CS 220: Survey of Programming Languages

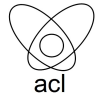
Lecture Slides

Dataflow Programming Languages

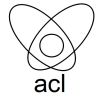
Jan Michael C. Yap

Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines, Diliman
jcyap@dcs.upd.edu.ph

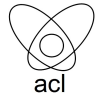
Session 9



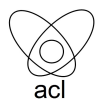
Dataflow Programming Languages



Dataflow Programming Languages



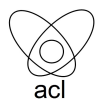
How we usually view computation...



http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$



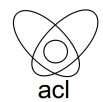
http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

```
z  = a * b  + c
w = z + 4
y =  z * z - (3 * z + b)
```



http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

```
z  = a * b  + c
w = z + 4
y =  z * z - (3 * z + b)
```

- Aim is to change the state of the variables by focusing on the operations needed to be performed.

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

```
z  = a * b  + c
w = z + 4
y =  z * z - (3 * z + b)
```

- Aim is to change the state of the variables by focusing on the operations needed to be performed.
  - Follow rules of precedence and associativity of operators

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

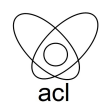
How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

```
z  = a * b  + c
w = z + 4
y =  z * z - (3 * z + b)
```

- Aim is to change the state of the variables by focusing on the operations needed to be performed.
  - Follow rules of precedence and associativity of operators
  - Sequentially apply operations based on order of statements/expressions



http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

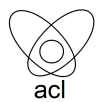
How we usually view computation...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

```
z  = a * b  + c
w = z + 4
y =  z * z - (3 * z + b)
```

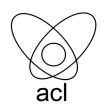
- Aim is to change the state of the variables by focusing on the operations needed to be performed.
  - Follow rules of precedence and associativity of operators
  - Sequentially apply operations based on order of statements/expressions
- (Sequential) Operator/Function-centric view of computation



http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Changing our perspective...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

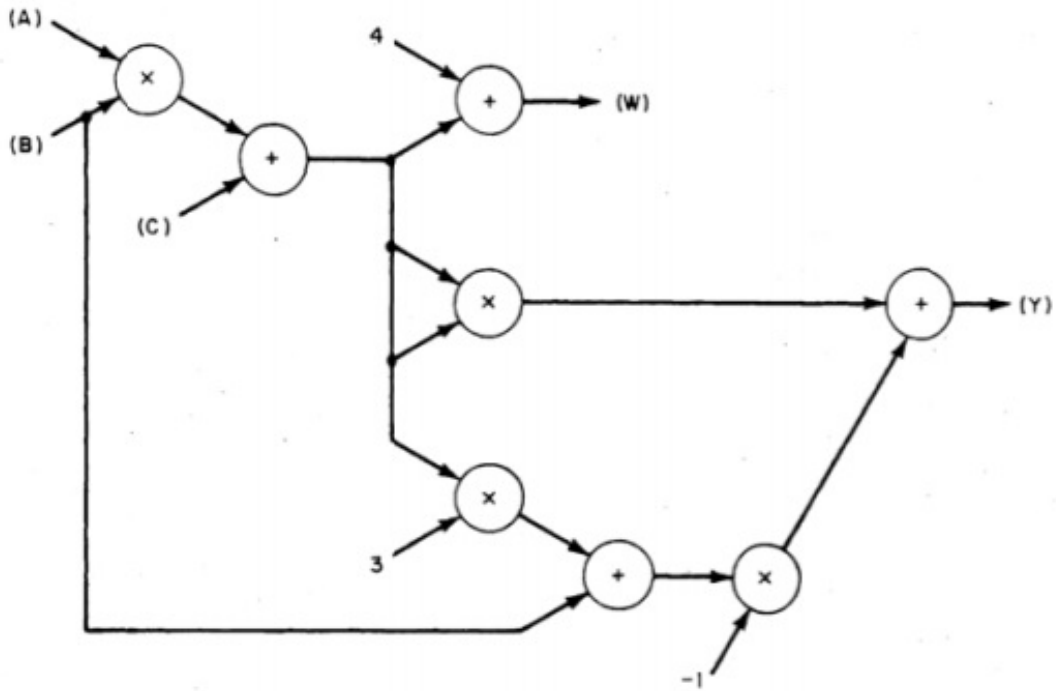


http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

Changing our perspective...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$

