UNIVERSITY
of
GLASGOW

Gannet

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

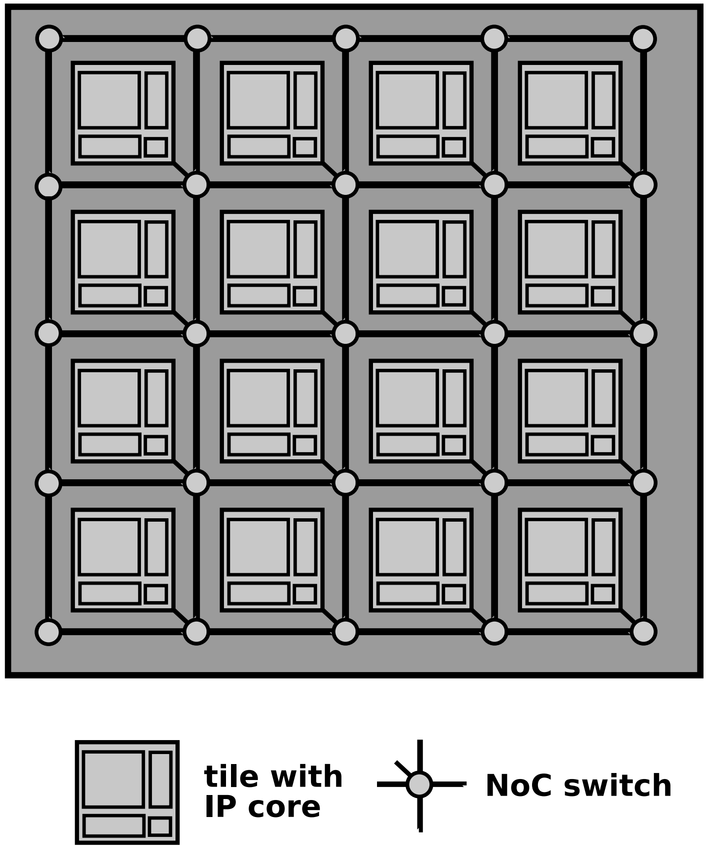
What is Gannet?

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



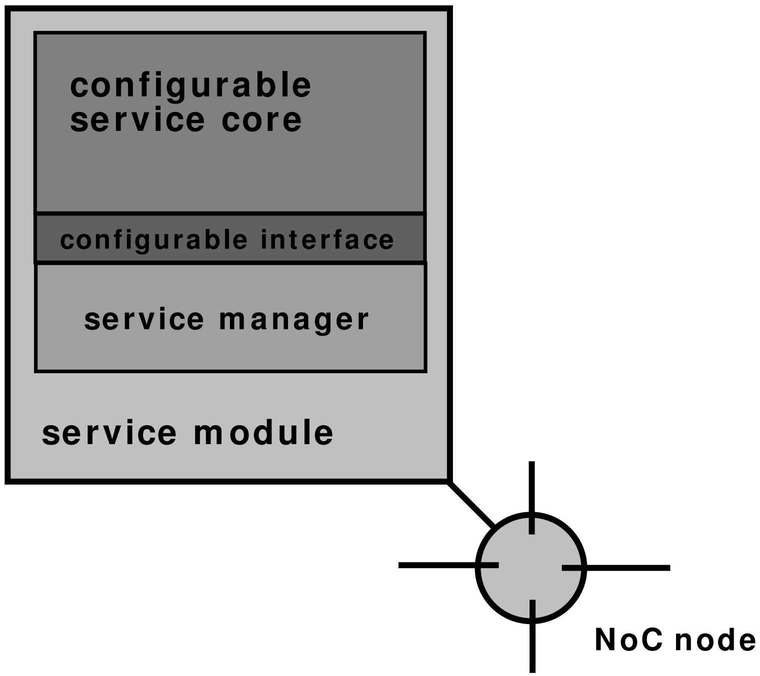
How does a Gannet SoC work?

