F# for Industrial Applications – Worth a Try?

Jazoon TechDays 2015

Dr. Daniel Egloff
Managing Director
Microsoft MVP
daniel.egloff@quantalea.net
October 23, 2015

Who are we

Software and solution provider for high performance and GPU computing

Technology driven financial services company

F# – worth a try?

QuantAlea

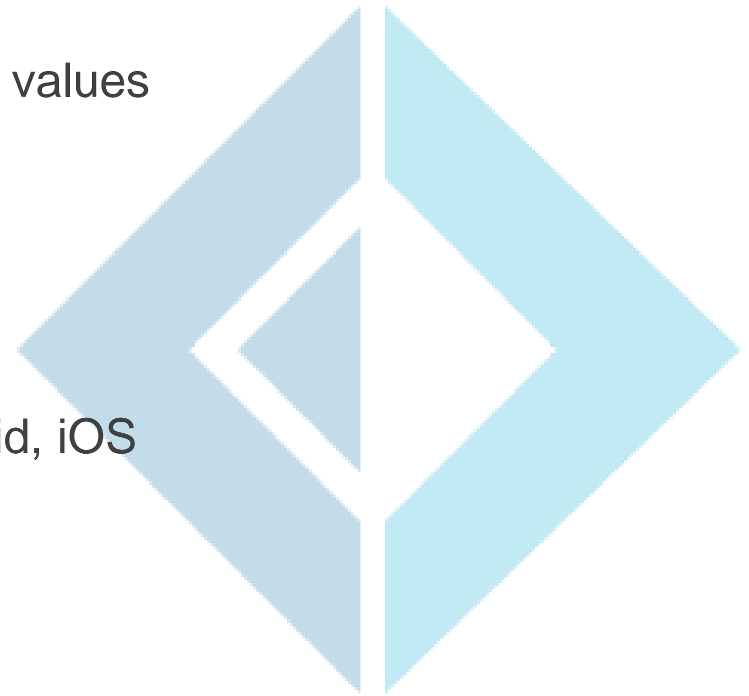
- What is F#?
- What distinguishes F#?
- What about the F# ecosystem?
- For what kind of projects would F# be a good fit?
- How do we use F# and what is our personal experience?
- What are the major benefits of F#?
- Conclusion

What is F#?

F# cornerstones

QuantAlea

- Multi-paradigm functional first programming language
  - Supports functional and object oriented programming paradigms
- Strongly typed, statically type-checked
  - Compiler automatically infers type of all values
- Open source
  - fsharp.org
- Cross platform
  - Mac, Linux, Windows, FreeBSD, Android, iOS



Functional first

QuantAlea

- Functions are first class members of the language
  - Bind an identifier to a function value
  - Store functions in data structures
  - Use functions as arguments in function calls and return values from function calls

```
let f1 a b = a + b
let f2 = f1 1

val f1 : int -> int -> int
val f2 : int -> int
```

Functional first

QuantAlea

```
let f x = x + 1
let g x = 2*x
let l = [f; g; f >> g; g >> f]
let eval f = f 10
let results = l |> List.map eval

val f : int -> int
val g : int -> int
val l : (int -> int) list
val eval : (int -> 'a) -> 'a
val results : int list = [11; 20; 22; 21]
```

Expressions

QuantAlea

- Every piece of code is a composable expression with a static type
- Expressions are evaluated eagerly
- Leads to safer and more compact code

```
let result =
    if cond then
        true_value
    else
        false_value
```

Algebraic data types

QuantAlea

- Compound data types are built by combining existing data types recursively as sums and products
- Sum = choice between variants of a type
- Product = tuple of types
- Recursive = type defined partially in terms of itself

```
type Tree<'T> =
| Leaf of 'T
| Node of 'T * Tree<'T> * Tree<'T>
```

Pattern matching

QuantAlea

- Representing sophisticated control flow
- Patterns = rules for transforming input data
- Pattern matching = test input data against patterns and dispatch
- Exhaustive pattern matching important for correctness
- Works well with algebraic data types

```
match expr with
| Add(l, r) -> let (lv, rv) = (eval l, eval r) in lv + rv
| Mul(l, r) -> let (lv, rv) = (eval l, eval r) in lv * rv
| Div(l, r) -> let (lv, rv) = (eval l, eval r) in lv / rv
| Constant(n) -> n
| Variable(var) ->
    match Map.tryFind var symbolTable with
        | Some v -> v
        | None -> failwith "variable %s not found" var
```

