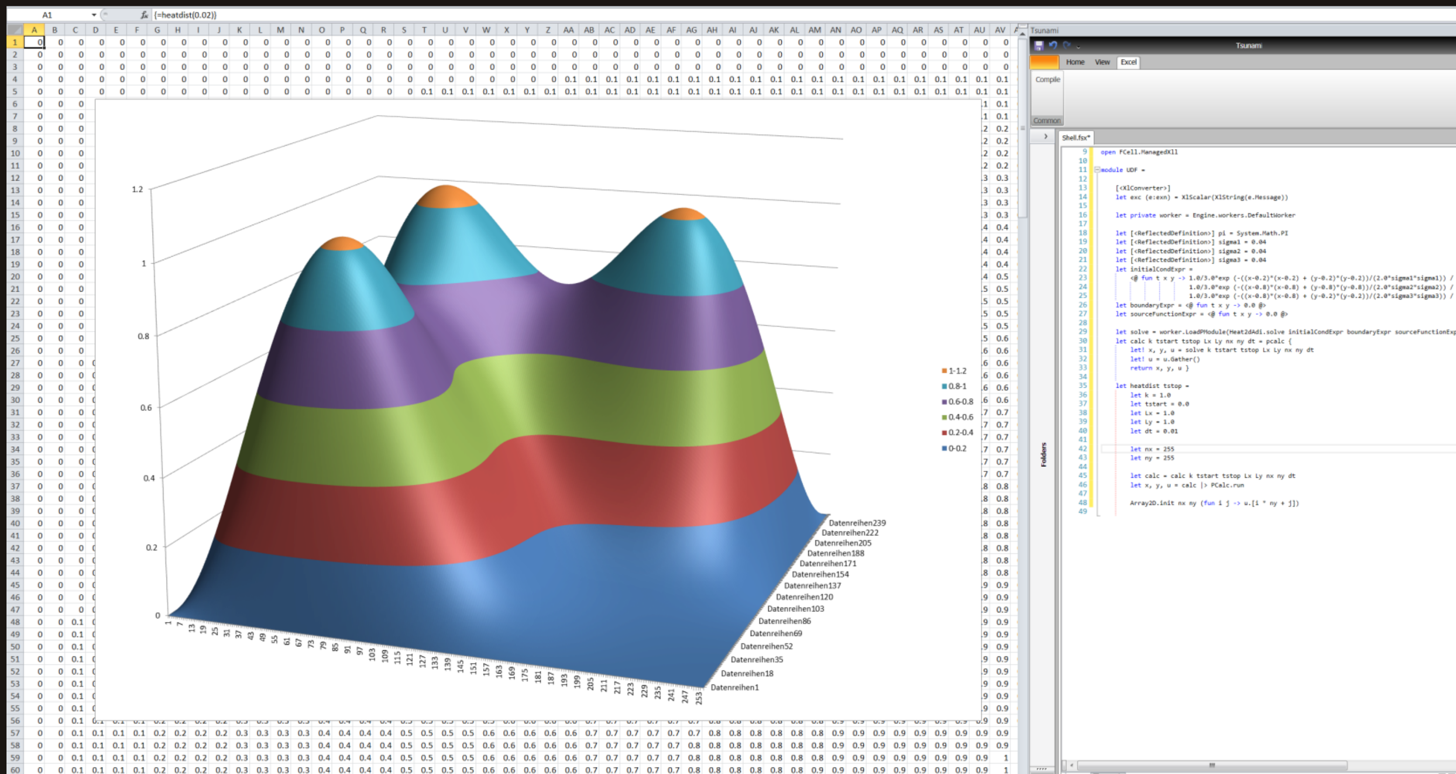


Benefits

QuantAlea

GPU algorithm scripting

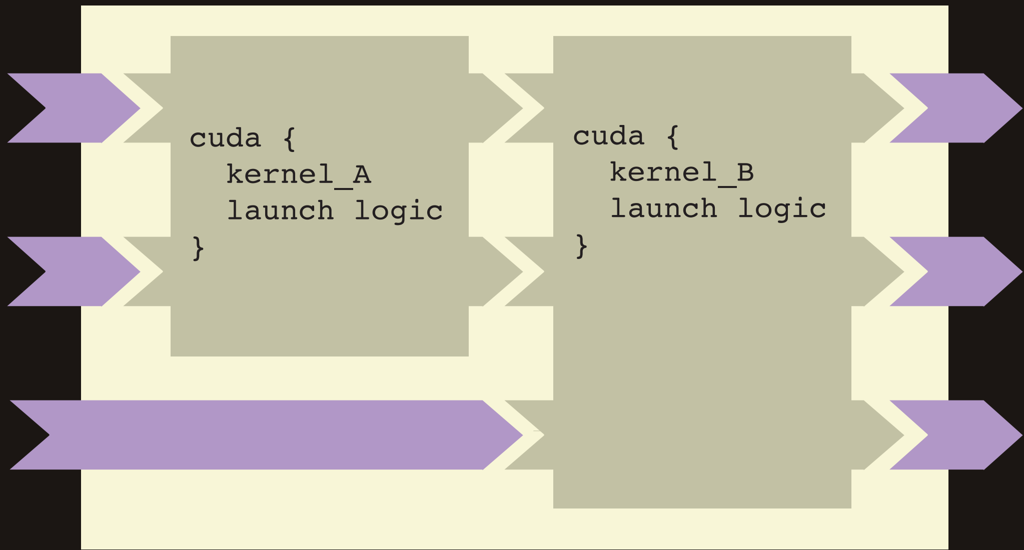
- Execute F# scripts with GPU algorithms on command line or in F# interactive
- GPU scripting in Excel
- Integrate Alea.cuBase directly with Python



Benefits

Solid framework for reusability

- Framework for type-safe definition of GPU resources
- CUDA monad to specify GPU resources together with launch logic in unified manner
- Reuse GPU kernel code and compose them to modular GPU kernel libraries



Benefits

QuantAlea

Industry grade performance

- Generating performance optimized code which is on par with compiled CUDA C/C++ code
- Low level device functions and special math functions
- Built in occupancy calculator to identify optimal thread block layout

```
define void @__kernel_0(i
entry:
  %0 = alloca i32
  %1 = call i32 @llvm.nvv
  %2 = call i32 @llvm.nvv
  %3 = mul i32 %1, %2
  %4 = call i32 @llvm.nvv
  %5 = add i32 %3, %4
  %6 = call i32 @llvm.nvv
  %7 = call i32 @llvm.nvv
  %8 = mul i32 %6, %7
  store i32 %5, i32* %0
  br label %while.cond

while.cond:
  %9 = load i32* %0
  %10 = icmp slt i32 %9,
  br i1 %10, label %while

while.body:
  %11 = load i32* %0
  %12 = load i32* %0
  %13 = sext i32 %12 to i
  %14 = getelementptr inb
  %15 = load double* %14
  %16 = load i32* %0
  %17 = sext i32 %16 to i
  %18 = getelementptr inb
  %19 = load double* %18
  %20 = fadd double %15,
