



Operational semantics for the Gannet functional machine language

Wim Vanderbauwhede
Department of Computing Science
University of Glasgow, UK



Overview

- What is a Service-Based System-on-Chip?
- How does it work?
- Gannet programs
- Operational semantics
- Conclusion



What is Gannet?

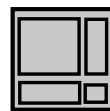
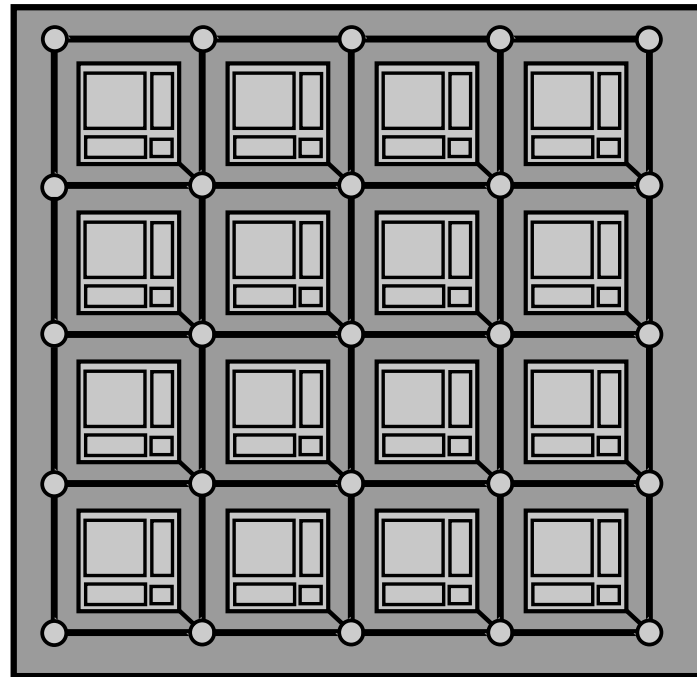
A distributed System-on-Chip (SoC) architecture

- a collection of processing cores (HW/SW)
- each core offers a a specific **service**
- all services are **fully connected** over an on-chip network (NoC)
- all information is transfered as **packets** over the NoC

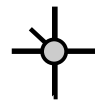


What is Gannet?

A distributed System-on-Chip (SoC) architecture



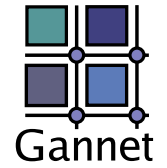
tile with
IP core



NoC switch



How does a Gannet SoC work?



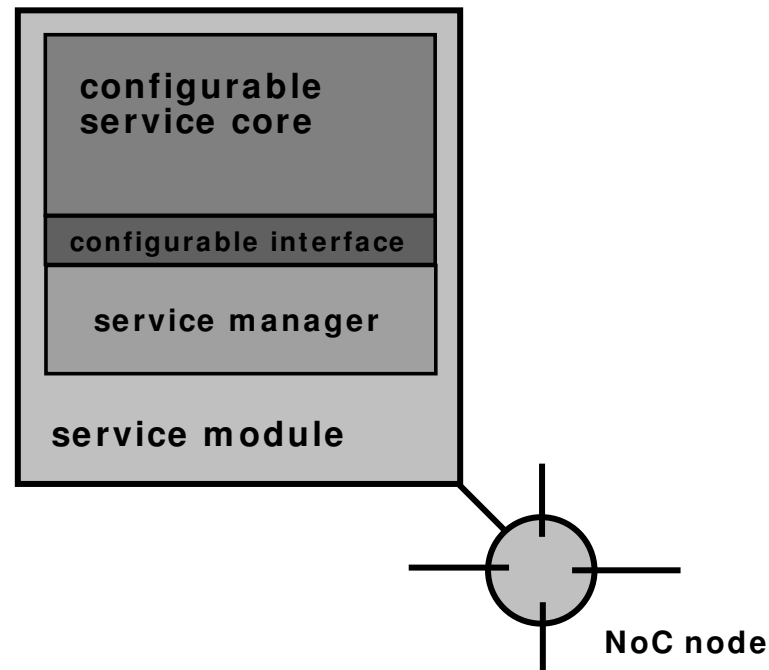
The services collaborate in a demand-driven dataflow fashion:

