pH: Lessons Learned

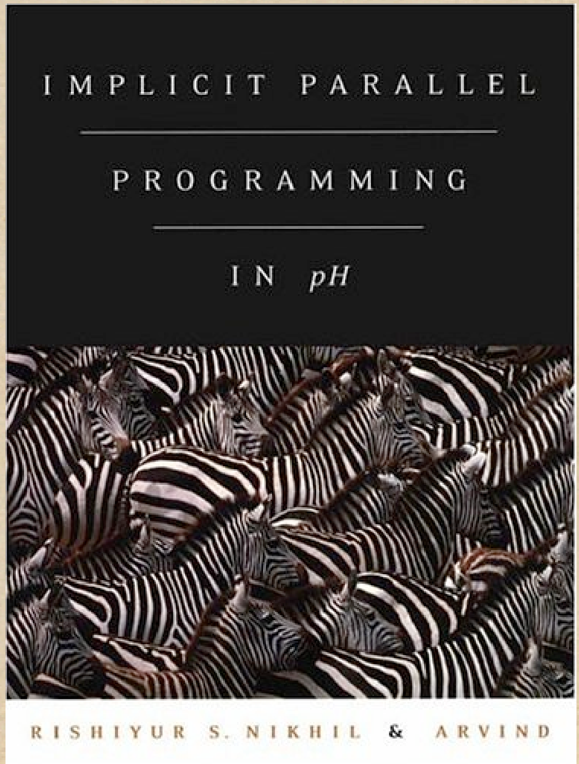
Jan-Willem Maessen
Sun Microsystems Laboratories
Arvind
MIT CSAIL

DPMCA, Charleson, SC, 15 Jan 2006

The pH Language

- Id execution model + Haskell syntax and types
- Implicitly parallel, non-strict, eager evaluation

```
flop :: Int -> [Int] -> [Int]
flop n xs = rs
  where (rs,ys) = for i <- [1..n] do
                       x:next xs = xs
                       next ys = x:ys
                     finally (ys, xs)
```



- Every subexpression may run in parallel
- Heap may hold partially-computed data

This talk

- Historical perspective
  - Roots in Id and dataflow execution model
  - Id becomes Id90, a modern FP language
  - Threads and von Neumann execution
  - Transition to pH and Eager Haskell
- Lessons
  - Unexpected hurdles
  - Multi-core architectures

The birth of Id

- Textual language describing dataflow graphs (1977-78, Arvind@UC Irvine)
  - Dynamically typed (Influences Lisp, FP)
- Idsys: Id compiler at MIT (1979 – 82; Pingali, Kathail)
- Id World: dataflow graph, executed on a graph interpreter GITA (1982-88)
  - Everything ran in parallel, parallelism profiles
  - Worked out dataflow function call
  - Invention of I- and M-structures to avoid copying arrays during construction

(Traub, Morais, Culler, Nikhil, Pingali...)

I- and M-structures

Storage + synchronization in one.

- I-structure: write once.
  - Read: block until write occurs
- M-structure: write many synchronization
  - Take: block until a value is written, then remove it and mark the location empty
  - Capture non-determinism in system code
  - Build classic mark-based graph algorithms

