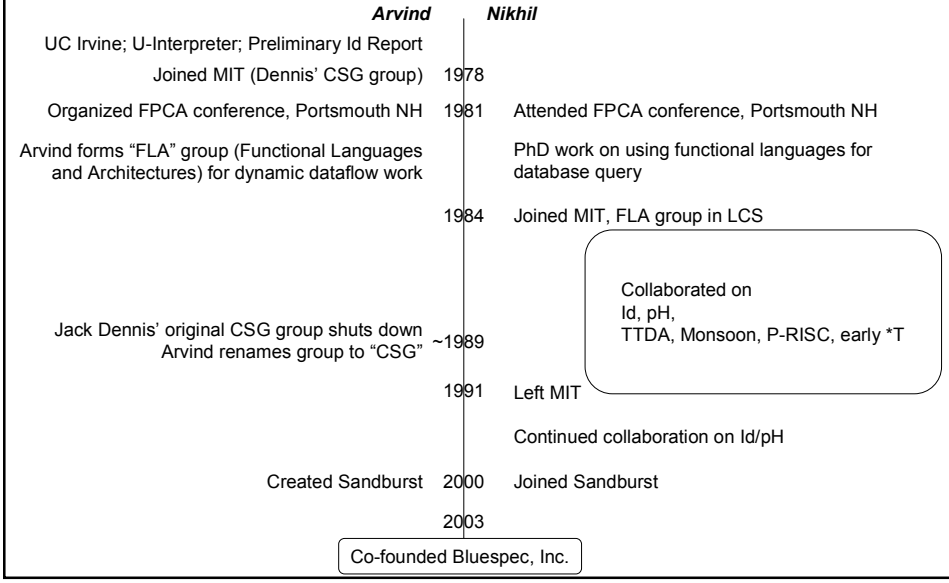
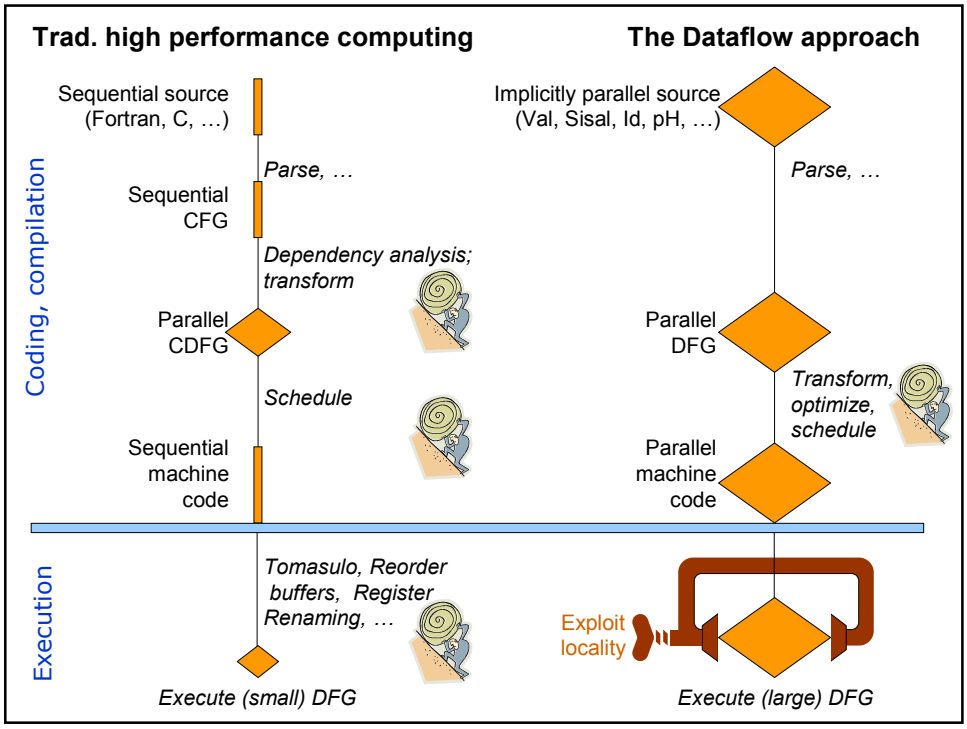
"TTDA, MEF, Id/pH, Monsoon, P-RISC"

