1

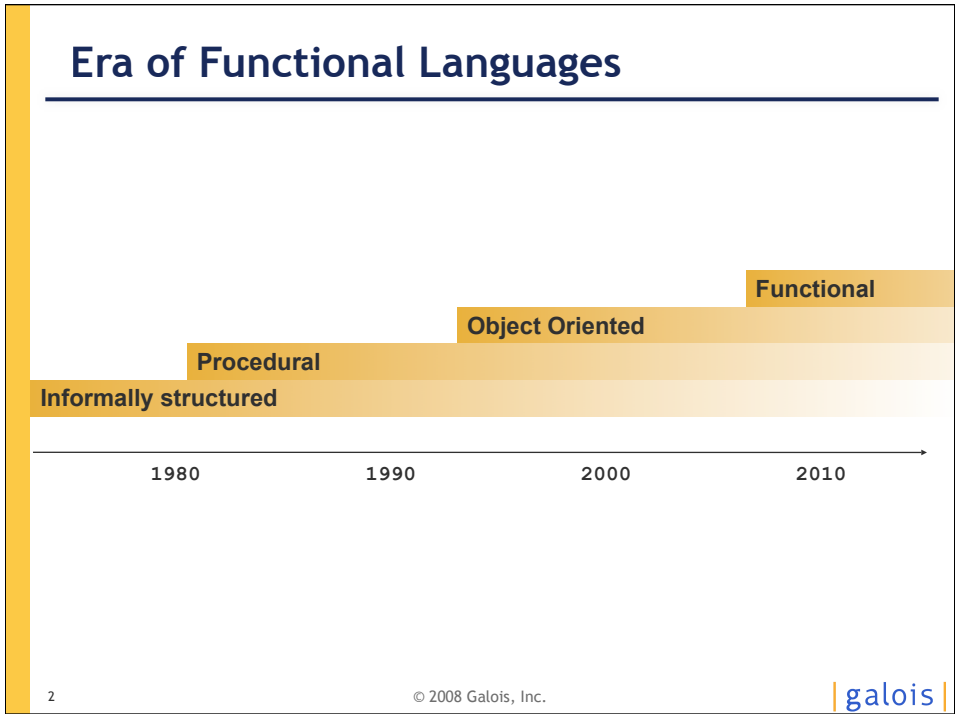
David Fellows, CTO Comcast, “The future is easy to see, it’s timing that’s hard to get right.”

-- Despite this, I’m going to make a prediction about the future

Abstract:

Functional languages have been under academic development for over 30 years. Throughout that time they supplied key innovations to mainstream languages, but did not meet widespread acceptance themselves. All that is changing now. A new ground swell of interest is emerging in functional languages because of their ability to provide compelling stories for the critical software challenges of the decade: multicore and software assurance. Microsoft has noticed this, and recently announced F#, their first functional language product.

In this talk, we show how the functional language Haskell may be used for system development, supporting a smooth transition from modeling to implementation. More broadly, we examine the industrial forces and trends and provide case studies, all to explain how and where functional languages will fit within future industrial software.



This is my claim. I'm going to back it up. By the end of my talk you might even believe me.
I want to show you the way the world is evolving & what's going on, and show you the implications

Not: "my language is better than yours"
Rather: "here's my prediction of what's going to occur"

What is a "Functional Language"?

Assembly

```
Load X
Add Y
Store T
Load W
Mul Z
Add T
Store A
```

Procedural

```
A := X*Y + W*Z
```

```
{N is array size}
a := 0;
for i := 1 to N;
do
begin
 a := a+X[i]*Y[i]
end;
```

Functional

```
a = sum (zipWith (*) xs ys)
```

Functional style handles *structures* as *values*

Examples

Haskell, ML, O'CAML, Scheme, F#, Lustre, Erlang, Mercury, Oz

© 2008 Galois, Inc.

All practical functional languages blend two styles:

- Purely functional
- Effects (communication, state, I/O)

The World Just Changed: Multicore

"It's incredibly tough to program to multicore; there are not that many people in the world that can even do it."

Ray DePaul
CEO RapidMind, 2007

© 2008 Galois, Inc.

How Bad Can it Get?