Id Noveau / Id 90

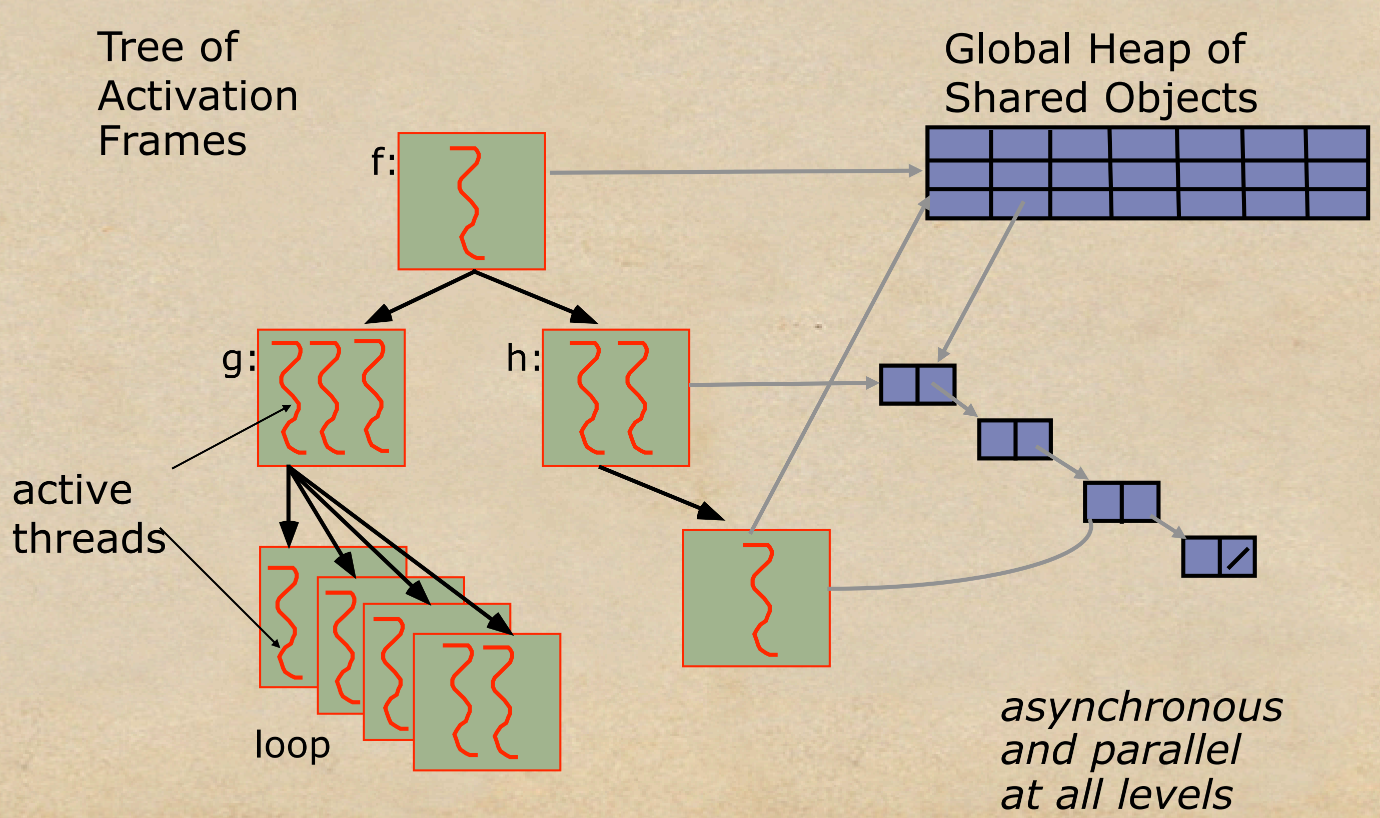
- Id becomes a modern Functional Language:
  - Hindley-Milner types
  - Array and list comprehensions
  - I-structures are de-emphasized in source
- Better resource management
  - Compiler-inserted free operations
- 2x larger compiler code base (Nikhil, Hicks)
  - For a compiler which already worked well

Shifting focus to runtime

- By the mid 80's, key issues had been resolved
  - There was more than enough parallelism
  - Id compiler was starting to see heavy use (LANL)
  - For systems programming tasks, too!
- Bounded loops
  - Limit actual parallelism, space consumption
- Classical optimizations on dataflow graphs
  - Basically an SSA compiler
- Suspensive threading...

Id/pH Thread Model

- Non-Suspensive Thread ≈ basic block



Great on Monsoon

Boon Ang, Derek Chiou, Jamey Hicks

	speed up				critical path (millions of cycles)			
	1pe	2pe	4pe	8pe	1pe	2pe	4pe	8pe
Matrix Multiply 500 x 500	1.00	1.99	3.90	7.74	1057	531	271	137
Paraffins n=22	1.00	1.99	3.92	7.25	322	162	82	44
GAMTEB-2C 40 K particles	1.00	1.95	3.81	7.35	590	303	155	80
SIMPLE-100 100 iters	1.00	1.86	3.45	6.27	4681	2518	1355	747

September, 1992

Could not have asked for more

Dataflow architecture supports the execution model beautifully.