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

UNIVERSITY of GLASGOW

Symbols and Symbol lists

Gannet

$$
\begin{aligned}
\mathit{Symbols} = {} & \mathit{Service\text{-}symbols} \\
& \cup \mathit{Data\text{-}symbols} \\
& \cup \mathit{Number\text{-}symbols} \\
& \cup \mathit{Quote\text{-}symbols} \\
& \cup \mathit{Variable\text{-}symbols} \\
& \cup \mathit{Argument\text{-}symbols}
\end{aligned}
$$

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

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

Expresssions

$$
\begin{aligned}
Expressions &\subset Symbol\text{-}lists \\
Expressions &= Evaluable\text{-}expressions \\
&\cup Quoted\text{-}expressions
\end{aligned}
$$

$$
\begin{aligned}
Evaluable\text{-}expressions &= Service\text{-}expressions \\
&\cup Number\text{-}symbols \\
&\cup Variable\text{-}symbols
\end{aligned}
$$

Expresssions

$$service\text{-}expression ::= \langle service\text{-}symbol \;\; expression+ \rangle$$

$$\begin{aligned} quoted\text{-}expression ::= & \langle quote\text{-}symbol \;\; expression+ \rangle \\ | \;\; & \langle quote\text{-}symbol \;\; argument\text{-}symbol \rangle \end{aligned}$$

$$gannet\text{-}program ::= service\text{-}expression$$

Values

$$
\begin{aligned}
\mathit{Values} = {} & \mathit{Numbers} \\
& \cup \mathit{Data} \\
& \cup \mathit{Expressions} \\
& \cup \mathit{Value\text{-}lists}
\end{aligned}
$$

$$
\mathit{value\text{-}list} ::= \mathit{value}+
$$

$$
\mathit{value\text{-}list} \in \mathit{Value\text{-}lists}
$$

Language constructs

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

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

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

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

$$apply\text{-}expr ::= \langle \textbf{apply}\ \ (lambda\text{-}expr\,|\,variable\text{-}symbol)\ \ expr+\rangle$$

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

UNIVERSITY of GLASGOW

Gannet

Shorthand

$expression : e$

$service - expression : se$

$quoted - expression : qe$

$quote - symbol : q$

$value : w$

$service - symbol : s$

$variable - symbol : v$

$argument - symbol : x$

$number - symbol : n$

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

$w^D : data.$ "black box" data defined outside the system.

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

$w^L : value\text{-}list.$

Operational semantics

- Evaluation context (Marshalling stage)

$$C_M ::= [\,] \mid \langle s \ldots C_M \ldots \rangle$$

- 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 $\mathbf{store}()$ concept must be contextualised (subscript to indicate context of store)

Basic semantics

- Evaluate expression (delegate subtask)

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

- Lookup global variable (request data)

$$(store_{data}(\ldots(d_1\, w_1^D)\ldots)\ C[\langle s \ \ldots d_1 \ldots\rangle])$$
$$\rightarrow^M \ (store_{data}(\ldots(d_1\, w_1^D)\ldots)\ C[s \ldots w_1^D \ldots])$$

Basic semantics

- Store Number value (reduce Number symbol to number)

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

- Store Quoted expression (defer evaluation to Processing stage)

$$C[\langle s\ \ldots q e_1 \ldots \rangle] \rightarrow^M C[s \ldots e_1 \ldots]$$

- Processing stage (delta reduction)

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

Conditional evaluation

- Eager (unquoted)

$$C[\langle \textbf{if } e_C \ e_t \ \ e_f \rangle] \rightarrow^M$$

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

- Lazy (quoted)

$$C[\langle \textbf{if } e_C \ qe_t \ \ qe_f \rangle] \rightarrow^M$$

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

Variables and lexical scoping

- Variable assignment

$$(\mathbf{store}_{\mathbf{assign}}(\ldots)\ C_g[\langle \mathbf{assign}\ qv\ e\rangle])$$
$$\rightarrow^M (\mathbf{store}_{\mathbf{assign}}(\ldots)\ C_g[\langle \mathbf{assign}\ v\ w\rangle])$$
$$\rightarrow^P \quad (\mathbf{store}_{\mathbf{assign}}(\ldots(v\ w)\ldots)\ C_g[v])$$

- Lookup lexically scoped variable

$$(\mathbf{store}_{\mathbf{assign}}(\ldots(v_1\, w)\ldots)\ C_{vl}[\langle s\ldots v_1\ldots\rangle])$$
$$\rightarrow^M \ (\mathbf{store}_{\mathbf{assign}}(\ldots(v_1\, w)\ldots)\ C_{vl}[s\ldots w\ldots])$$