  %21 = sext i32 %11 to i
  %22 = getelementptr inb
  store double %20, doubl
  %23 = load i32* %0
  %24 = add i32 %23, %8
  store i32 %24, i32* %0
  br label %while.cond

while.end:                                        ; preds = %while.cond
  ret void
}
```

```
.visible .entry __kernel_0(
    .param .u32 __kernel_0_param_0,
    .param .u32 __kernel_0_param_1,
    .param .u32 __kernel_0_param_2,
    .param .u32 __kernel_0_param_3
)
{
    .reg .pred  %p<3>;
    .reg .s32   %r<21>;
    .reg .f64   %fd<4>;


    ld.param.u32    %r8, [__kernel_0_param_0];
    ld.param.u32    %r9, [__kernel_0_param_1];
    ld.param.u32    %r10, [__kernel_0_param_2];
    ld.param.u32    %r11, [__kernel_0_param_3];
    cvta.to.global.u32  %r1, %r11;
    cvta.to.global.u32  %r2, %r10;
    cvta.to.global.u32  %r3, %r9;
    mov.u32     %r12, %ntid.x;
    mov.u32     %r13, %ctaid.x;
    mov.u32     %r14, %tid.x;
    mad.lo.s32  %r20, %r12, %r13, %r14;
    mov.u32     %r15, %nctaid.x;
    mul.lo.s32  %r5, %r12, %r15;
    setp.ge.s32     %p1, %r20, %r8;
    @%p1 bra    BB0_2;

BB0_1:
    shl.b32     %r16, %r20, 3;
    add.s32     %r17, %r3, %r16;
    add.s32     %r18, %r2, %r16;
    ld.global.f64   %fd1, [%r18];
    ld.global.f64   %fd2, [%r17];
    add.f64     %fd3, %fd2, %fd1;
    add.s32     %r19, %r1, %r16;
    st.global.f64   [%r19], %fd3;
    add.s32     %r20, %r20, %r5;
    setp.lt.s32     %p2, %r20, %r8;
    @%p2 bra    BB0_1;

BB0_2:
    ret;
}
```