pH: parallel Haskell

- Id said nothing too new about types
- Haskell had a sexy new type system and a community of researchers
- 1993: Adopt a Haskell personality for Id
- Peyton Jones, Augustsson, Nikhil, Arvind
  - Front end by Lennart Augustsson
- First back end: the Id compiler
- pH back end (1998)
  - Alejandro Caro, RTS by J-W Maessen

Parallel Iteration

- Uses unfold, synthesize (parallel unfold)
- Can say associative, commutative:

```
sum xs = reduce (+) 0 (someOrder xs)
```

- Produce a foldl (not foldr) where possible

```
foldr ∘ someOrder = foldl
```

- The concat function yields nested parallelism

```
reduce (+) 0∘concat = reduce (+) 0∘map (reduce (+) 0)
```

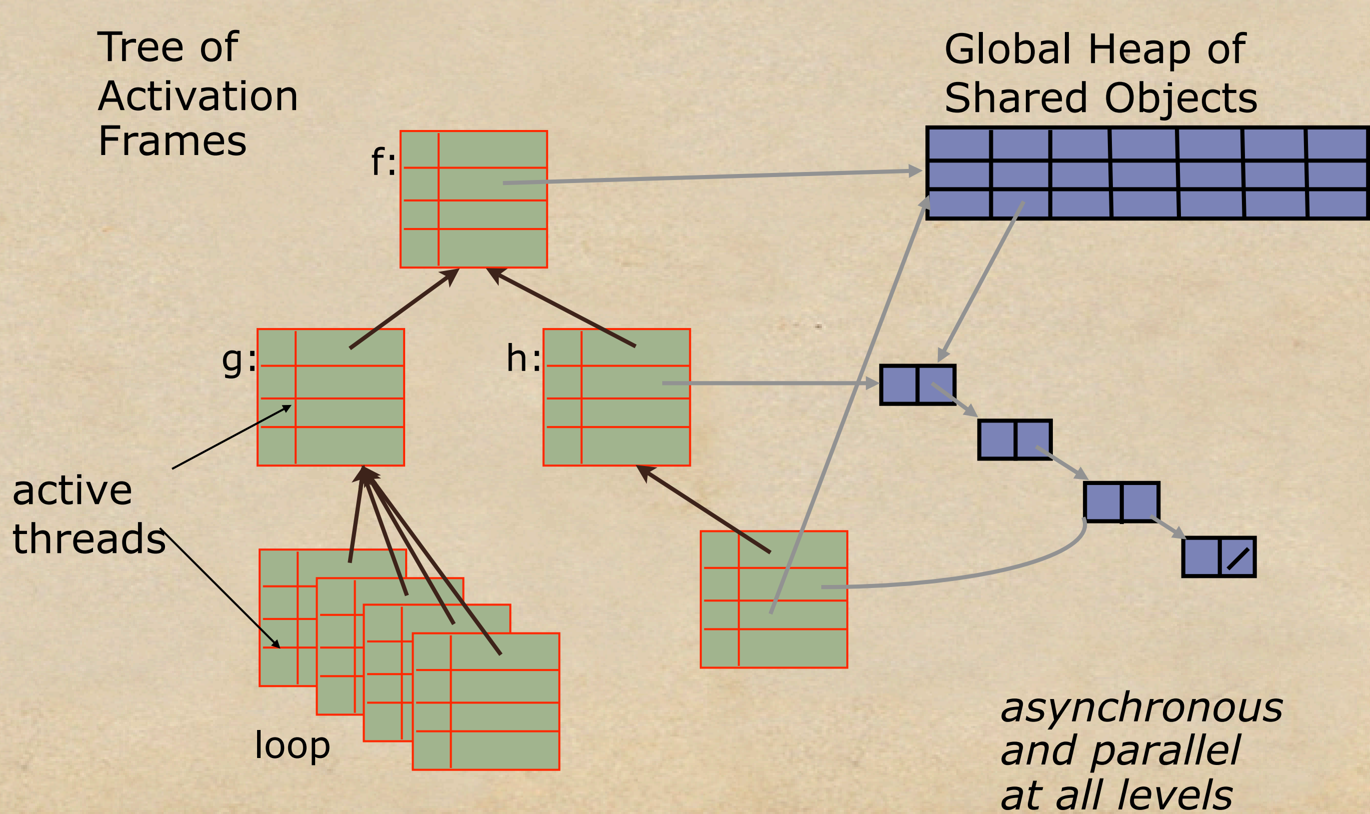
- Abelian operator merges I-structure effects
  - Monadic approach would impose an order