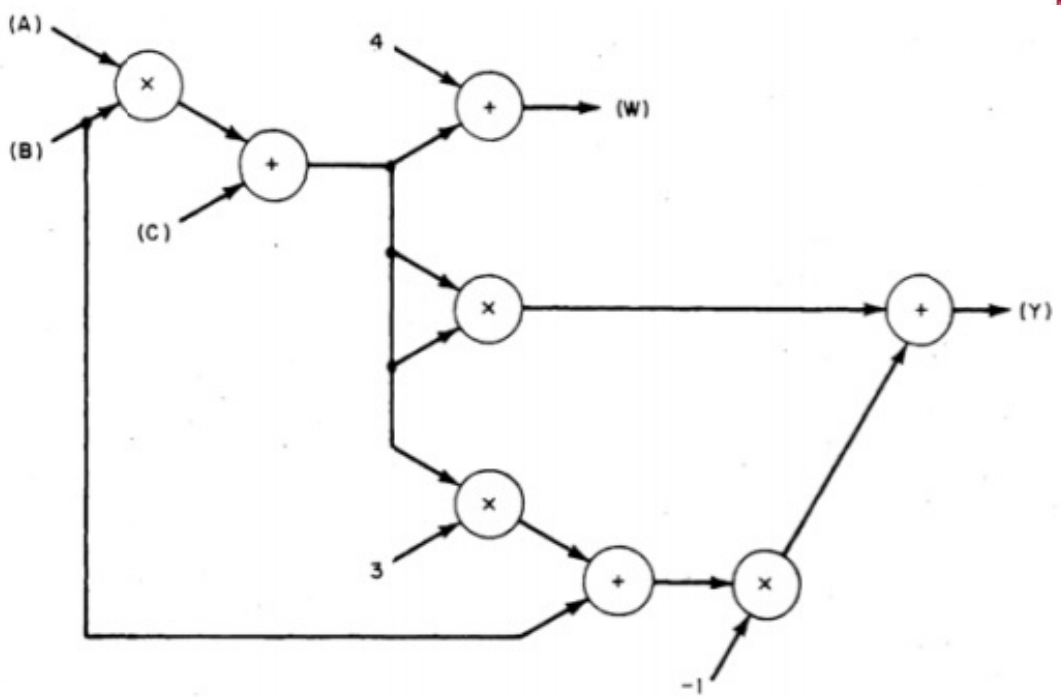


http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Changing our perspective...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$



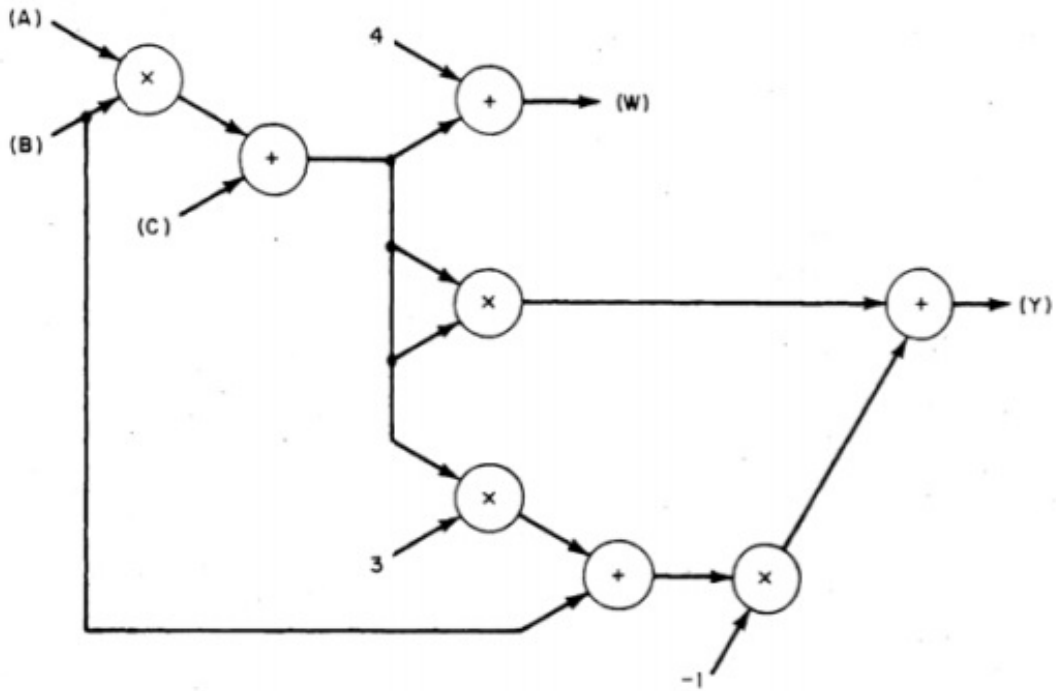
- Dataflow graph

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Changing our perspective...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