Tesla: 64 core machine

Recent email:

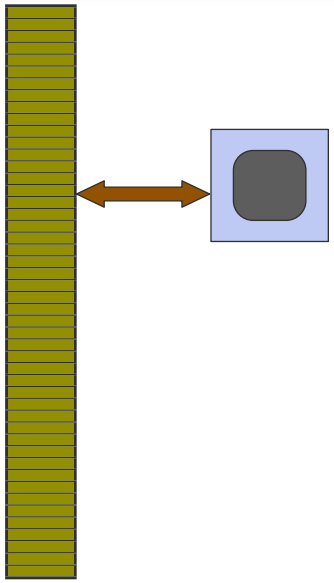
I'm on the GPGPU side here at [...].
Does 320 cores sound interesting?
What do you need?

Your C programs will not scale

© 2008 Galois, Inc.

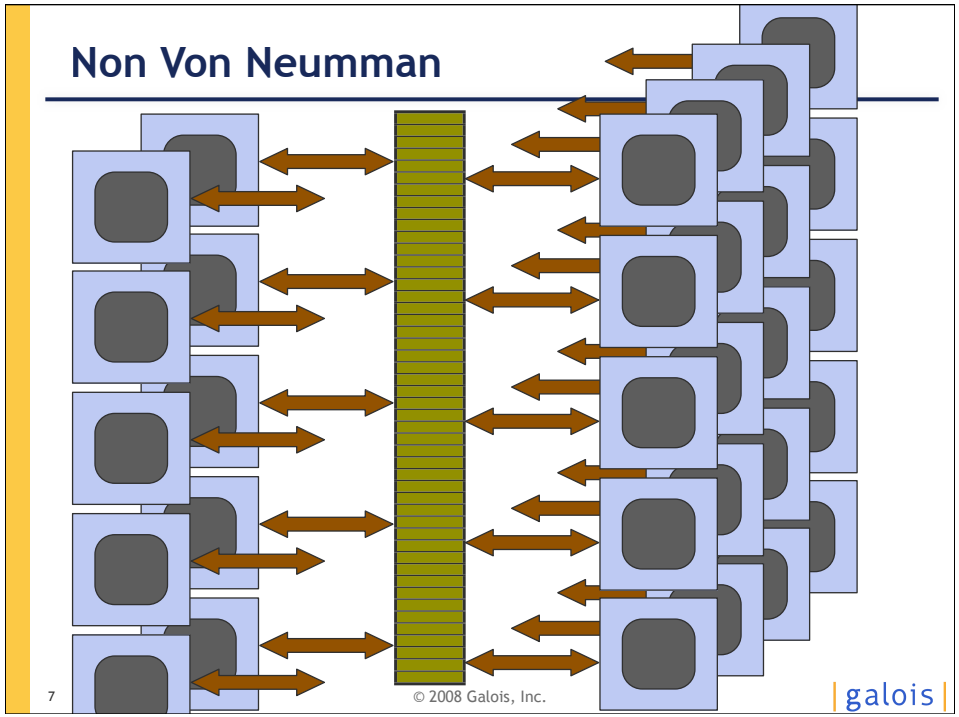
Tesla, 64 core machine

Von Neumman Assumption



© 2008 Galois, Inc.

Instructions are sequenced
Mainstream languages have this assumption by default



One program spread out over all the processors

Parallel Haskell

Jun 12 1990, 12:13 pm

Subject: Re: Haskell availability

Someone asked about whether anyone was working on parallel implementations of Haskell.

Good news: we are; we have a parallel implementation running on the GRIP multiprocessor, with absolute wall-clock speedup over the same programs running on a comparable uniprocessor (never to be taken for granted!) ...

The compiler would port rather easily to a shared-memory multiprocessor, but we don't have access to one at present.

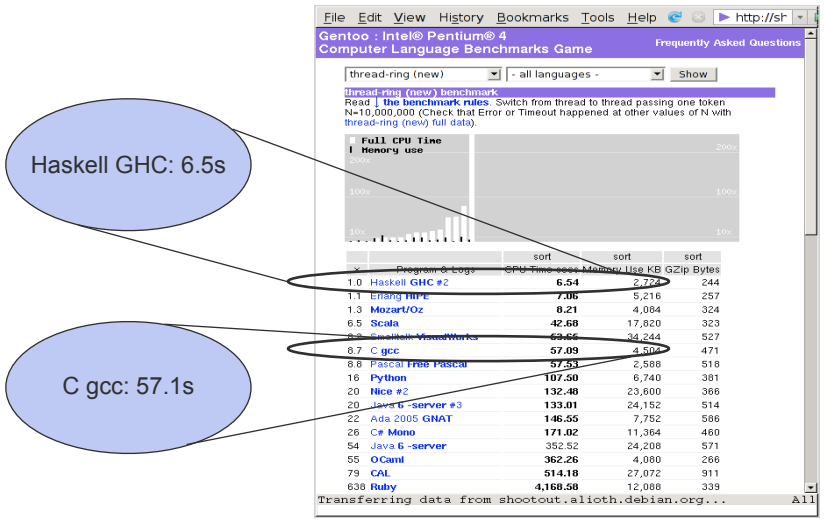
Simon Peyton Jones, Glasgow University

© 2008 Galois, Inc.