On to von Neumann

- By mid-90s it was clear dataflow wouldn't keep pace with off-the-shelf processors.
  - Target stock SMP machines instead.
- Must tell the hardware which thread to run
- Suspensive Thread ≈ super-block
  - Chain together dependent threads
- Re-think data copying between frames

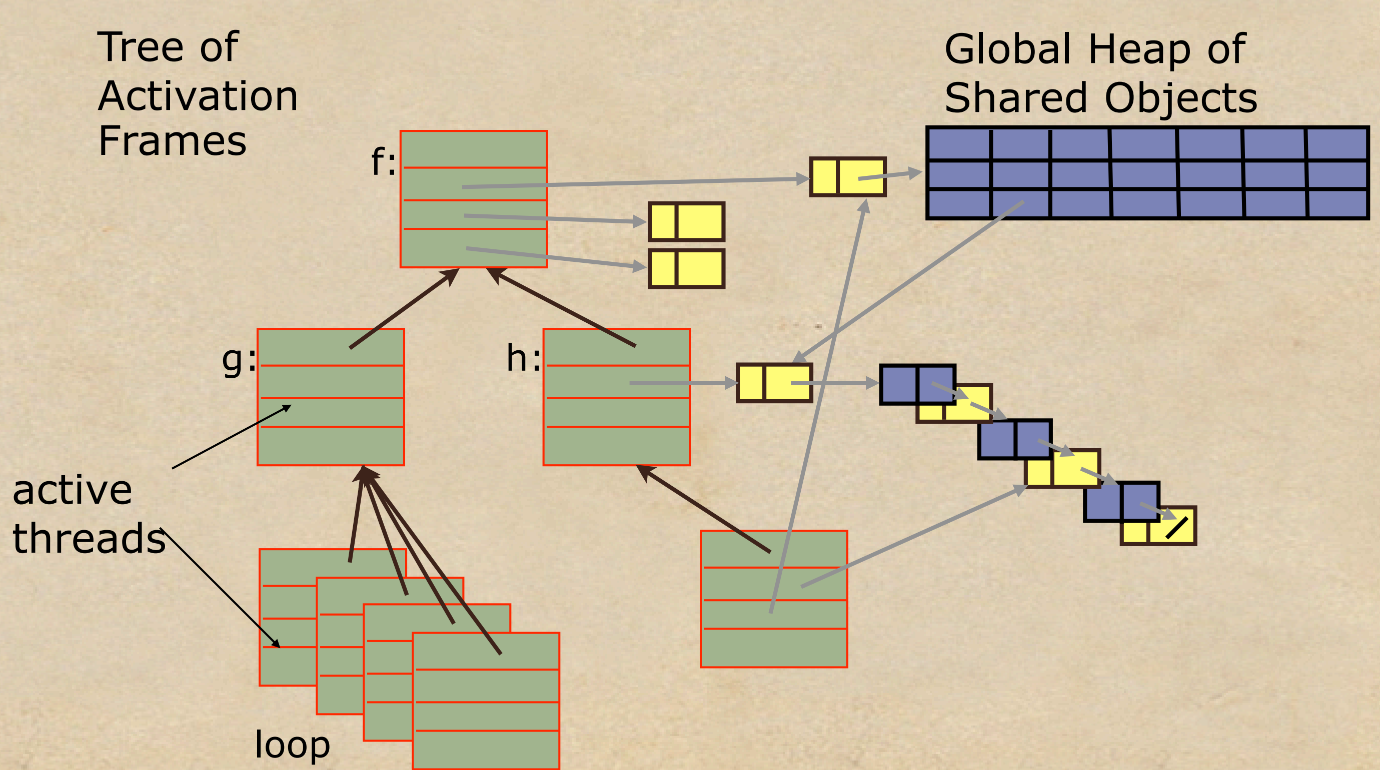
Id Thread Model

- Everything is an I-structure cell
- Copy data from caller frame to callee and back



pH Thread Model

- I-structure proxy if possibly uncomputed
- No copying required; call-by-reference