(roughly 1978-1991)

A personal recollection
Rishiyur S. Nikhil
at "Dataflow to Synthesis Retrospective"
in celebration of Arvind's 60th
May 18, 2007

Intersection of "Nick Hill's" career with that of "Arvind Johnson/ Johnson Arvind"





Dataflow Graphs

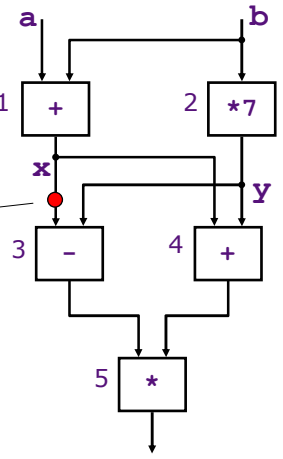
```
{x = a + b;
 y = b * 7
in
    (x-y) * (x+y)}
```

- Values in dataflow graphs are represented as tokens

token < ip , p , v >

instruction ptr, port, data

ip = 3
p = L



- An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators

no separate control flow

An early dataflow quote

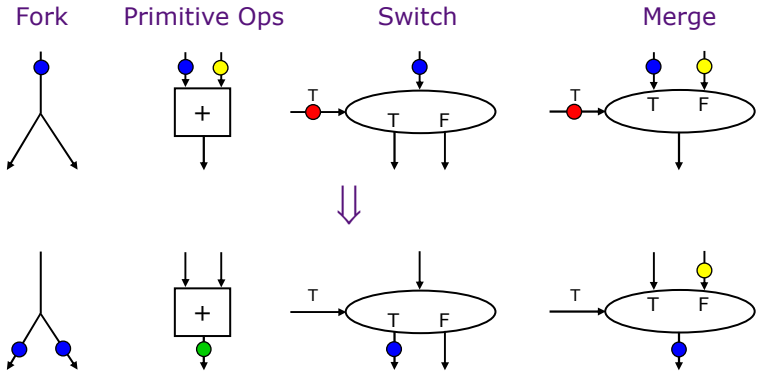
"You may fire when ready, Gridley"