Pattern matching

QuantAlea

```
type Expr =
| Add of Expr * Expr
| Mul of Expr * Expr
| Div of Expr * Expr
| Constant of int
| Variable of string

let evaluate symbolTable expr =
    let rec eval expr =
        match expr with
        | Add(l, r) -> let (lv, rv) = (eval l, eval r) in lv + rv
        | Mul(l, r) -> let (lv, rv) = (eval l, eval r) in lv * rv
        | Div(l, r) -> let (lv, rv) = (eval l, eval r) in lv / rv
        | Constant(n) -> n
        | Variable(var) ->
            match Map.tryFind var symbolTable with
            | Some v -> v
            | None -> failwith "variable %s not found" var
     eval expr

let expr = Mul(Add(Variable "x", Constant 10), Variable "y")
let eval = evaluate (["x", 5; "y", 6] |> Map.ofSeq)
let value = eval expr
```

What distinguishes F#?

What distinguishes F#?

QuantAlea

- Concise
  - No clutter such as unnecessary braces, commas, semi-colons
  - Type specification almost never needed
- Expressive
  - Many programming tasks are shorter and simpler in F#
  - Less code to read, understand and maintain
- Designed for correct and robust code
  - Powerful type system prevents many common errors
  - Option type, no null values
  - Immutable by default
  - Sophisticated control flow with exhaustive pattern matching

Some advanced F#

QuantAlea

- Support for concurrency
  - Built in support for asynchronous programming with async work flow
  - Simplified parallel programming with messages and agents
- Computation expressions
  - Representing computations with non-standard aspects such as laziness, asynchronicity, state
- Type providers
  - Provide types, properties, and methods based on external information sources
  - F# contains several built-in type providers for Internet and enterprise data
  - The R Provider gives smooth interoperability between F# and R

Some advanced F#

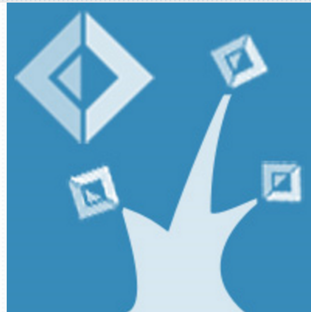
QuantAlea

- Code quotations
  - F# language feature to generate and work with F# code expressions programmatically
  - Typed and untyped code quotations
  - Very useful for building DSLs and compilers
  - Can be used for remote computations
- Active patterns
  - Dynamic customized decomposition of input data

The F# ecosystem

F# Community Projects

QuantAlea



The F# Community Space for incubating open community projects.

Admin and Statistics

Post an issue to add or remove a project. List your project on fsharp.org. Follow the guidelines.

73 public repos. 22 members

Recently updated View All on GitHub

Paket.VisualStudio Oct 21, 2015 · 47 stargazers · 19 forks
VisualFSharpPowerTools Oct 21, 2015 · 202 stargazers · 64 forks
Paket Oct 21, 2015 · 384 stargazers · 134 forks

Paket
F#
A package dependency manager for .NET with support for NuGet packages and GitHub repositories.
384 stargazers · 134 forks

VisualFSharpPower
F#
Power commands for F# in Visual Studio
202 stargazers · 64 forks

Paket.VisualStudio
C#
Manage your Paket (http://fsprojects.github.io/Paket/) dependencies from Visual Studio!
47 stargazers · 19 forks

70+ active projects

FSharp.Data.SqlClie
F#
An F# Type Provider for statically typed access to T-SQL command parameters and result set
80 stargazers · 27 forks

FSharpx.Extras
F#
Functional programming and other utilities from the original "fsharpx" project
427 stargazers · 126 forks