Threading in pH

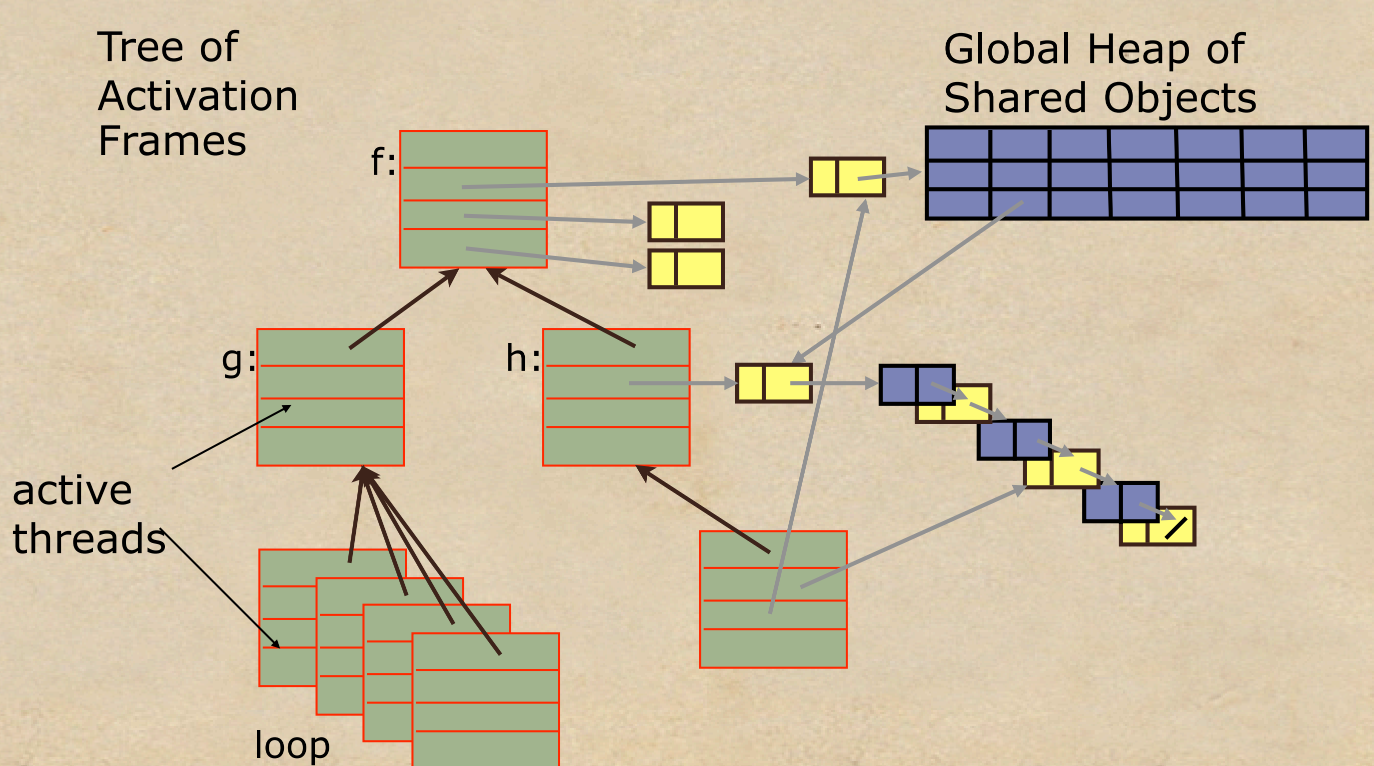
- Incorporate control flow in suspensive threads
- Spawn a new thread only when:
  - There are multiple dependent blocks
  - One of them actually suspends
- Compiled code looks like familiar strict code
  - Except there's a lot of checking
  - And a scattering of resumption points
- Obvious how to exploit (eg) strictness analysis

Scheduling

- Work stealing a la Cilk
  - Follows usual call/return pattern
  - Good temporal locality in practice
  - Low overhead in common case
- But what about I-structures?
  - Read: add ourselves to a defer list
  - Run the defer list on write
- Adds check in common case
- Wrecks temporal locality