$\Rightarrow$



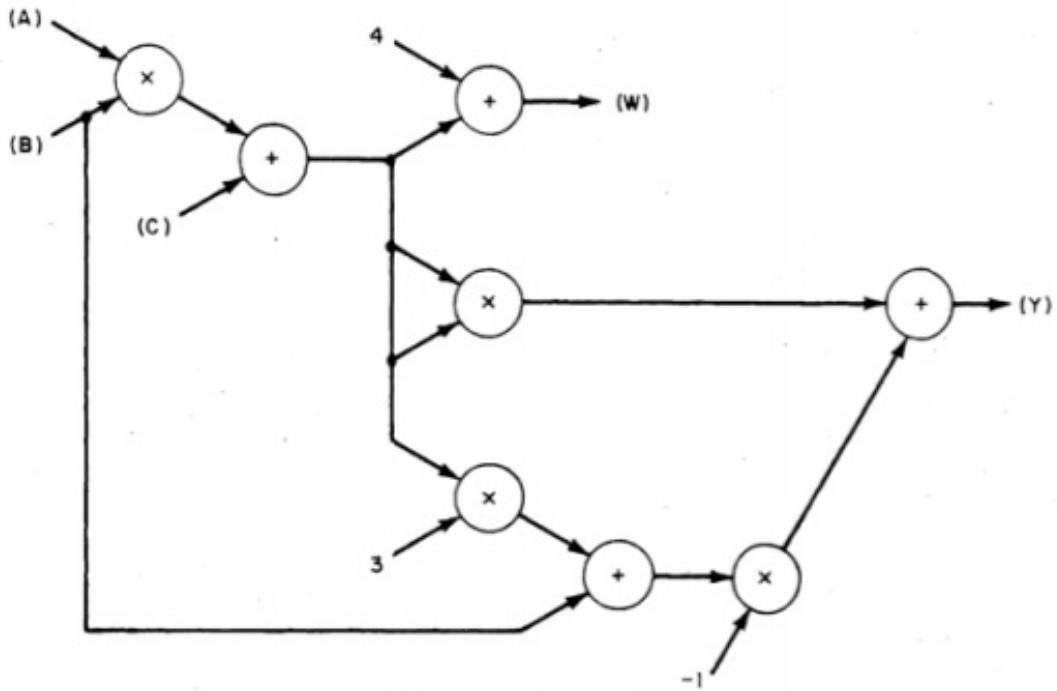
- Dataflow graph
- From focusing on which operator to operate first to focus on which variables/data to move towards an operation

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Changing our perspective...

$$Z = A \times B + C$$
$$W = Z + 4$$
$$Y = Z^2 - (3Z + B)$$

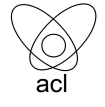
$\Rightarrow$



- Dataflow graph
- From focusing on which operator to operate first to focus on which variables/data to move towards an operation
- Possible parallelization of independent operations

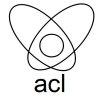
http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Characteristics of a dataflow PL



Characteristics of a dataflow PL

- Computation is represented internally as a dataflow graph

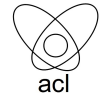


Characteristics of a dataflow PL

- Computation is represented internally as a dataflow graph
- Data are represented as tokens created by nodes, placed on (output) arcs, and removed when they are accessed as input to other nodes

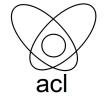
Characteristics of a dataflow PL

- Computation is represented internally as a dataflow graph
- Data are represented as tokens created by nodes, placed on (output) arcs, and removed when they are accessed as input to other nodes
- Control structures are represented as valves



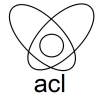
Characteristics of a dataflow PL

- Computation is represented internally as a dataflow graph
- Data are represented as tokens created by nodes, placed on (output) arcs, and removed when they are accessed as input to other nodes
- Control structures are represented as valves
  - Selector valves take in 2 input tokens and uses a (Boolean) control token to select one of the inputs for outputting to an arc