The compiler gets to make lots of choices about compiling expressions
-- purity is important

Intellectual reach

Haskell Threads are Cheap



© 2008 Galois, Inc.
galois

Haskell was 26 times faster than C# on this test
Other tests have different precise orderings of course, C often on top.

The point: functional languages should not be rejected as “slow”
Also: low cost of threads => programmers will use them all over the place

Haskell on Multicore

- Purely functional language
  - Effects are controlled, no inherent sequencing
  - Long history of parallel evaluation research
  - World-class optimizing compiler
- Threads
  - Forking/blocking Haskell threads, as well as OS threads
  - Race conditions rarely arise
  - Composable concurrency: transactional memory
- Parallel evaluation strategies
  - GHC 6.10: parallel GC
  - GHC 6.10: -fvectorise – automatic parallel arrays
  - Research on `par` discovery

© 2008 Galois, Inc.

Race conditions come from side effects and concurrency

The Forces at Work

- Multicore !
- Microsoft !!
- Professional engineering discipline
- Productivity and cost

© 2008 Galois, Inc.

Microsoft's Functional Language

"One of the important themes in programming languages over recent years has been a move to embrace ideas from functional programming.

[Ideas] from functional languages are helping us address some of the biggest challenges facing the industry today, from the impedance mismatch between data and objects to the challenges of the multi-core and parallel computing space...

F# stems from the functional programming tradition (hence the 'F') and has strong roots in the ML family of languages, though also draws from C#, LINQ and Haskell ...

We will ...
fully integrate the F# language into Visual Studio and continue innovating and evolving F#. In my mind, **F# is another first-class programming language on the CLR** ..."

Soma Somasegar
Microsoft Developer Division Chief
17 Oct 2007

© 2008 Galois, Inc.

The Forces at Work

- Multicore !
- Microsoft !!
- Professional engineering discipline
- Productivity and cost

© 2008 Galois, Inc.

Fred Brooks

" How Do We Know What to Design?

One easily thinks of the design of complex systems, hardware or software, as a rational process that has a rational model [...]

Upon examination, though, such models don't seem to fit the way real designers work. In particular, I would assert that **it is impossible to set the requirements for such a design before beginning**.

[...] in software engineering, at least, the Waterfall Model persists, tenaciously and disastrously. Why? Is there any hope for remedying this situation? "

Cambridge, 17 Oct 2007

Fred's Solution:

Separate procurement into **design** and **build.**

Different phases and contracts (like civil engineering).

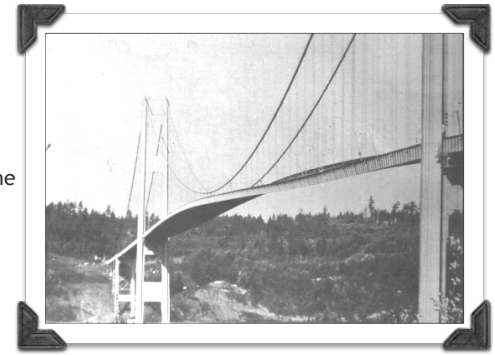
© 2008 Galois, Inc.

1986 - No Silver Bullet

The design contract emphasizes proof-of-concept and fast prototyping.

High Assurance Software

- Let the software itself be trustworthy
  - Software artifacts to speak for themselves
  - ... rather than hoping to rely on the process that created them
- Use mathematical models to enable tractable analysis
  - Executable models and formal methods
  - A model is an abstraction that allows thought at a higher level
- Follow open standards
  - Build components with high internal integrity
  - Maximize interoperability



Have software built with
the same diligence and analysis
as other engineers build bridges

© 2008 Galois, Inc.

Rule of signs is an example of an abstract model

Engineering in Haskell

Modeling

- Mathematical foundation
  - Allows for mathematical guarantees of behavior
  - High assurance
- Very powerful abstraction
  - Say what needs to be said, nothing more
  - Easier to build smarter software
- Executable models
  - Automatic memory access and protection
  - Flexible and powerful

Production

- Smooth path from model to product
  - The executable model is the first prototype
  - Incremental refinements from problem focus to solution focus
- Huge productivity benefits
  - Shorter (2-10x), clearer, and more maintainable code
  - Reducing time-to-deployment
- Scalable to complex systems
  - Concise expression
  - Multicore ready !!