UNIVERSITY of GLASGOW

Gannet

Contexts for variables

$$C_g ::= \langle \textbf{group} \ \ldots[\,]\ldots \rangle$$

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

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

$$\begin{aligned} C_{vl} ::= & \ C_g[\ldots C_a[v_1]\ldots C_{va}[v_1]\ldots] \\ | & \ C_g[\ldots C_{va}[v_1]\ldots C_M[v_1]\ldots] \end{aligned}$$

Scoping construct

- Concurrent (unquoted)

$$(\mathbf{store}_{\mathbf{assign}}(\ldots)\; C[\langle \mathbf{group}\ \ldots \langle \mathbf{assign}\ q v_i\ e_i\rangle \ldots\ e\rangle])$$

$$\rightarrow^{M}\ (\mathbf{store}_{\mathbf{assign}}(\ldots(v_i\ w_i)\ldots)\; C[\langle \mathbf{group}\ \ldots v_i \ldots\ w\rangle])$$

$$\rightarrow^{P}\ (\mathbf{store}_{\mathbf{assign}}(\ldots)\; C[w])$$

- Sequential (quoted): $C_P ::= ([\,]\ldots e\ldots) \mid (w\ldots[\,]\ e\ldots)$

$$(\mathbf{store}_{\mathbf{assign}}(\ldots)\; C[\langle \mathbf{group}\ \ldots q\langle \mathbf{assign}\ q v_i\ e_i\rangle \ldots\ qe\rangle])$$

$$\rightarrow^{M}\ (\mathbf{store}_{\mathbf{assign}}(\ldots)\; C[\langle \mathbf{group}\ \ldots \langle \mathbf{assign}\ q v_i\ e_i\rangle \ldots\ e\rangle])$$

$$\rightarrow^{P}\ (\mathbf{store}_{\mathbf{assign}}(\ldots(v_i\ w_i)\ldots)\; C_P[\ \ldots v_i \ldots\ w])$$

$$\rightarrow^{P}\ (\mathbf{store}_{\mathbf{assign}}(\ldots)\; C_P[w])$$

UNIVERSITY of GLASGOW

Gannet

Functions

- Function definition

$$C[\langle \textbf{lambda}\ qx_1 \ldots qx_i \ldots qx_n\ qe_a \rangle]$$
$$\to^M C[\langle \textbf{lambda}\ x_1 \ldots x_i \ldots x_n\ e_a \rangle]$$
$$\to^P C[\langle\ x_1 \ldots x_i \ldots x_n\ e_a \rangle]$$

- Function application (unquoted: value substitution)

$$(\textbf{store}_{\textbf{apply}}(\ldots)\ C[\langle \textbf{apply}\ \langle \textbf{lambda}\ qx_1 \ldots qx_n\ qe_a \rangle\ e_1 \ldots e_n \rangle])$$
$$\to^M (\textbf{store}_{\textbf{apply}}(\ldots)\ C[\langle \textbf{apply}\ \langle\ x_1 \ldots x_n\ e_a \rangle\ w_1 \ldots w_n \rangle])$$
$$\to^P (\textbf{store}_{\textbf{apply}}(\ldots(x_1\ w_1)\ldots(x_i\ w_i)\ldots(x_n\ w_n))\ C[e_a[\,x_i/\,w_i]])\quad (\beta_v)$$
$$\to^P (\textbf{store}_{\textbf{apply}}(\ldots)\ C[w_a])\quad (\delta)$$

Functions

- Function application (quoted: expression reference substitution)

$$(\mathbf{store}_{\mathbf{apply}}(\ldots)\ \ C[\langle\mathbf{apply}\ \langle\mathbf{lambda}\ qx_1 \ldots qx_n\ \ qe_a\rangle\ \ qe_1 \ldots e_n\rangle])$$

$$\longrightarrow^M \quad (\mathbf{store}_{\mathbf{apply}}(\ldots)\ \ C[\langle\mathbf{apply}\ \langle\ x_1 \ldots x_n\ \ e_a\rangle\ \ e_1\ \ldots e_n\rangle])$$

$$\longrightarrow^P \quad (\mathbf{store}_{\mathbf{apply}}(\ldots(x_1\ e_1)\ldots(x_i\ e_i)\ldots(x_n\ \ e_n))\ \ C[e_a[\ x_i/\ e_i]]) \qquad (\beta_v)$$

$$\longrightarrow^P \quad (\mathbf{store}_{\mathbf{apply}}(\ldots)\ \ C[w_a]) \qquad (\delta)$$

Future work

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

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