Commodore George Dewey
USS Olympia, Battle of Manila Bay

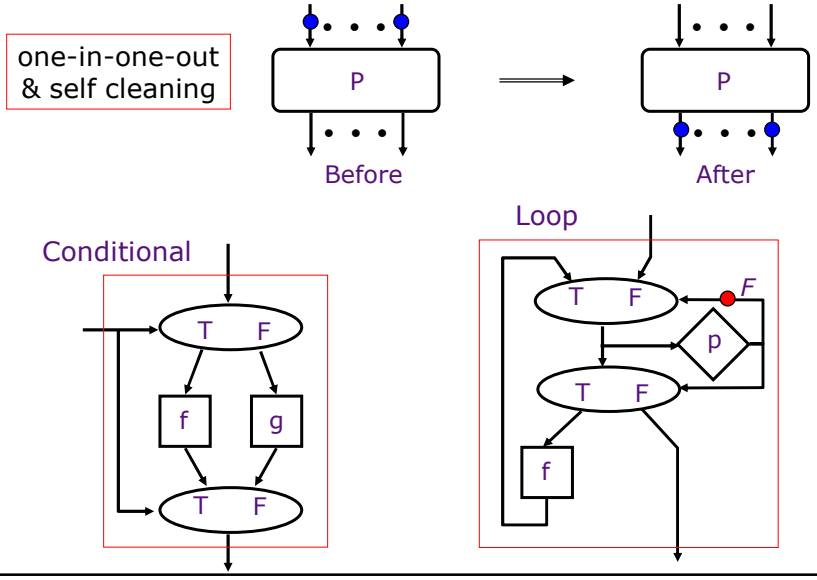
1, May 1898

Dataflow Operators

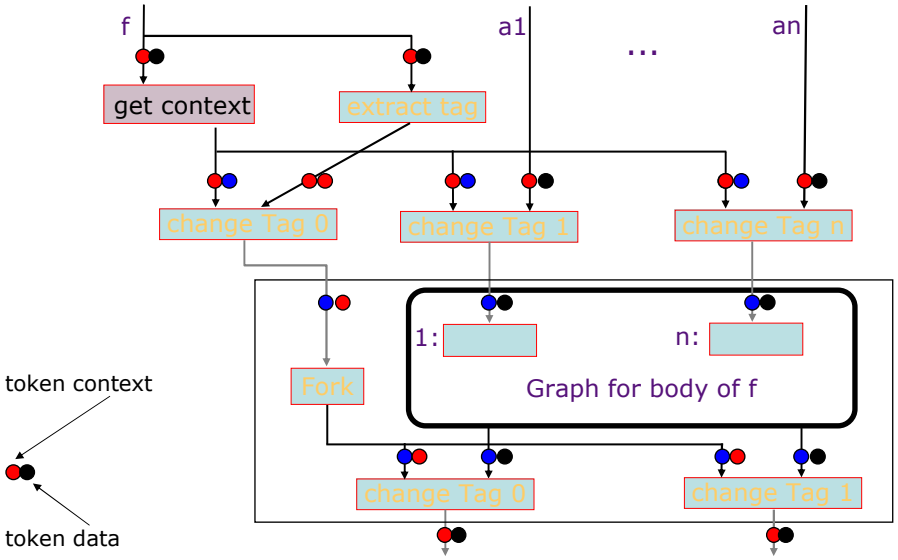
- A small set of dataflow operators can be used to define a general programming language



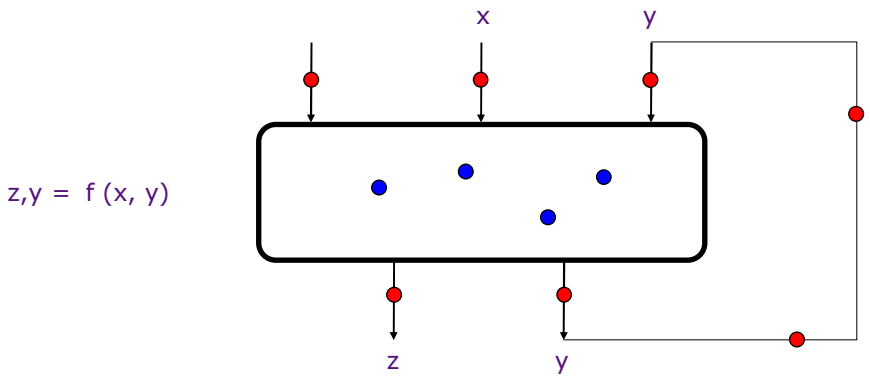
Well Behaved Schemas



Function call: f (a1,…,an)



Non-strictness



Function can execute, and even return results, before args available. Results can even be fed back as args.

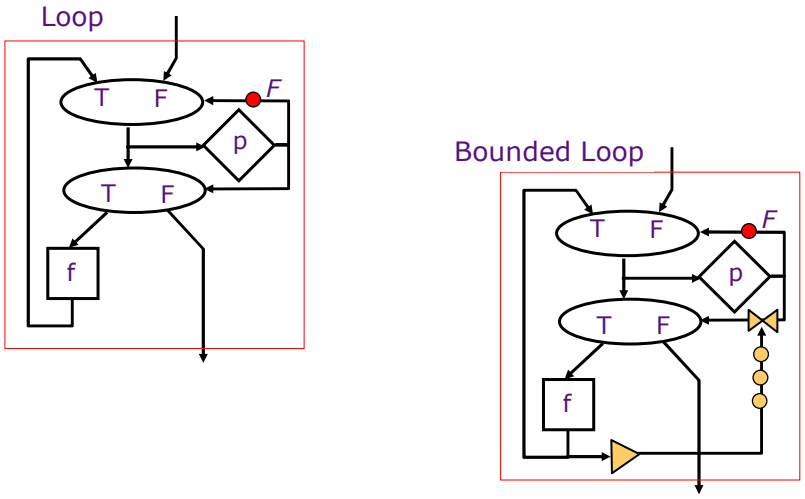
Still one-in-one-out & self cleaning!