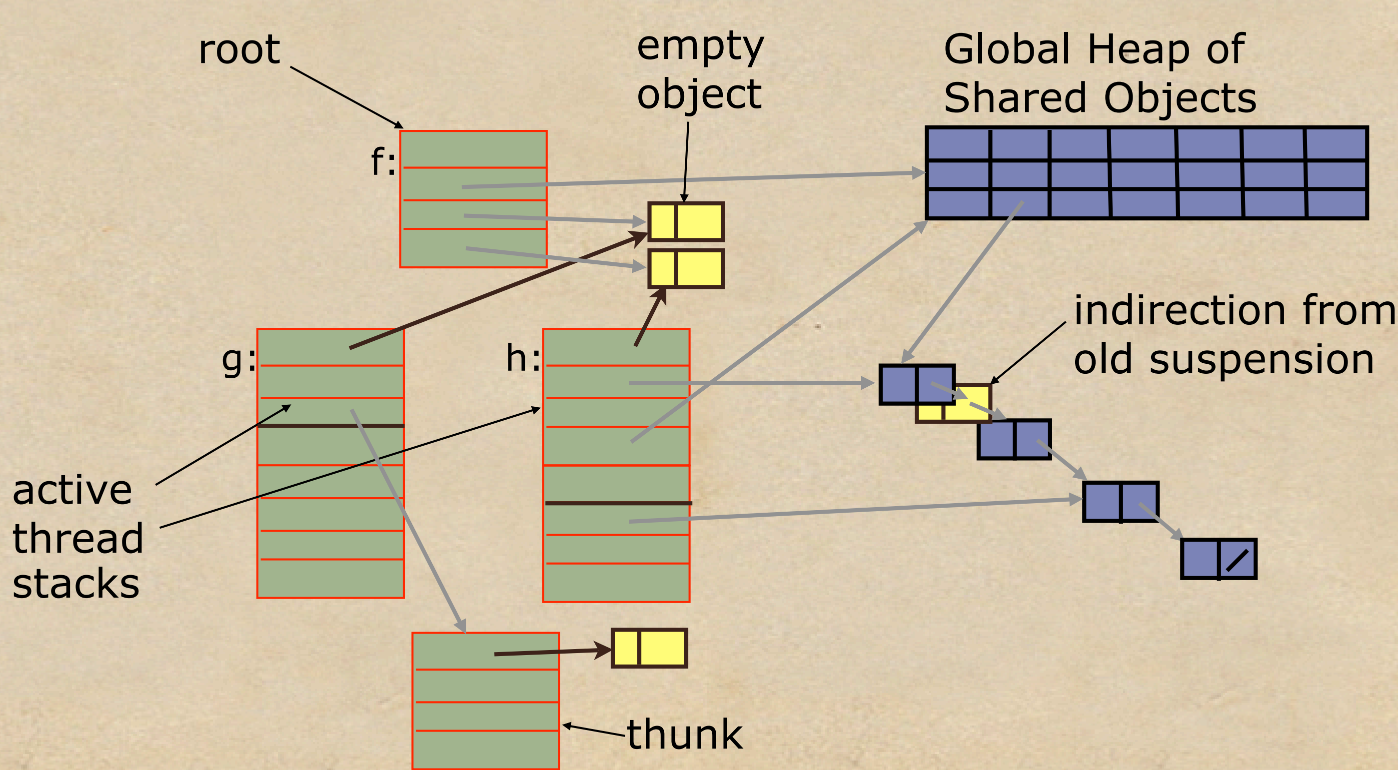
pH Thread Model

- Look at all those levels of indirection!



Eager Thread Model

- Active threads arranged in a stack
- Indirections only for stuff which suspends