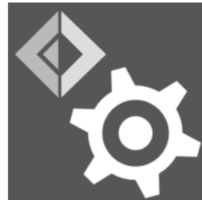
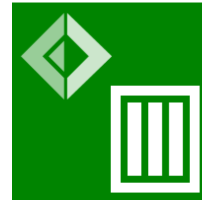
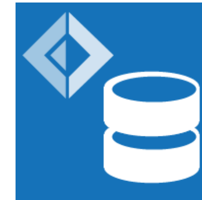
FSharp.Manageme
F#
The FSharp.Management project contains various type providers for the management of the machine.
34 stargazers · 15 forks

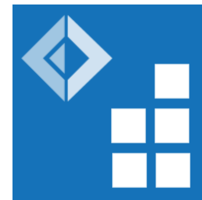
fsharp-dnx-templates
HTML
Visual Studio templates for creating DNX projects with F#.
5 stargazers · 0 forks

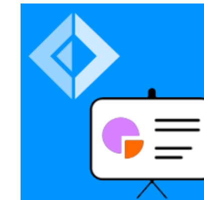
fsprojects.github.io

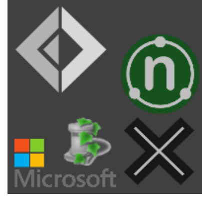
Growing F# ecosystem

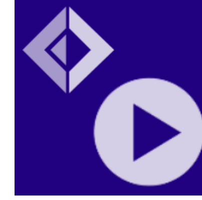
QuantAlea









Growing popularity and community

QuantAlea

- Active and friendly community
- Serious community contribution to the open source F# compiler project
- Popularity of F# is growing slowly
  - Increasing F# open source projects and contributions
  - In March 2014 the popularity of F# was ranked 12 by the TIOBE index
  - In October 2015 it is ranked 35, Scala 24, Lisp 28, Erlang 37, Haskell 40, Scheme 41 (close together from 0.8% to 0.3%)
- Functional concepts of increasing importance
  - Main stream programming languages adopt functional concepts

Where to apply F#?

Some application fields

QuantAlea

- Modelling complex domains
- Algorithms and numerical computing
- Compilers
- Internal and external domain specific languages
- Parallel programming
- Distributed computing
- Cloud and GPU computing
- Big data analysis
- Machine learning
- Type-safe scripting and rapid prototyping

Projects

GPU Compiler for .NET

Challenges

- Functional correctness
- Robustness
- Extensible architecture
- Flexible usability
- Challenging parallel GPU algorithms
- High expectations of professional developers
- Time to market

F# benefits

- Compact code
- Exhaustive pattern matching for code readability and correctness
- Code quotations to embed CUDA as internal DSL in F#
- Computation expression to define GPU resources in flexible way
- Cross platform

Code generation

QuantAlea

- Use active patterns to translate expressions into LLVM values

```
let irValue =
    match expr with
     | Value(clrObject, clrType) ->...
     | Var(var) ->...
     | VarSet(var, valueExpr) ->...
     | Let(var, valueExpr, followingExpr) ->...
     | Call(objectExpr, info, paramExprs) ->...
     | Sequential(valueExpr1, valueExpr2) ->...
     | IfThenElse(condExpr, thenExpr, elseExpr) ->...
     | WhileLoop(condExpr, bodyExpr) ->...
     | ForIntegerRangeLoop(indexVar, beginIndexExpr, endIndexExpr, bodyExpr) ->...
     | PropertyGet(objectExpr, info, paramExprs) ->...
     | PropertySet(objectExpr, info, paramExprs, valueExpr) ->...
     | FieldGet(objectExpr, info) ->...
     | FieldSet(objectExpr, info, valueExpr) ->...
     | Lambdas(vars, bodyExpr) ->...
     | Applications(lambdaExpr, valueExprs) ->...
```

GPU coding with quotations

QuantAlea

- Use quotations with splicing to write GPU kernel
- Use cuda work flow to define the GPU module

```
let moduleDefinition = cuda {
    let! kernel =
        <@ fun (output:deviceptr<'U>) (input:deviceptr<'T>) (n:int) ->
            let start = blockIdx.x * blockDim.x + threadIdx.x
            let stride = gridDim.x * blockDim.x
            let mutable i = start
            while i < n do
                output.[i] <- (%funcD) input.[i]
                i <- i + stride @>
        |> Compiler.DefineKernel

    return Entry(fun (program:Program) ->...

use program = moduleDefinition |> Compiler.load Worker.Default
Array.init n gen |> program.Run
```