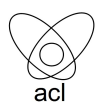


Characteristics of a dataflow PL

- Computation is represented internally as a dataflow graph
- Data are represented as tokens created by nodes, placed on (output) arcs, and removed when they are accessed as input to other nodes
- Control structures are represented as valves
  - Selector valves take in 2 input tokens and uses a (Boolean) control token to select one of the inputs for outputting to an arc
  - Distributor valves take in an input token and uses a (Boolean) control token to send the input to one of 2 possible output arcs (representing "true" and "false")



Characteristics of a dataflow PL



[3] http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

Characteristics of a dataflow PL

- Implementation of dataflow PLs usually entail alternative hardware architecture

[3] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Characteristics of a dataflow PL

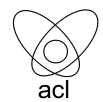
- Implementation of dataflow PLs usually entail alternative hardware architecture
  - That is why dataflow PLs are usually for embedded systems applications development

[3] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Characteristics of a dataflow PL

- Implementation of dataflow PLs usually entail alternative hardware architecture
  - That is why dataflow PLs are usually for embedded systems applications development
- Advantages[3]

[3] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf



Characteristics of a dataflow PL

- Implementation of dataflow PLs usually entail alternative hardware architecture
  - That is why dataflow PLs are usually for embedded systems applications development
- Advantages[3]
  - Inherent concurrency

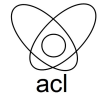
[3] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Characteristics of a dataflow PL

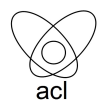
- Implementation of dataflow PLs usually entail alternative hardware architecture
  - That is why dataflow PLs are usually for embedded systems applications development
- Advantages[3]
  - Inherent concurrency
  - Existence of visual implementations of dataflow PLs

[3] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Dataflow Programming Languages



Background

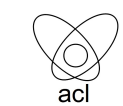


---

http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf
http://csg.csail.mit.edu/Users/dennis/barc-presentation/sld015.htm

Background

- Streams and Iteration in a Single Assignment Language



---

http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf
http://csg.csail.mit.edu/Users/dennis/barc-presentation/sld015.htm

Background

- Streams and Iteration in a Single Assignment Language
- Developed in 1985 as a derivative of VAL (Variable Assembly Language)

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf
http://csg.csail.mit.edu/Users/dennis/barc-presentation/sld015.htm

Background

- Streams and Iteration in a Single Assignment Language
- Developed in 1985 as a derivative of VAL (Variable Assembly Language)
- Language has Pascal-like syntax

http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf
http://csg.csail.mit.edu/Users/dennis/barc-presentation/sld015.htm

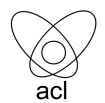
Background

- Streams and Iteration in a Single Assignment Language
- Developed in 1985 as a derivative of VAL (Variable Assembly Language)
- Language has Pascal-like syntax
- "The language intended to compete in performance with Fortran while using the dataflow model to introduce parallel computation in the first multi-core machines."
  - "... [M]icro-tasking environment [was provided to support] dataflow architecture on traditional single-core machines."

http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf
http://csg.csail.mit.edu/Users/dennis/barc-presentation/sld015.htm

Evaluation

- Data types[4]



[4]Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

Evaluation

- Data types[4]
  - Integer, single and double precision floating point real numbers, Boolean, and character primitive data types

[4] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

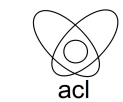
Evaluation

- Data types[4]
  - Integer, single and double precision floating point real numbers, Boolean, and character primitive data types
  - Arrays, streams, records and unions as derived types

[4] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

Evaluation

- Data types[4]
  - Integer, single and double precision floating point real numbers, Boolean, and character primitive data types
  - Arrays, streams, records and unions as derived types
    - Arrays have dynamic bounds



[4]Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

Evaluation

- Data types[4]
  - Integer, single and double precision floating point real numbers, Boolean, and character primitive data types
  - Arrays, streams, records and unions as derived types
    - Arrays have dynamic bounds
    - Streams are ordered, homogenous sequence/list of values, components of which are not indexed (as with arrays)

[4] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

Evaluation

- Data types[4]
  - Integer, single and double precision floating point real numbers, Boolean, and character primitive data types
  - Arrays, streams, records and unions as derived types
    - Arrays have dynamic bounds
    - Streams are ordered, homogenous sequence/list of values, components of which are not indexed (as with arrays)
    - Union types are choice types

[4] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, pp. 112-115

Evaluation

- Syntax design (Based on the grammar of a Javascript interpreter for SISAL[5]):