Non-strictness: pro and con

- Pro: unleashes even more parallelism
- Con: results in even less locality
  - In traditional sequential computing, we are familiar with the notion of a *stack of frames,* of which only the topmost frame is "active"
  - In the dataflow model, functions can be called in parallel ➔ *tree of frames*
  - Because of non-strictness, *all* the frames can be simultaneously active

Less locality ➔ more resources

Pure DFG approach to "throttling" parallelism



[Switch to TTDA slides from Supercomputing 93 tutorial on "Multithreaded Architectures"]

(Slides 1-7 of ArvindFest_Nikhil_PDF.pdf)

What's the difference between a dataflow machine and a peep show?

One has a Waiting-Matching Section and the other has a Mating-Watching section

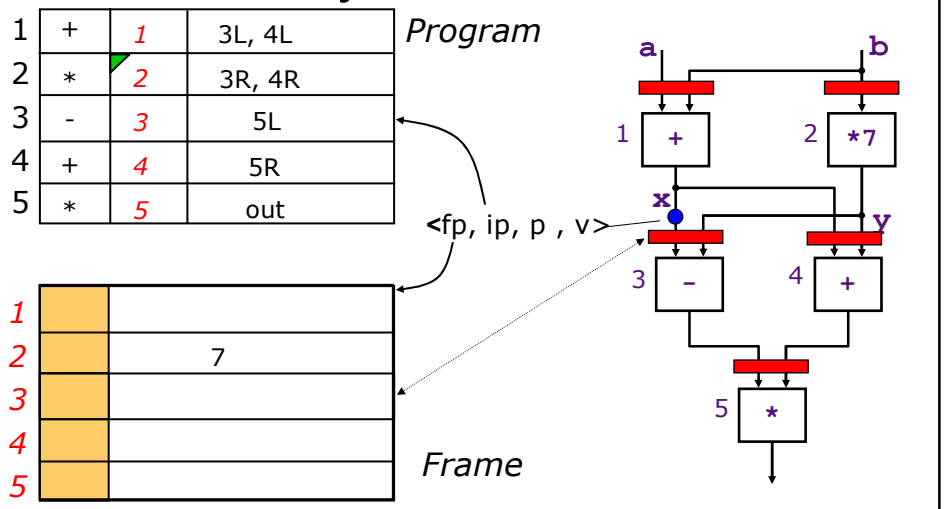
MEF
the Multiprocessor Emulation Facility

- The TTDA was never built
  - "Similar" machines were built elsewhere: Manchester, ETL Japan (Sigma-1)
- In Arvind's group, the TTDA was studied on the MEF, one of the first large-scale "computing clusters"
  - Originally planned to be Symbolics Lisp Machines ("Slimebollix", in LCS jargon)
  - Actually used TI Explorer Lisp Machines ("Exploders", in CSG jargon)

ETS: Explicit Token Stores

- The TTDA's Waiting-Matching section required a large, fast, associative memory to hold and lookup waiting tokens
- The Explicit Token Store replaced it with a directly-addressed memory, by organizing token storage into conventional "stack" frames
  - (actually a "tree" of frames, as described earlier)