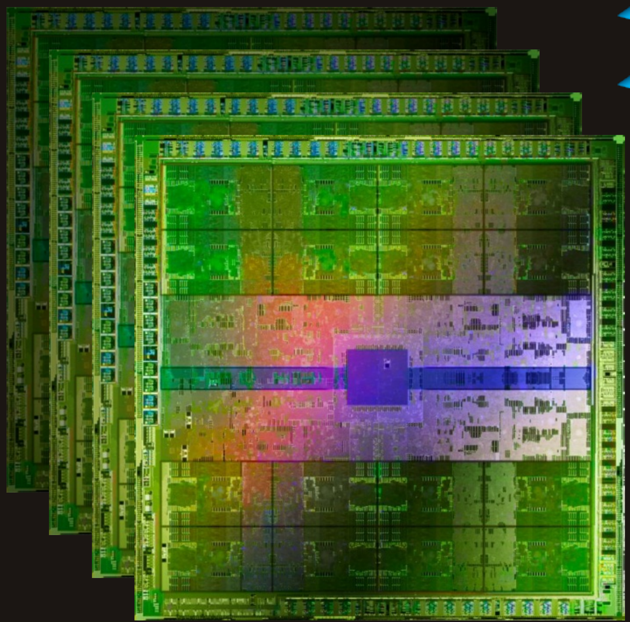
Segmented Scan by Key Alea.cuExtension against CUDA Thrust