- **data** enter the system
- to be processed by **services**
- the **results** of which are, like the data, processed by services
- this process evolves according to a predefined but configurable **task**
- the **description** of such a task is a Gannet **program**



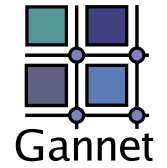
Managing the services

- to manage the flow of **data** and **task descriptions** between the **heterogenous service cores**, every core interfaces with the system through a **service manager**





Link with functional languages



- the Gannet architecture can be viewed as a distributed demand-driven dataflow machine
- by providing dataflow control services, the system can be made Turing-complete
- the machine language for this architecture is a pure functional language



Gannet program

Example task: factorial

```
(group
  (assign 'n 7)
  (assign 'fact (lambda 'n 'f_
    '(if (< n 2)
        1
        '(* n (apply f_ '(- n 1) 'f_ )))
  )))
(apply fact 'n 'fact))
```



Symbols

- Symbol has 5 fields: Kind, Task, Subtask, Name, Count
- In the current prototype, a symbol is 64 bits long

Example in symbols:

...

S 1 2 **assign** 4

Q 1 0 ' 2

V 1 1 fact 1

R 1 4 **lambda** 1

...



Core language constructs

- **if**: conditional evaluation
- **group/assign**: lexical scoping and variables
- **lambda/apply**: functions
- arithmetic and logic operations
- lists and list operations



Operational semantics

- For a Gannet language program in the context of the Gannet machine
- Context-sensitive reduction semantics (Felleisen)
- Evaluation of a service expression has two distinct, atomic stages:
 - **marshalling** stage (M): performed by service manager
 - **processing** stage (P): performed by service core



Symbols and Symbol lists

$Symbols = Service\text{-}symbols$

$\cup Data\text{-}symbols$

$\cup Number\text{-}symbols$

$\cup Quote\text{-}symbols$

$\cup Variable\text{-}symbols$

$\cup Argument\text{-}symbols$

$symbol\text{-}list ::= \langle (symbol \mid symbol\text{-}list)^+ \rangle$

$symbol\text{-}list \in Symbol\text{-}lists$



Expressions

Expressions \subset *Symbol-lists*

Expressions = *Evaluable-expressions*

\cup *Quoted-expressions*

Evaluable-expressions = *Service-expressions*

\cup *Number-symbols*

\cup *Variable-symbols*



Expressions

$service-expression ::= \langle service-symbol \ expression^+ \rangle$

$quoted-expression ::= \langle quote-symbol \ expression^+ \rangle$
 $\quad \quad \quad | \langle quote-symbol \ argument-symbol \rangle$

$gannet-program ::= service-expression$



Values

Values = Numbers

\cup *Data*

\cup *Expressions*

\cup *Value-lists*

value-list ::= *value* +

value-list \in *Value-lists*



Language constructs

$group\text{-}expr ::= \langle \mathbf{group} \ (quote\text{-}symbol? \ assign\text{-}expr) * \ expr+ \rangle$

$assign\text{-}expr ::= \langle \mathbf{assign} \ quote\text{-}symbol \ variable\text{-}symbol \ expr \rangle$

$quoted\text{-}arg\text{-}list ::= \langle quote\text{-}symbol \ argument\text{-}symbol \rangle *$

$lambda\text{-}expr ::= \langle \mathbf{lambda} \ quoted\text{-}arg\text{-}list \ quoted\text{-}expr \rangle$

$apply\text{-}expr ::= \langle \mathbf{apply} \ (lambda\text{-}expr \mid variable\text{-}symbol) \ expr+ \rangle$

$if\text{-}expr ::= \langle \mathbf{if} \ evaluable\text{-}expr \ expr \ expr \rangle$



expression : e

service – expression : se

quoted – expression : qe

quote – symbol : q

value : w

service – symbol : s

variable – symbol : v

argument – symbol : x

number – symbol : n

w^N : *number*. $\#t \equiv 1.0$, $\#f \equiv 0.0$.

w^D : *data*. “black box” data defined outside the system.

w^E : *expression*. includes single-symbol expressions.

w^L : *value-list*.