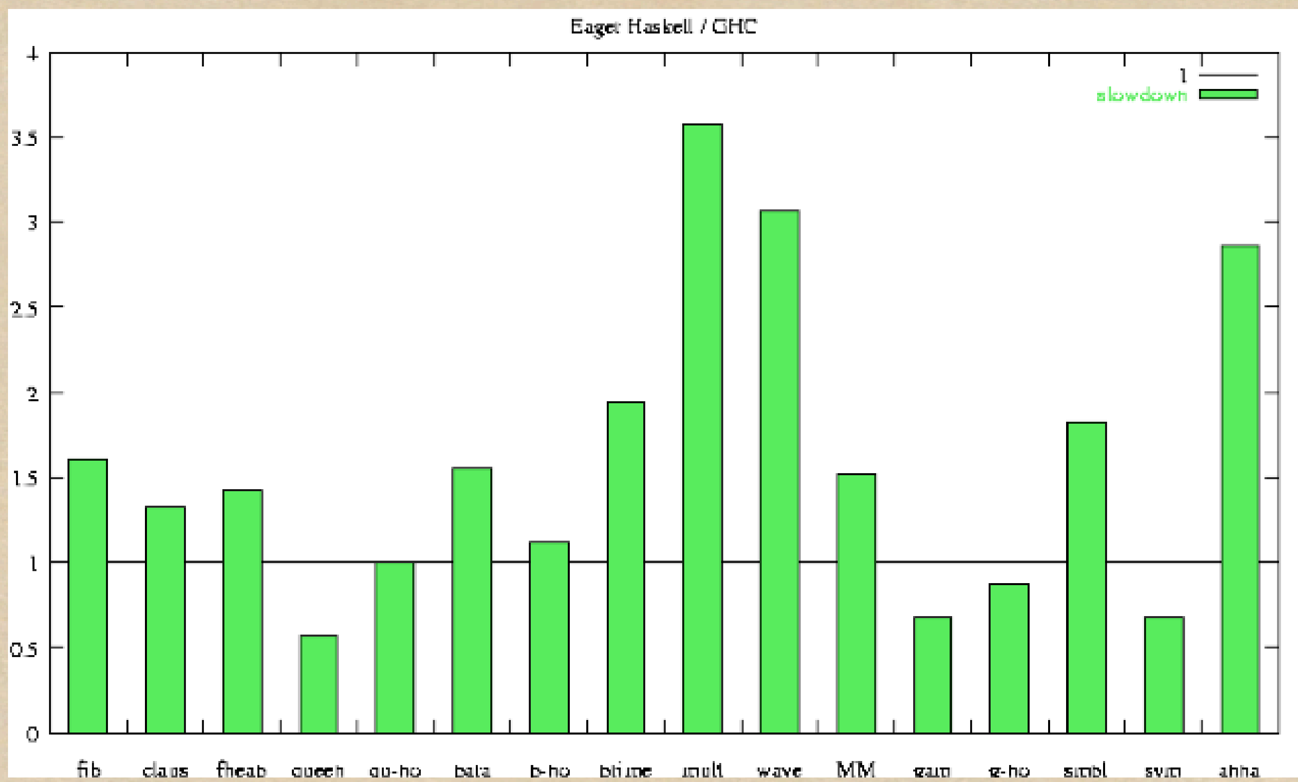
Resource-boundedness

- Allow a call to suspend for any reason
- Run Haskell code eagerly with same semantics
- Accumulating parameters in constant space!

but...

- Function call no longer returns a value
  - Use an alternate continuation / walk stack
- Indirections add unexpected synchronization
  - Even for stuff known to be computed
  - "Retry" semantics for case expressions

Did it work?



- As long as programs weren't very lazy
- Suffered from lack of man-years

Most critical lesson

Non-strictness carries a fundamental cost:

Must be ready to deal with un-computed data

- Code must check unless it knows
- There must be a mechanism to suspend
- There must be a mechanism to resume
- All else is deciding how these mechanisms work

All of the above applies to exploiting parallelism as well!

Idiom mismatches

- Idiomatic Haskell uses laziness gratuitously:

```
zip [1..] xs
take n (iterate f x)

case xs of
  (x:xs) | x < 0 = f y
         | x > 0 = g y
         | otherwise = z
         where y = ...
               z = ...
```

- We expected less code tweaking to port to pH
- Yes, language does affect how you think

Unexpected challenges

- Parallel GC
  - Absolute necessity (says Amdahl's law)
  - Requires 2-3 man years to do credibly
  - Readable version and fast version
  - Code generator only took man months!
- Dynamic linking
  - Played badly with weird control flow hacks
- Shifting language (Haskell 1.3 → 1.4 → 98 → ...)

Strictness analysis

In the late 80's, strictness analysis was going to allow us to parallelize lazy functional programs

- Tells us which expressions we must compute
- Just run those in parallel!

but...

Elaborate strictness didn't work well

- Very good at finding local dependencies
- Tells us where to serialize our code!

