"TTDA, MEF, Id/pH, Monsoon, P-RISC"
(roughly 1978-1991)

A personal recollection
Rishiyr 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”

<table>
<thead>
<tr>
<th>Arvind</th>
<th>Nikhil</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC Irvine; U-Interpreter; Preliminary Id Report</td>
<td>1978</td>
</tr>
<tr>
<td>Joined MIT (Dennis’ CSG group)</td>
<td>1981</td>
</tr>
<tr>
<td>Organized FPCA conference, Portsmouth NH</td>
<td>1981</td>
</tr>
<tr>
<td>Arvind forms “FLA” group (Functional Languages and Architectures) for dynamic dataflow work</td>
<td>1984</td>
</tr>
<tr>
<td>Jack Dennis’ original CSG group shuts down</td>
<td>1989</td>
</tr>
<tr>
<td>Arvind renames group to “CSG”</td>
<td>1991</td>
</tr>
<tr>
<td>Left MIT</td>
<td>2000</td>
</tr>
<tr>
<td>Collaborated on Id, pH, TTDA, Monsoon, P-RISC, early &quot;T&quot;</td>
<td>2003</td>
</tr>
<tr>
<td>Created Sandburst</td>
<td>2000</td>
</tr>
<tr>
<td>Joined Sandburst</td>
<td>2003</td>
</tr>
<tr>
<td>Co-founded Bluespec, Inc.</td>
<td>2003</td>
</tr>
</tbody>
</table>

Collaborated on Id, pH, TTDA, Monsoon, P-RISC, early "T"
Dataflow Graphs

\[
\begin{align*}
\{ & x = a + b; \\
& y = b * 7 \\
\text{in} & (x-y) * (x+y) \}
\end{align*}
\]

- Values in dataflow graphs are represented as tokens

\[
\text{token } < \text{ip}, p, v >
\]

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

\[
\text{no separate control flow}
\]
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

Conditional

T F
f g
T F

Loop

T F
f
T F
f

Function call: \( f(a_1, \ldots, a_n) \)
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)
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td>5</td>
</tr>
</tbody>
</table>

3L, 4L 3R, 4R 5L 5R out

Program

Frame

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

- Registers evaporate when an instruction thread is broken
- Registers are also used for exceptions & interrupts

n sets of registers ($n = \text{pipeline depth}$)

The Monsoon Project

Motorola Cambridge Research Center + MIT
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 Anditya
- R Paul Johnson
- Paul Barth
- Jan Maessen
- Christine Flood
- Jonathan Young
- Derek Chiou
- Arun Iyangar
- Zena Ariola
- Mike Bekerle
- K. Elkadham (IBM)
- Wim Bohm (Colorado)
- Joe Stoy (Oxford)
- ...

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?

“This sinned was I ere I saw Dennis”
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 today’s 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
Arvind and I started writing this book ~1988!

Dataflow: Passing the Token

Arvind keynote talk at ISCA 2005, Madison, WI

**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!