© 2008 Galois, Inc.

Web server, federation of media wikis,

Paul Graham: Shop.com --> Yahoo Stores

Erlang in telecom switches

X-Monad: open source in Haskell by one of our engineers

X-Monad Feedback

"Over the past twelve months, 31 developers contributed new code to xmonad. This is one of the largest open-source teams in the world, and is in the top 2% of all project teams on Ohloh ..."

Ohloh Metrics, Feb 2008

"Suspiciously I relate to any software written in the "exotic" languages of programming.

Usually either break is obtained or memory gorges much.

But here everything is written on the fashionable nowadays Haskell, very rapid and memory it does not gorge."

(Russia)

© 2008 Galois, Inc.

About 500 lines of code – very well designed
Don't assume short = quick
(Story about Pascal)

Lot of design work embodied in this code

QuickCheck: Property-based Testing

- Write properties (using Haskell)
- State a property about other Haskell functions
- Quick Check automatically generates test cases based on the structure of the type

```
insert x xs = takeWhile (<x) xs++[x]++dropWhile (<x) xs
```

```
ordered xs = and (zipWith (<=) xs (drop 1 xs))
prop_Insert x xs = ordered xs ==> ordered (insert x xs)
```


© 2008 Galois, Inc.

Example Coverage Markup

```
 1  reciprocal :: Int -> (String, Int)
 2  reciprocal n | n > 1 = ('0' : '.' : digits, recur)
 3               | otherwise = error
 4                   "attempting to compute reciprocal of number <= 1"
 5    where
 6    (digits, recur) = divide n 1 []
 7  divide :: Int -> Int -> [Int] -> (String, Int)
 8  divide n c cs | c `elem` cs = ([], position c cs)
 9                | r == 0      = (show q, 0)
10                | r /= 0      = (show q ++ digits, recur)
11    where
12    (q, r) = (c*10) `quotRem` n
13    (digits, recur) = divide n r (c:cs)
14
15  position :: Int -> [Int] -> Int
16  position n (x:xs) | n==x       = 1
17                    | otherwise = 1 + position n xs
18
19  showRecip :: Int -> String
20  showRecip n =
21    "1/" ++ show n ++ " = " ++
22    if r==0 then d else take p d ++ "(" ++ drop p d ++ ")"
23    where
24    p = length d - r
25    (d, r) = reciprocal n
26
27  main = do
28    number <- readLn
29    putStrLn (showRecip number)
30    main
```


© 2008 Galois, Inc.

Haskell Program Coverage Dashboard

module	Top Level Definitions		Alternatives		Expressions	
	%	covered / total	%	covered / total	%	covered / total
module CSG	-	0/0	-	0/0	-	0/0
module Construct	100%	25/25	100%	12/12	100%	569/569
module Data	91%	22/24	60%	24/40	90%	527/585
module Eval	95%	19/20	93%	59/63	96%	541/561
module Geometry	100%	45/45	60%	6/10	95%	335/351
module Illumination	100%	15/15	72%	26/36	96%	415/428
module Intersections	100%	22/22	81%	68/83	87%	879/1001
module Interval	70%	12/17	73%	17/23	78%	129/165
module Main	100%	1/1	-	0/0	100%	5/5
module Misc	100%	1/1	-	0/0	100%	10/10
module Parse	100%	17/17	100%	8/8	100%	234/234
module Primitives	33%	2/6	-	0/0	41%	10/24
module Surface	55%	5/9	85%	17/20	92%	205/221
Program Coverage Total	92%	186/202	80%	237/295	92%	3859/4154

© 2008 Galois, Inc.

Interaction of property-based testing (e.g. quickcheck) and coverage is fascinating. Worth a paper. Or PhD.

Haskell Engineering Advice

- Use the type system to enforce properties
- Use QuickCheck as your design assistant
- Use HPC (program coverage) to test QuickCheck's reach
- Enforce code quality with serious testing on every commit
- Don't be tempted by partial functions
- Model effectful systems in purely functional data structures
- Don't be tempted by side effects
- Be responsive to bug reports
- Look at your competition's bugs, audit and prevent them

© 2008 Galois, Inc.

"My favorite pro: ease of maintenance! Change the data type and let the compiler walk you through the entire code base pointing to every single place you need to worry about."

The Forces at Work

- Multicore !
- Microsoft !!
- Professional engineering discipline
- Productivity and cost

© 2008 Galois, Inc.