[5] https://github.com/parsifal-47/sisal-js/blob/master/helpful_data/sisal30bnf.txt#
[6] Manually counted
[7] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/16.Seq.loops.html

Evaluation

- Syntax design (Based on the grammar of a Javascript interpreter for SISAL[5]):
  - Production rules: 98[6]
  - Keywords: $\geq$ 33[6]

[5] https://github.com/parsifal-47/sisal-js/blob/master/helpful_data/sisal30bnf.txt#

[6] Manually counted

[7] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/16.Seq.loops.html

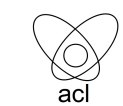
Evaluation

- Syntax design (Based on the grammar of a Javascript interpreter for SISAL[5]):
  - Production rules: 98[6]
  - Keywords: ≥ 33[6]
  - While it has Pascal-like syntax, SISAL no longer has the uniformity issues of Pascal, particularly the `repeat` issue[7]

[5] https://github.com/parsifal-47/sisal-js/blob/master/helpful_data/sisal30bnf.txt#

[6] Manually counted

[7] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/16.Seq.loops.html



Evaluation

- Abstraction:

[8] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 113-115

[9] http://www2.cmp.uea.ac.uk/∼jrwg/Sisal/14.Funcs.progs.html

[10] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 115

Evaluation

- Abstraction:
  - Any of the derived types allow for creation of data structures[8]
    - But union types are more popular (recommended?)

[8] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 113-115

[9] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/14.Funcs.progs.html

[10] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 115

Evaluation

- Abstraction:
  - Any of the derived types allow for creation of data structures[8]
    - But union types are more popular (recommended?)
  - Access to functions and variables is similar to a protected mode of access in OO/imperative languages by default, but can be made public via the `global` keyword[9]

[8] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 113-115

[9] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/14.Funcs.progs.html

[10] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 115

Evaluation

- Abstraction:
  - Any of the derived types allow for creation of data structures[8]
    - But union types are more popular (recommended?)
  - Access to functions and variables is similar to a protected mode of access in OO/imperative languages by default, but can be made public via the `global` keyword[9]
- Expressivity: Has no built-in libraries (save for the core), but can access native libraries of operating systems [9]

[8] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 113-115

[9] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/14.Funcs.progs.html

[10] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 115

Evaluation

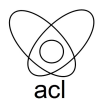
- Abstraction:
  - Any of the derived types allow for creation of data structures[8]
    - But union types are more popular (recommended?)
  - Access to functions and variables is similar to a protected mode of access in OO/imperative languages by default, but can be made public via the `global` keyword[9]
- Expressivity: Has no built-in libraries (save for the core), but can access native libraries of operating systems [9]
- Exception handling: Done by (explicitly) generating error values, and preserving component values known to be correct prior to generation of error values. [10]

[8] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 113-115

[9] http://www2.cmp.uea.ac.uk/~jrwg/Sisal/14.Funcs.progs.html

[10] Proceedings of Future Parallel Computers: An Advanced Course in Pisa, Italy, June 9-20, 1986, p. 115

Evaluation



[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]

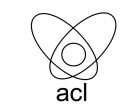
[11] RA Finkel, Chapter 6
[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]

[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf

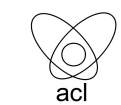


Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking

[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/∼prodei/dsie12/papers/paper_17.pdf



Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking
  - "... SISAL's compiler was able to distribute computation between nodes in an optimized way."

[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking
  - "... SISAL's compiler was able to distribute computation between nodes in an optimized way."
    - Compiler is responsible for building the dataflow graph

[11] RA Finkel, Chapter 6

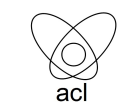
[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking
  - "... SISAL's compiler was able to distribute computation between nodes in an optimized way."
    - Compiler is responsible for building the dataflow graph
    - Each node is executed by an independent thread

[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

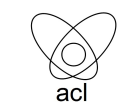


Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking
  - "... SISAL's compiler was able to distribute computation between nodes in an optimized way."
    - Compiler is responsible for building the dataflow graph
    - Each node is executed by an independent thread
    - Data processing is done upon arrival at a node, and forwarded along the dataflow chain

[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf



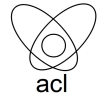
Evaluation

- Restricted aliasing: Since it is a derivative of VAL, aliasing is not allowed at all in SISAL[11]
- Efficiency [12]
  - Employs static type-checking
  - "... SISAL's compiler was able to distribute computation between nodes in an optimized way."
    - Compiler is responsible for building the dataflow graph
    - Each node is executed by an independent thread
    - Data processing is done upon arrival at a node, and forwarded along the dataflow chain
  - "In some benchmarks, SISAL was able to outperform Fortran in computation performance"