Benefits

QuantAlea

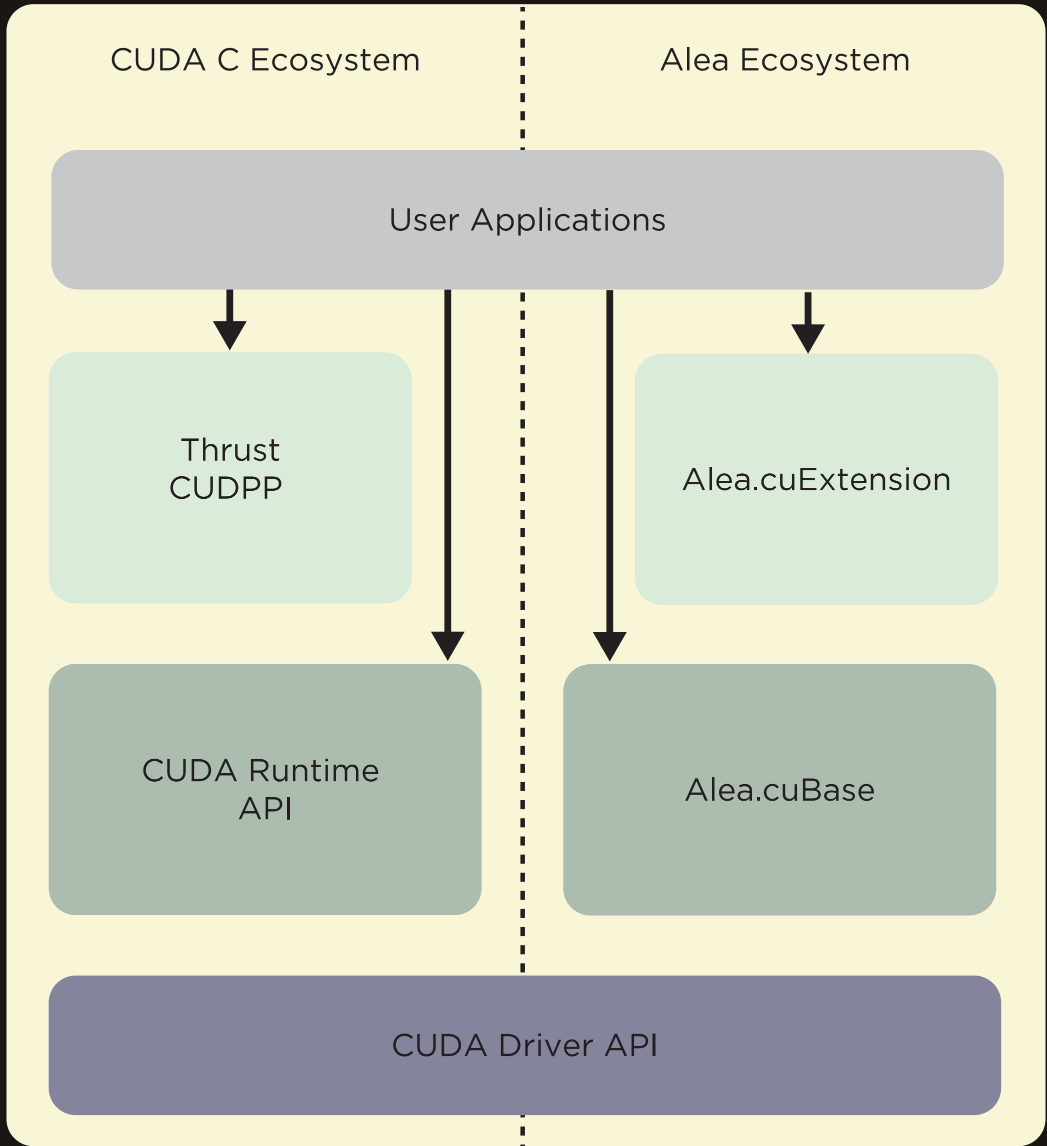
Advanced CUDA programming

- Support for texture, constant and shared memory
- Pointer operations to partition array data
- Special pointer types such as volatile pointers
- Runtime compilation control e.g. fast math
- Multiple streams
- Thread safe use of multiple GPUs
- Inline PTX assembly instructions