Option Pricing Project

QuantAlea

Challenges

- Complex domain model
- Ability to expose domain model in other languages
- Challenging numerical algorithms
- Robust implementation with error handling
- Integration with legacy systems
- High performance and high throughput needs
- Significant time pressure

F# benefits

- Algebraic data types for domain modelling
- Higher order functions
- Integrated GPU computing capabilities
- Compiler development and code transformation
- Parallel and asynchronous programming
- Particularly suited for mathematical algorithms

Asset Management Project

Challenges

- Domain specific language for non-programmers
- Extensible distributed architecture
- Integration with legacy systems
- Rapid prototyping during project

F# benefits

- Algebraic data types for domain modelling
- Algebraic data types for DSL design
- Developing parser and compiler frontend
- Particularly suited for mathematical algorithms

Algorithmic Trading Project

Challenges

- Correctness
- Rapid prototyping capabilities
- Integration with R and C
- Time pressure

F# benefits

- Productivity
- Integration with .NET
- Suitable community libraries, in particular Akka.NET
- R Provider
- Easy refactoring capabilities

Major benefits of F#?

F# Pros and Cons

QuantAlea

Pros

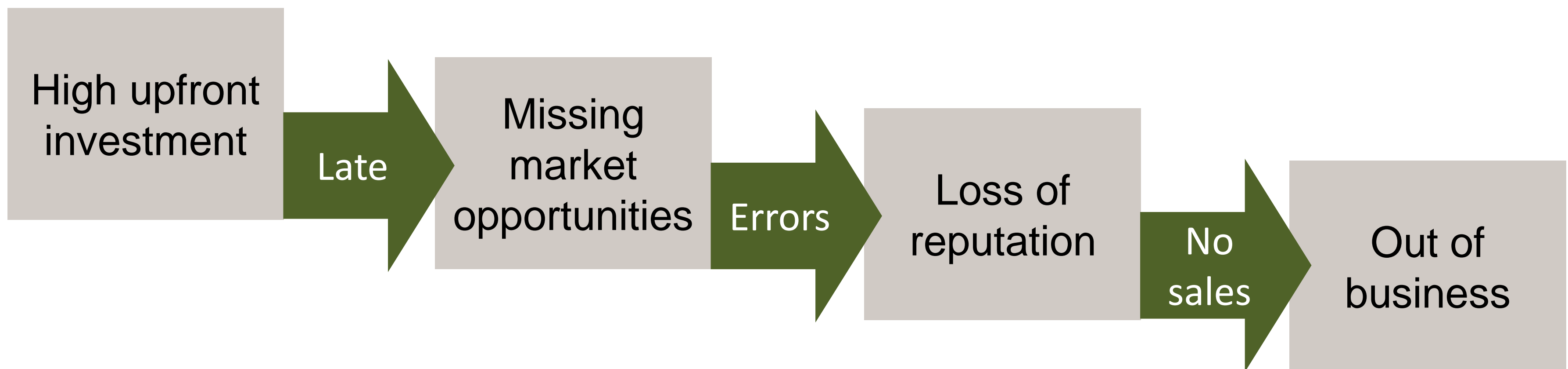
- Productivity
- Ability to develop compact, robust and correct code
- Modern technology
- Unique language features

Cons

- Niche language, most likely never main stream
- Small job market
- Tooling not at the level of C# or Java
- Smaller ecosystem and community

Business impact

- Ability to develop compact, correct and robust code has a business impact
  - Time of engineers is highly valuable
  - Software shipping on time with fewer defects is worth a lot



Conclusions

Worth to learn F#?

QuantAlea

- F# is a highly productive language
- F# can give you a competitive edge
- We have successfully used F# in various projects for over 5 years
- We develop commercial applications with F#

- Make the choice yourself – use the best tool for the problem at hand

We look for talents

QuantAlea

recruiting@incubegroup.com

Thank you

Dr. Daniel Egloff
daniel.egloff@quantalea.net
October 23, 2015