Operational semantics

- Evaluation context (Marshalling stage)

$$C_M ::= [] \mid \langle s \dots C_M \dots \rangle$$

- Store

- the Gannet machine does not have global, shared memory; every service has its own local memory, with read-only access for the other services
- the **store**() concept must be contextualised (subscript to indicate context of store)



Basic semantics

- Evaluate expression (delegate subtask)

$$C[\langle s \dots se_1 \dots \rangle] \rightarrow^M C[s \dots w_1 \dots]$$

- Lookup global variable (request data)

$$\begin{aligned} & (store_{data}(\dots (d_1 w_1^D) \dots) C[\langle s \dots d_1 \dots \rangle]) \\ \rightarrow^M & (store_{data}(\dots (d_1 w_1^D) \dots) C[s \dots w_1^D \dots]) \end{aligned}$$



Basic semantics

- Store Number value (reduce Number symbol to number)

$$C[\langle s \dots n_1 \dots \rangle] \rightarrow^M C[s \dots w_1^N \dots]$$

- Store Quoted expression (defer evaluation to Processing stage)

$$C[\langle s \dots qe_1 \dots \rangle] \rightarrow^M C[s \dots e_1 \dots]$$

- Processing stage (delta reduction)

$$C_P[(s \ w_1 \dots w_i \dots w_n)] \rightarrow^P \delta(s, w_1, \dots, w_i, \dots, w_n)$$



Conditional evaluation

- Eager (unquoted)

$$C[\langle \mathbf{if} \ e_c \ e_t \ e_f \rangle] \rightarrow^M$$

$$C[\langle \mathbf{if} \ w_c \ w_t \ w_f \rangle] \rightarrow^P \begin{cases} C[w_t] & (w_c \neq \#f) \\ C[w_f] & (w_c = \#f) \end{cases}$$

- Lazy (quoted)

$$C[\langle \mathbf{if} \ e_c \ qe_t \ qe_f \rangle] \rightarrow^M$$

$$C[\langle \mathbf{if} \ w_c \ e_t \ e_f \rangle] \rightarrow^P \begin{cases} C[w_t] & (w_c \neq \#f) \\ C[w_f] & (w_c = \#f) \end{cases}$$



- Variable assignment

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots) C_g[\langle \mathbf{assign} \mathit{qv} \mathit{e} \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots) C_g[\langle \mathbf{assign} \mathit{v} \mathit{w} \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots(\mathit{v} \mathit{w})\dots) C_g[\mathit{v}]) \end{aligned}$$

- Lookup lexically scoped variable

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots(\mathit{v}_1 \mathit{w})\dots) C_{vl}[\langle s \dots \mathit{v}_1 \dots \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots(\mathit{v}_1 \mathit{w})\dots) C_{vl}[s \dots \mathit{w} \dots]) \end{aligned}$$



Contexts for variables

$$C_g ::= \langle \mathbf{group} \dots [] \dots \rangle$$

$$C_{va} ::= \langle \mathbf{assign} \ q [] \ e \rangle$$

$$C_a ::= \langle \mathbf{assign} \ qv \ C_M \rangle$$

$$C_{vl} ::= C_g [\dots C_a [v_1] \dots C_{va} [v_1] \dots] \\ | C_g [\dots C_{va} [v_1] \dots C_M [v_1] \dots]$$



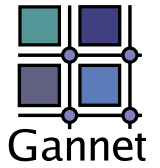
Scoping construct

- Concurrent (unquoted)

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{group} \dots \langle \mathbf{assign} \mathit{qv}_i \mathit{e}_i \rangle \dots \mathit{e} \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots (\mathit{v}_i \mathit{w}_i) \dots) C[\langle \mathbf{group} \dots \mathit{v}_i \dots \mathit{w} \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots) C[\mathit{w}]) \end{aligned}$$