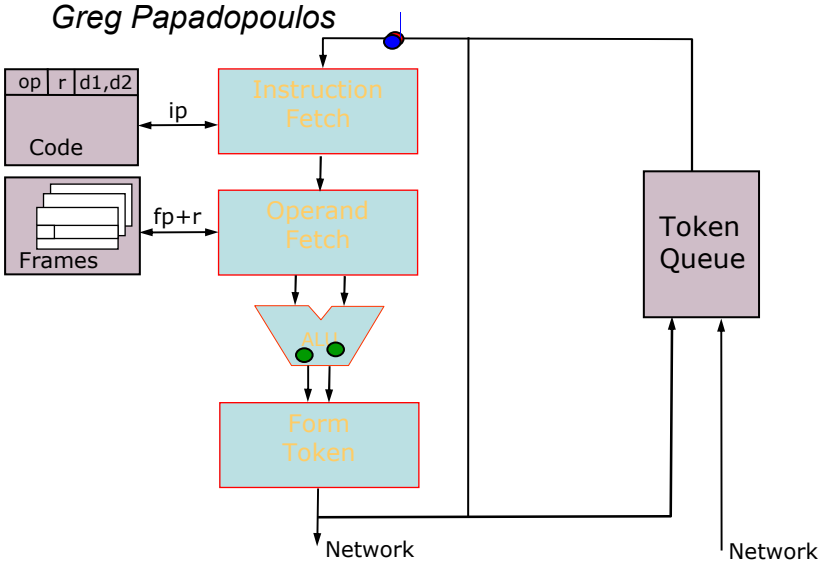
A Frame in Dynamic Dataflow



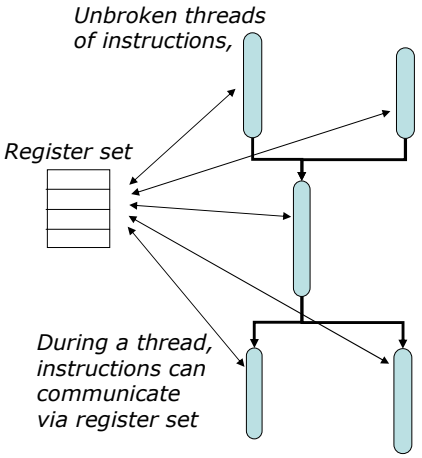
Need to provide storage for only one operand/operator

Monsoon Processor

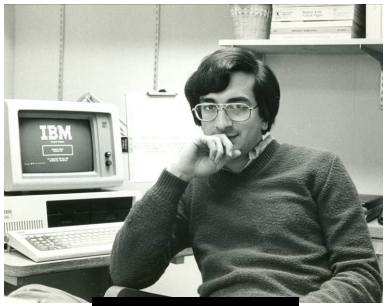
Greg Papadopoulos



Introducing threading and thread-lifetime register sets



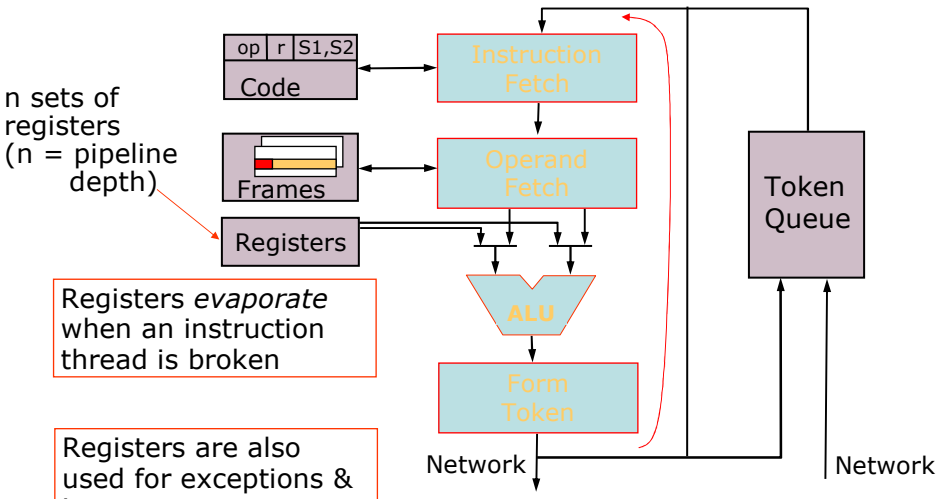
"Hybrid Dataflow/ von Neumann Architecture"