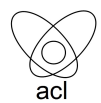
[11] RA Finkel, Chapter 6

[12] http://paginas.fe.up.pt/~prodei/dsie12/papers/paper_17.pdf

Dataflow Programming Languages



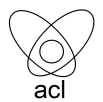
Background



http://www.tau.ac.il/∼flaxer/edu/course/vhdl/slides/VHDL01.pdf

Background

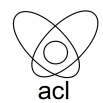
- Hardware design languages (HDLs) allow for quick design, implement, test, and document, complex digital systems



http://www.tau.ac.il/∼flaxer/edu/course/vhdl/slides/VHDL01.pdf

Background

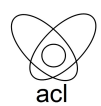
- Hardware design languages (HDLs) allow for quick design, implement, test, and document, complex digital systems
- VHDL stands for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language



http://www.tau.ac.il/~flaxer/edu/course/vhdl/slides/VHDL01.pdf

Background

- Hardware design languages (HDLs) allow for quick design, implement, test, and document, complex digital systems
- VHDL stands for Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
- Created by the US Department of Defense to document military designs for portability, a standard of the language of which was adopted in 1987



http://www.tau.ac.il/~flaxer/edu/course/vhdl/slides/VHDL01.pdf

Evaluation

- Data types[13]

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf
[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html
[15] Manually counted

Evaluation

- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf
[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html
[15] Manually counted

Evaluation

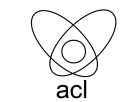
- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types
  - Arrays and records as derived types

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf
[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html
[15] Manually counted

Evaluation

- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types
  - Arrays and records as derived types
- Syntax design[14]:

[13] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html
[15] Manually counted



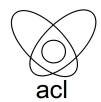
Evaluation

- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types
  - Arrays and records as derived types
- Syntax design[14]:
  - Production rules: 238[15]
  - Keywords: ∼116[16]

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf

[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html

[15] Manually counted



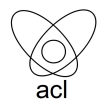
Evaluation

- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types
  - Arrays and records as derived types
- Syntax design[14]:
  - Production rules: 238[15]
  - Keywords: ∼116[16]
  - Think building ICs *in silico*: you only have "gates" which perform operations orthogonal from each other with flow directors, and then build everything on top of that

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf

[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html

[15] Manually counted



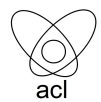
Evaluation

- Data types[13]
  - Integers, bits, characters, logical, and enumeration as primitive types
  - Arrays and records as derived types
- Syntax design[14]:
  - Production rules: 238[15]
  - Keywords: ∼116[16]
  - Think building ICs *in silico*: you only have "gates" which perform operations orthogonal from each other with flow directors, and then build everything on top of that
  - Comparable to assembly language programming

[13] http://cseweb.ucsd.edu/∼tweng/cse143/VHDLReference/04.pdf

[14] http://tams-www.informatik.uni-hamburg.de/vhdl/tools/grammar/vhdl93-bnf.html

[15] Manually counted



Evaluation

- Abstraction: “Not all operations in your design can be shared...Operations can be shared only if they are in the same process.”[16]

[16] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/09.pdf
[17] http://www.csee.umbc.edu/portal/help/VHDL/samples/samples.html
[18] http://www.eda.org/VIUF_proc/Spring95/BERGERON95A.PDF

Evaluation

- Abstraction: "Not all operations in your design can be shared...Operations can be shared only if they are in the same process."[16]
- Expressivity: Comparable to assembly language programming, plus libraries for handling (basic) I/O[17]

[16] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/09.pdf
[17] http://www.csee.umbc.edu/portal/help/VHDL/samples/samples.html
[18] http://www.eda.org/VIUF_proc/Spring95/BERGERON95A.PDF

Evaluation

- Abstraction: "Not all operations in your design can be shared...Operations can be shared only if they are in the same process."[16]
- Expressivity: Comparable to assembly language programming, plus libraries for handling (basic) I/O[17]
- Exception handling: [18]

[16] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/09.pdf
[17] http://www.csee.umbc.edu/portal/help/VHDL/samples/samples.html
[18] http://www.eda.org/VIUF_proc/Spring95/BERGERON95A.PDF