Dijkstra on Code Size

A line of code is a cost not an asset

"[...] expressed programmer productivity in terms of 'number of lines of code produced'.

[...] I pointed out that a programmer should produce solutions, and that, therefore, we should not talk about the number of lines of code produced, but the number of lines used, and that this number ought to be booked on the other side of the ledger. "

E.W. Dijsktra, Sept 1975

© 2008 Galois, Inc.

Factor of 6-10: Assembly -> C
Factor of 6-10: C -> Haskell

Old Prototyping Study

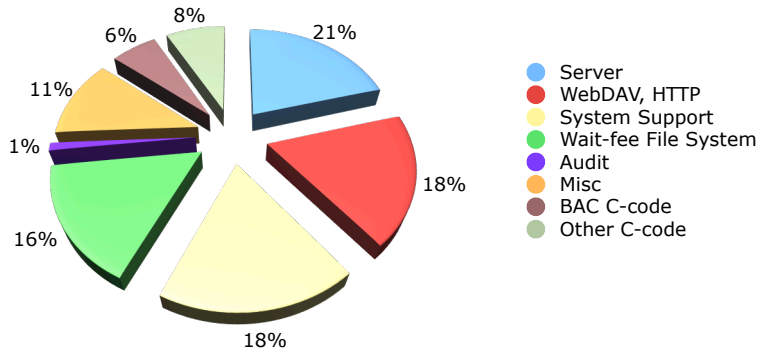
	Lines of Code	Lines of Doc	Hours
Haskell	85	465	10
Ada	767	714	23
C++	1105	130	
Awk	250	150	

Haskell vs. Ada vs. C++ vs. Awk vs. ...
An Experiment in Software Prototyping Productivity
Hudak & Jones, 1994

© 2008 Galois, Inc.

Ericsson: claim a factor of 6 for Erlang, privately say this is grossly understated

Line Count Breakdown of Web-Server

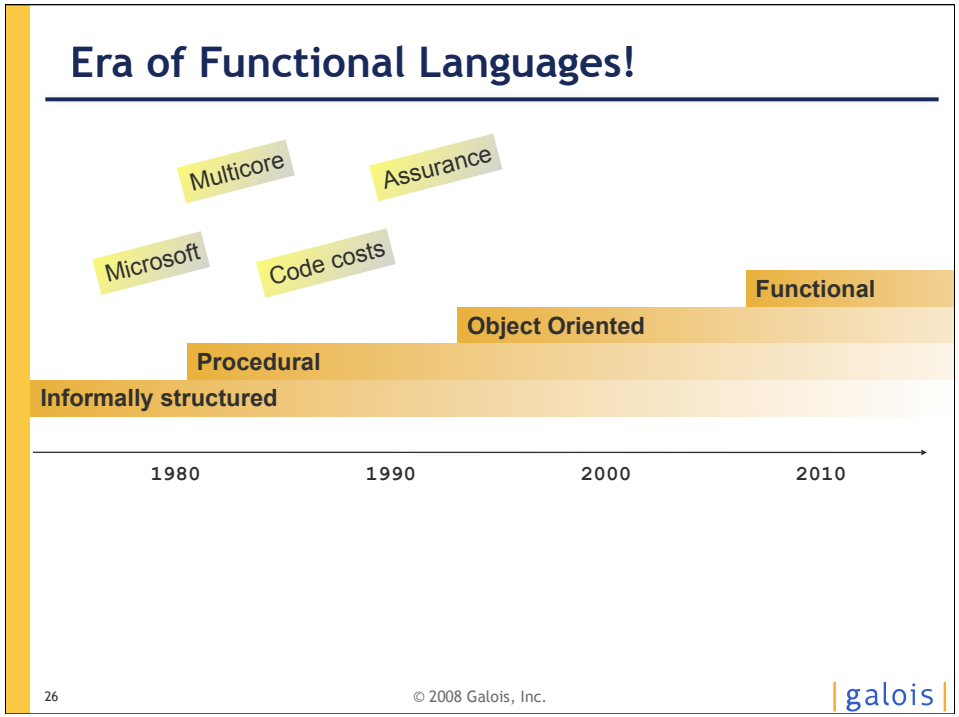


Total: 54KLOC of Haskell & C

25 © 2008 Galois, Inc. | galois |

10K of code gives a good chunk of major functionality

Use other languages when they are what is needed



Let's make a date: HCSS 2018.

You can praise me for my insightful prophecy,

or you can throw rotten tomatoes at me.