Robert Iannucci

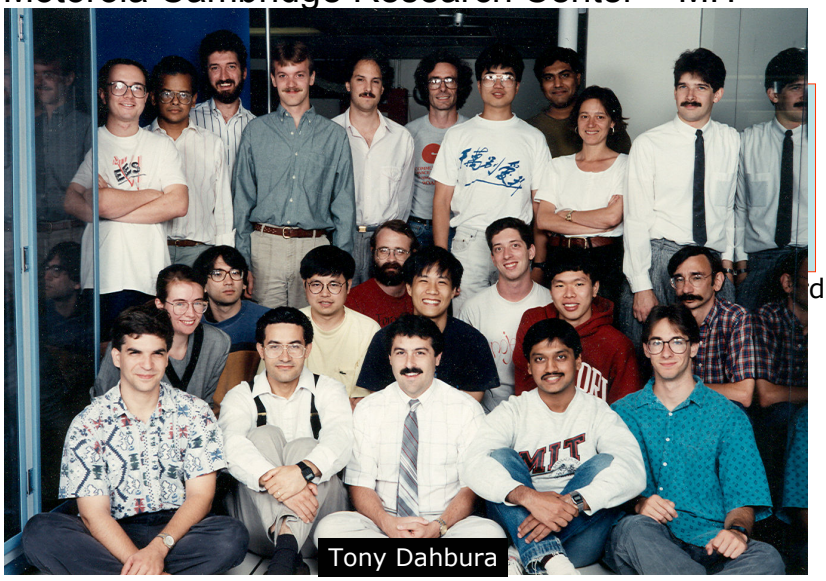
Temporary Registers & Threads

in Monsoon



The Monsoon Project

Motorola Cambridge Research Center + MIT



ds

Id World people

- Rishiyur Nikhil,
- Keshav Pingali,
- Vinod Kathail,
- David Culler
- Ken Traub
- Steve Heller,
- Richard Soley,
- Dinart Mores
- Jamey Hicks,
- Alex Caro,
- Andy Shaw,
- Boon Ang
- Shail Aditya
- R Paul Johnson
- Paul Barth
- Jan Maessen
- Christine Flood
- Jonathan Young
- Derek Chiou
- Arun Iyangar
- Zena Ariola
- Mike Bekerle

- K. Eknadham (IBM)
- Wim Bohm (Colorado)
- Joe Stoy (Oxford)
- ...


R.S. Nikhil


Keshav Pingali


David Culler


Ken Traub




Boon S. Ang Jamey Hicks Derek Chiou


Steve Heller

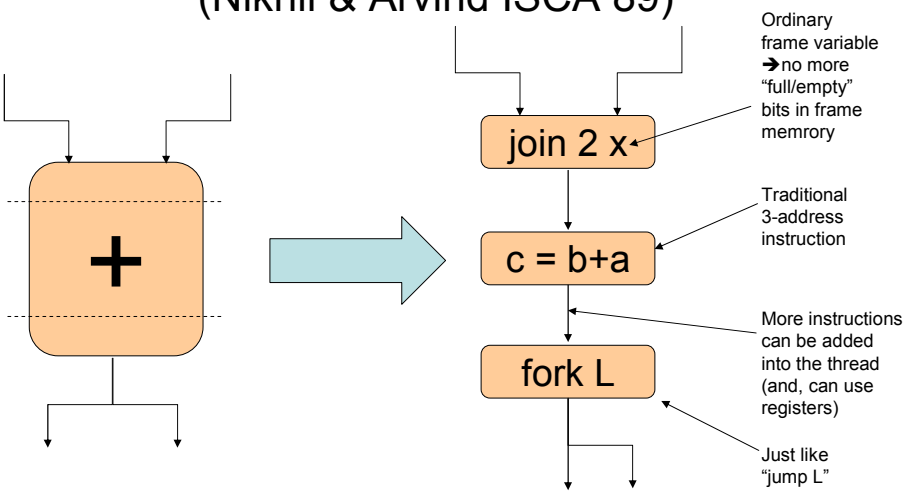
Id Applications on Monsoon @ MIT

- Numerical
  - Hydrodynamics - SIMPLE
  - Global Circulation Model - GCM
  - Photon-Neutron Transport code -GAMTEB
  - N-body problem
- Symbolic
  - Combinatorics - free tree matching,Paraffins
  - Id-in-Id compiler
- System
  - I/O Library
  - Heap Storage Allocator on Monsoon
- Fun and Games
  - Breakout
  - Life
  - Spreadsheet

The Monsoon Experience

- Performance of implicitly parallel Id programs scaled effortlessly.
- Id programs on a single-processor Monsoon took 2 to 3 times as many cycles as Fortran/C on a modern workstation.
  - Can certainly be improved
- Effort to develop the invisible software (loaders, simulators, I/O libraries,....) dominated the effort to develop the visible software (compilers...)

After the Monsoon
P-RISC: "RISCifying" dataflow
(Nikhil & Arvind ISCA 89)



From P-RISC to *T (“starT”)
[Nikhil, Papadopoulos, Arvind, 1991]

- P-RISC, in turn, led to the idea of augmenting a standard processor with a “synchronizing” processor that simply took care of the “dataflow” instructions:
  - Join, fork, split-phase loads, …
- Derek Chiou will explore this in more detail in the next talk

[See Unification slide from Supercomputing 93 tutorial on “Multithreaded Architectures”]

(Slide 8 of ArvindFest_Nikhil_PDF.pdf)

What did the palindrome-loving computer architect say when he understood the dataflow approach?