Evaluation

- Abstraction: "Not all operations in your design can be shared...Operations can be shared only if they are in the same process."[16]
- Expressivity: Comparable to assembly language programming, plus libraries for handling (basic) I/O[17]
- Exception handling: [18]
  - Use `wait on... until` structure to raise exceptions

[16] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/09.pdf
[17] http://www.csee.umbc.edu/portal/help/VHDL/samples/samples.html
[18] http://www.eda.org/VIUF_proc/Spring95/BERGERON95A.PDF

Evaluation

- Abstraction: "Not all operations in your design can be shared...Operations can be shared only if they are in the same process."[16]
- Expressivity: Comparable to assembly language programming, plus libraries for handling (basic) I/O[17]
- Exception handling: [18]
  - Use `wait on... until` structure to raise exceptions
  - Best practice for exception handling: "...[I]ntroduce an explicit [outer] loop and the outset of the process which handles normal processing."

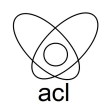
[16] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/09.pdf
[17] http://www.csee.umbc.edu/portal/help/VHDL/samples/samples.html
[18] http://www.eda.org/VIUF_proc/Spring95/BERGERON95A.PDF

Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

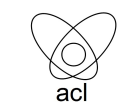




Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

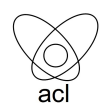


Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

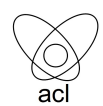




Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.
  - "Each process invocation is costly."

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

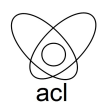


Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.
  - "Each process invocation is costly."
  - Use variables over signals, whenever possible

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt



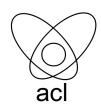


Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.
  - "Each process invocation is costly."
  - Use variables over signals, whenever possible
  - "Integer types perform better than logic types."

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

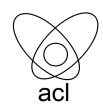




Evaluation

- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.
  - "Each process invocation is costly."
  - Use variables over signals, whenever possible
  - "Integer types perform better than logic types."
  - Case statements are more efficient than if statements

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

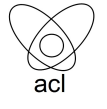


Evaluation

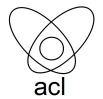
- Restricted aliasing: "VHDL Compiler does not support access (pointer) types because no equivalent hardware construct exists." [19]
- Efficiency[20,21]:
  - Employs static type checking.
  - "Each process invocation is costly."
  - Use variables over signals, whenever possible
  - "Integer types perform better than logic types."
  - Case statements are more efficient than if statements
  - (Bit) Arrays are more efficient than individual (bit) variables

[19] http://cseweb.ucsd.edu/~tweng/cse143/VHDLReference/04.pdf
[20] http://www.eda.org/VIUF_proc/Fall96/WICKS96A.PDF
[21] http://www.eda.org/rassp/vhdl/guidelines/vhdl-tips.txt

Dataflow Programming Languages

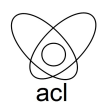


Background



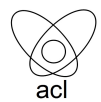
Background

- Pure Data is an open source visual programming language



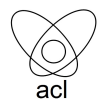
Background

- Pure Data is an open source visual programming language
- Derived from the Max programming languages and developed by Miller Puckette in the 1990's



Background

- Pure Data is an open source visual programming language
- Derived from the Max programming languages and developed by Miller Puckette in the 1990's
- "[Pure Data] enables musicians, visual artists, performers, researchers, and developers to create software graphically, without writing lines of code."



Evaluation

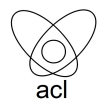
- Data types[22,23]

[22] http://www.pd-tutorial.com/english/ch02s02.html#id412354
[23] http://en.flossmanuals.net/pure-data/network-data/osc/

Evaluation

- Data types[22,23]
  - Integer, single and double precision floating point real numbers, string, logical, and signals as primitive data types

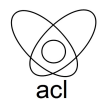
[22] http://www.pd-tutorial.com/english/ch02s02.html#id412354
[23] http://en.flossmanuals.net/pure-data/network-data/osc/



Evaluation

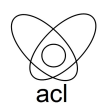
- Data types[22,23]
  - Integer, single and double precision floating point real numbers, string, logical, and signals as primitive data types
  - Lists as derived types

[22] http://www.pd-tutorial.com/english/ch02s02.html#id412354
[23] http://en.flossmanuals.net/pure-data/network-data/osc/