- Sequential (quoted): $C_P ::= ([] \dots \mathit{e} \dots) \mid (\mathit{w} \dots [] \mathit{e} \dots)$

$$\begin{aligned} & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{group} \dots \mathit{q} \langle \mathbf{assign} \mathit{qv}_i \mathit{e}_i \rangle \dots \mathit{qe} \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\text{assign}}(\dots) C[\langle \mathbf{group} \dots \langle \mathbf{assign} \mathit{qv}_i \mathit{e}_i \rangle \dots \mathit{e} \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots (\mathit{v}_i \mathit{w}_i) \dots) C_P[\dots \mathit{v}_i \dots \mathit{w}]) \\ \rightarrow^P & (\mathbf{store}_{\text{assign}}(\dots) C_P[\mathit{w}]) \end{aligned}$$



- Function definition

$$\begin{aligned}
 & C[\langle \mathbf{lambda} \mathit{qx}_1 \dots \mathit{qx}_i \dots \mathit{qx}_n \mathit{qe}_a \rangle] \\
 & \rightarrow^M C[\langle \mathbf{lambda} \mathit{x}_1 \dots \mathit{x}_i \dots \mathit{x}_n \mathit{e}_a \rangle] \\
 & \rightarrow^P C[\langle \mathit{x}_1 \dots \mathit{x}_i \dots \mathit{x}_n \mathit{e}_a \rangle]
 \end{aligned}$$

- Function application (unquoted: value substitution)

$$\begin{aligned}
 & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle \mathbf{lambda} \mathit{qx}_1 \dots \mathit{qx}_n \mathit{qe}_a \rangle \mathit{e}_1 \dots \mathit{e}_n \rangle]) \\
 \rightarrow^M & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle \mathit{x}_1 \dots \mathit{x}_n \mathit{e}_a \rangle \mathit{w}_1 \dots \mathit{w}_n \rangle]) \\
 \rightarrow^P & (\mathbf{store}_{\mathbf{apply}}(\dots (\mathit{x}_1 \mathit{w}_1) \dots (\mathit{x}_i \mathit{w}_i) \dots (\mathit{x}_n \mathit{w}_n)) C[\mathit{e}_a[\mathit{x}_i / \mathit{w}_i]]) (\beta_v) \\
 \rightarrow^P & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\mathit{w}_a]) (\delta)
 \end{aligned}$$



Functions

- Function application (quoted: expression reference substitution)

$$\begin{aligned} & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle \mathbf{lambda} \mathit{qx}_1 \dots \mathit{qx}_n \mathit{qe}_a \rangle \mathit{qe}_1 \dots \mathit{e}_n \rangle]) \\ \rightarrow^M & (\mathbf{store}_{\mathbf{apply}}(\dots) C[\langle \mathbf{apply} \langle x_1 \dots x_n e_a \rangle e_1 \dots e_n \rangle]) \\ \rightarrow^P & (\mathbf{store}_{\mathbf{apply}}(\dots (x_1 e_1) \dots (x_i e_i) \dots (x_n e_n)) C[e_a[x_i / e_i]]) \quad (\beta_v) \\ \rightarrow^P & (\mathbf{store}_{\mathbf{apply}}(\dots) C[w_a]) \quad (\delta) \end{aligned}$$

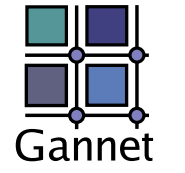


Future work

- Lists (call by reference)
- Semantics for control and state
- Semantics for scheduling and prioritization



UNIVERSITY
of
GLASGOW





Why Gannet?

- tomorrow's SoC's will be **very big** (10^{10} logic gates)
 - traditional bus-style interconnect causes a bottleneck:
 - Synchronisation over large distances is impossible
 - Fixed point-to-point result in huge wire overhead
 - **on-chip networks** provide a solution
 - globally asynchronous/locally synchronous
 - flexible connectivity
- design reuse is essential => IP ("Intellectual Property") cores
- IP cores are highly complex, self-contained units
- treating such blocks as **services** is a logical abstraction