"Sinned was I ere I saw Dennis"

Multiple threads of research in FLA/CSG

Architectures

Formally defined abstract machines

"Pure" dataflow (TTDA)

Synchronizing memory (I- and M-structures

Threading, for locality (Hybrid, Monsoon, P-RISC)

Unification/integration into traditional processors (*T)

Parallel Functional Languages (Id, pH) and Compilation

DFG semantics, efficient demand-driven evaluation

Optimal reduction in lambda calculus

Resource-bounded DFGs

Thread formation under non-strictness

Parallel synchronizing data structures: I-structures (confluent) and M-structures (not)

Formal Semantics using Rewrite Rules (P-TAC, Lambda-S, …)

"Full featured" language

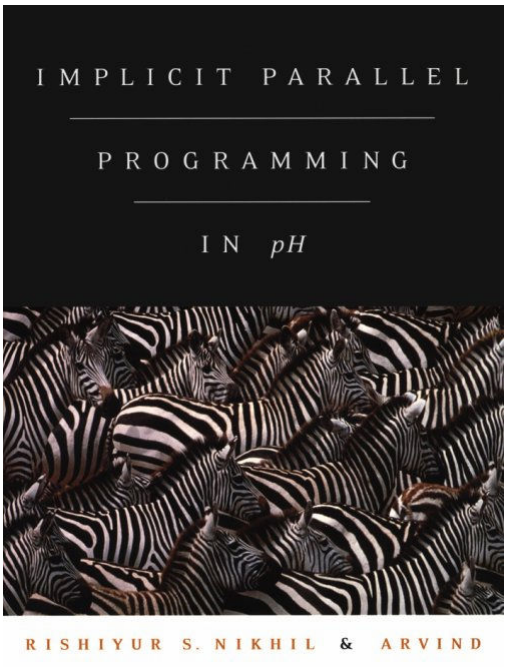
Programming environment for studying parallelism (Id World)

The many versions of Id

- *The (preliminary) Id Report* [Arvind, Gostelow and Plouffe 78, UC Irvine]
  - Textual language "derived" from dataflow graphs
- *Id/83s* [Nikhil & Arvind 85], *Id Nouveau* [Nikhil, Pingali & Arvind 86]
  - Major redesign for teaching 6.83s summer course
  - Functional subset inspired by ML
  - Plus, incorporate loops, I-structures, from original Id
  - Operational semantics defined with DFGs, graph-reduction, and rewrite rules
- *Id 88.0, Id 88.1* [Nikhil 88]
  - Hindley-Milner polymorphic types, pattern matching
- *Id 90.0* [Nikhil 90], *Id 90.1* [Nikhil 91]
  - Overloading, array comprehensions, accumulators, abstract types, explicit sequencing, M-structures, delayed evaluation, bounded loops, pragmas, I/O, "systems programming" features

From Id to pH

- In 1990, many functional programming research groups had their own "lazy functional programming language"
  - All the same, modulo silly syntactic differences
- The functional subset of Id had the same *non-strict* semantics as lazy languages, although it did not use lazy evaluation because of I- and M-structures
- We, along with researchers from many of these groups decided to "standardize" on a common non-strict functional language. This is what became todays *Haskell*
- In CSG, we worked on a variant called *pH,* for "parallel Haskell", which had Haskell syntax, extensions for I- and M-structures and sequencing, and dataflow execution



Morgan Kaufmann 2001

Arvind and I started writing this book ~1988!

Summary

It's been a terrifically exciting time at CSG/FLA/CSG

The dataflow approach was beautiful,
and still captures the interest of the research communities in high performance computing and declarative languages.

Arvind keynote talk at ISCA 2005, Madison, WI

Dataflow: Passing the Token

Why do they invite a dataflow person to ISCA?

Because they need a token minority

Perhaps, as we now move into the era of multi-cores and SoCs, it's time to evaluate some of these ideas again!

Thank you!