Atomicity concerns

- pH data representation:
  - Numbers look like valid IEEE doubles
  - Pointers look like NaNs (mask high bits)
  - Write requires load, test, Fence, CAS
  - This runs faster on multi-cores!
- Eager Haskell memory representation:
  - Write requires fence, tag update; no CAS
  - Read check combines with case expression
    - Use HW transactions to combine checks

Good ideas unexplored

- Strong classical optimization for Haskell etc
  - Strength reduction
  - Conditional rewriting
  - Partial Redundancy for arbitrary exprs?
- Specialize code based on computedness
  - Compile better code when non-strict arguments are WHNF
- Explore more strategies for resource-bounding computation

Hope for architecture

- Architecture is once again in flux
- Software folks seem to have real clout

- Hardware which would benefit everyone:
  - Transactional memory (can it scale?)
  - Hardware read/write barrier
    - GC, fast software TM, I-structures...
    - Log or trap? Under what circumstances?
  - Get rid of pipeline drains in synchronization

Onward to Fortress

- Parallel for loops, comprehensions, tupling
  - Everything looks like a reduction
  - Some reduction operators involve effects
  - Commutativity, associativity, idempotence
- Generators: parallel unfold (must deforest)
  - Track orderedness, uniqueness
- Equational manipulation in libraries if possible
  - Cross products, simple nesting
  - Data dependent nested generators?

Rogue's Gallery

Arvind

Rishiyur S. Nikhil

Lennart Augustsson

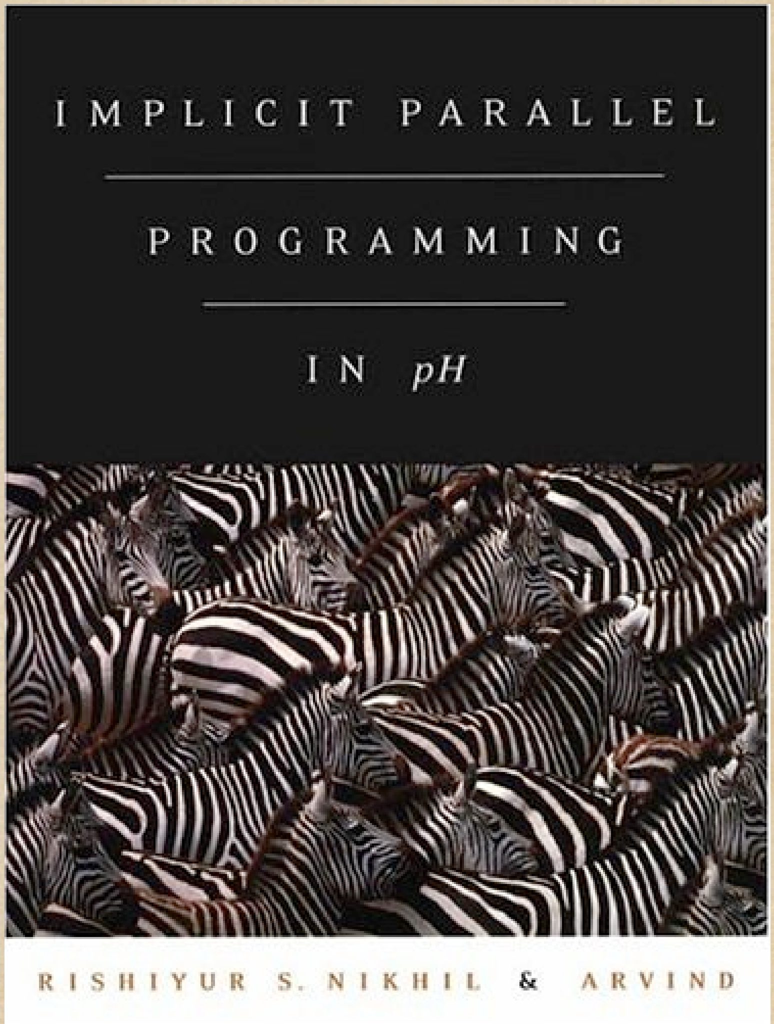
Jan-Willem Maessen

Alejandro Caro

Jacob Schwartz

Mieszko Lis

Joe Stoy



Non-suspensive threads

- Groups of instructions can run together safely
  - They share input dependencies
  - Or contribute to the same outputs
  - Group them into non-suspensive threads
- Compile these for von Neumann architectures
  - I-structure access breaks a thread
  - Very fine-grained, ~10instrs/thread
- Compiler's goal: biggest possible threads.