Evaluation

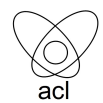
- Syntax design



http://www.pd-tutorial.com/english/ch02.html

Evaluation

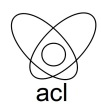
- Syntax design
  - No grammar specification found, (number of) usable objects and keywords will be used instead as indicator
  - Surface objects: 16, each with varying number of inlets and outlets
  - Keywords: 187



http://www.pd-tutorial.com/english/ch02.html

Evaluation

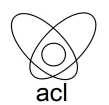
- Syntax design
  - No grammar specification found, (number of) usable objects and keywords will be used instead as indicator
  - Surface objects: 16, each with varying number of inlets and outlets
  - Keywords: 187
  - The use of the "Object" surface object spans several usages, from declaration of lists, printing out variables, creating expressions to invoking operations and functions



http://www.pd-tutorial.com/english/ch02.html

Evaluation

- Syntax design
  - No grammar specification found, (number of) usable objects and keywords will be used instead as indicator
  - Surface objects: 16, each with varying number of inlets and outlets
  - Keywords: 187
  - The use of the "Object" surface object spans several usages, from declaration of lists, printing out variables, creating expressions to invoking operations and functions
  - In addition, some of the capabilities of the "Object" surface object can be performed in the same manner as the "Message' surface object



http://www.pd-tutorial.com/english/ch02.html

Evaluation

- Abstraction: Can be achieved via subpatch(ing of operations) and streamlining subpatches[25]

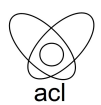
[25] http://www.pd-tutorial.com/english/ch05.html
[26] http://www.pd-tutorial.com/english/ch03.html
[27] http://www.pd-tutorial.com/english/ch04.html
[28] http://www.pd-tutorial.com/english/ch02s02.html
[29] http://www.pd-tutorial.com/english/ch04s04.html

Evaluation

- Abstraction: Can be achieved via subpatch(ing of operations) and streamlining subpatches[25]
- Expressivity: Has rich set of functionalities, but mostly for audio signal processing and music generation[25,26,27]

[25] http://www.pd-tutorial.com/english/ch05.html
[26] http://www.pd-tutorial.com/english/ch03.html
[27] http://www.pd-tutorial.com/english/ch04.html
[28] http://www.pd-tutorial.com/english/ch02s02.html
[29] http://www.pd-tutorial.com/english/ch04s04.html



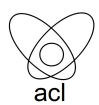


Evaluation

- Abstraction: Can be achieved via subpatch(ing of operations) and streamlining subpatches[25]
- Expressivity: Has rich set of functionalities, but mostly for audio signal processing and music generation[25,26,27]
- Exception handling: No special mechanism for exception handling, but can be done via proper rerouting of a particular condition to a corresponding operation [28,29]

[25] http://www.pd-tutorial.com/english/ch05.html
[26] http://www.pd-tutorial.com/english/ch03.html
[27] http://www.pd-tutorial.com/english/ch04.html
[28] http://www.pd-tutorial.com/english/ch02s02.html
[29] http://www.pd-tutorial.com/english/ch04s04.html





Evaluation

- Abstraction: Can be achieved via subpatch(ing of operations) and streamlining subpatches[25]
- Expressivity: Has rich set of functionalities, but mostly for audio signal processing and music generation[25,26,27]
- Exception handling: No special mechanism for exception handling, but can be done via proper rerouting of a particular condition to a corresponding operation [28,29]
- Restricted aliasing: No (explicit) aliasing begin done

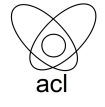
[25] http://www.pd-tutorial.com/english/ch05.html
[26] http://www.pd-tutorial.com/english/ch03.html
[27] http://www.pd-tutorial.com/english/ch04.html
[28] http://www.pd-tutorial.com/english/ch02s02.html
[29] http://www.pd-tutorial.com/english/ch04s04.html

Evaluation

- Efficiency

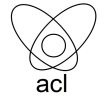
Evaluation

- Efficiency
  - Employs dynamic type-checking



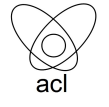
Evaluation

- Efficiency
  - Employs dynamic type-checking
  - Pure Data is an interpreted language



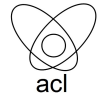
Evaluation

- Efficiency
  - Employs dynamic type-checking
  - Pure Data is an interpreted language
  - Significant delays in creation of large arrays and graph



Evaluation

- Efficiency
  - Employs dynamic type-checking
  - Pure Data is an interpreted language
  - Significant delays in creation of large arrays and graph
  - Prone to stack overflow errors due to combined memory requirement of interpreter itself and requirement of execution



END OF SESSION 9