Ecosystem

QuantAlea



How does

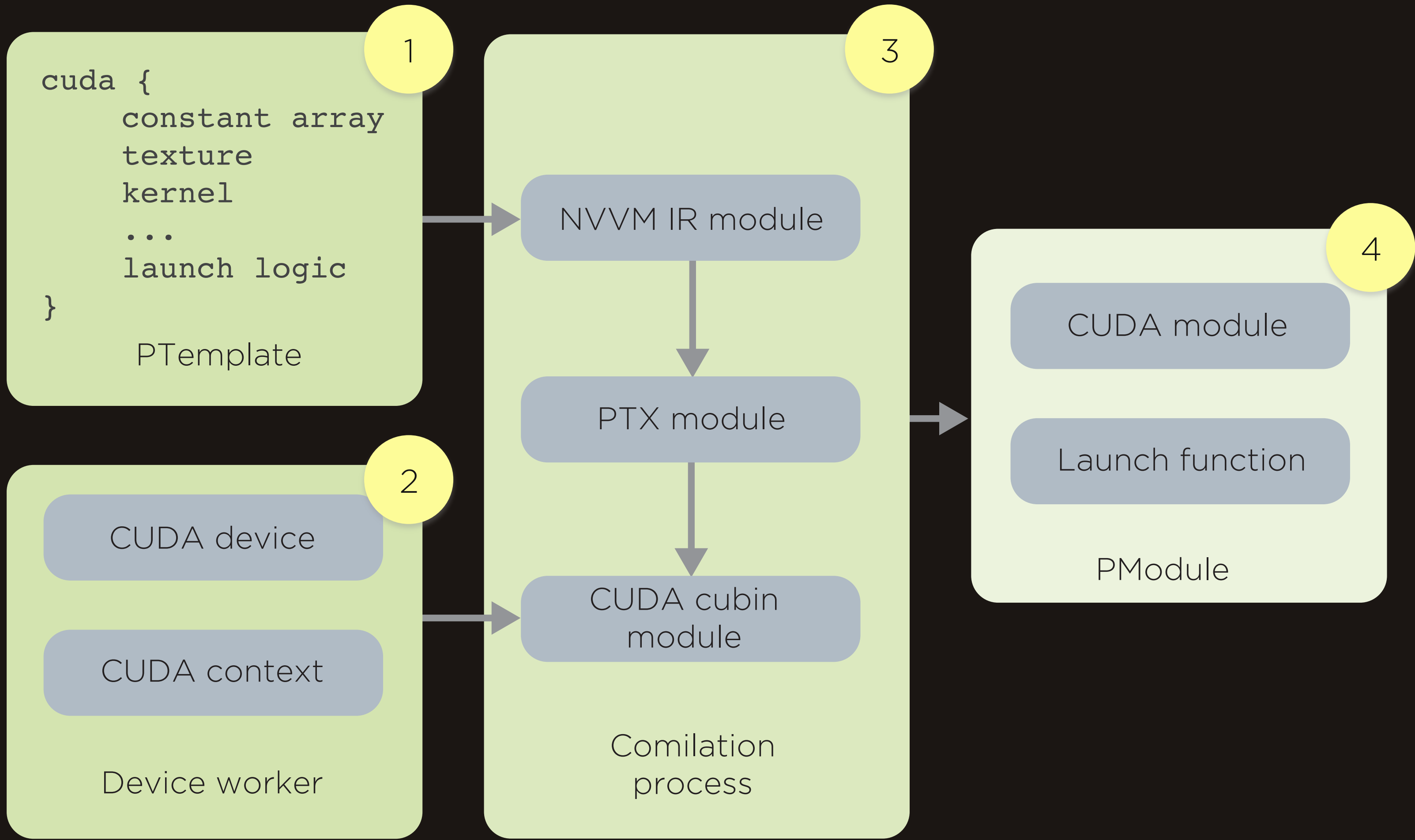
Alea.cuBase

work ?

Development Process

QuantAlea

- Four steps to a CUDA kernel with F# and Alea.cuBase



How easily can I use

Alea.cuBase

?

Live Coding

QuantAlea

- Basic kernel programming
- Excel GPU scripting with Alea.cuBase and Alea.cuExtension, in Tsunami IDE and FCell
  - Excel based Monte Carlo simulation
  - PDE solver for 2d heat equation in GPU

More Resources and Free Licenses

QuantAlea

- More resources
  - https://www.quantalea.net/products/resources/
  - https://github.com/quantalea
- How to set up
  - Fermi device or higher
  - Windows with .NET 4 and F# 2.0
  - CUDA 5 driver
  - Install Alea.cuBase
  - No need for CUDA toolkit or NVCC compiler
- Apply for free licenses
  - https://www.quantalea.net/news/22/

QuantAlea

Thank you