Separation of Data flow and Control flow in Reconfigurable Multi-core SoCs using the Gannet Service-based Architecture

Wim Vanderbauwhede
Department of Computing Science
University of Glasgow
Overview

- Task control in NoC-based SoCs
- Gannet system
- Gannet language
- Separation of data flow and control flow
Overview

- We propose a novel approach to separating control flow from data flow in NoC-based SoCs consisting of multiple heterogeneous reconfigurable IP cores.

- This mechanism enables full data path control by an embedded microcontroller whilst avoiding the potential memory bandwidth bottleneck and without requiring centralised control over the NoC.
Overview

- We assume a generic SoC where
  - data is processed by IP cores interacting through a NoC
  - control structures are implemented on a microcontroller.

- The proposed mechanism employs a service-based SoC architecture (the Gannet architecture) where
  - the control services are implemented using a Virtual Machine
  - IP cores acquire service behaviour through the use of a generic data marshalling and interfacing circuit.
Overview

- Task control in NoC-based SoCs
- Gannet system
- Gannet language
- Separation of data flow and control flow
- NoC-based SoC with embedded microprocessor:
Task control in NoC-based SoCs

- SoCs in general use an embedded microprocessor for control.
- Conventional way of controlling hardware blocks using an embedded microprocessor: memory-mapped IO+ interrupts.
- In a NoC-based SoC, the microprocessor interacts with a NoC transceiver and transfers data as NoC packets ⇒
  - efficient data transmission;
  - considerably reduction of required number of interrupts;
  - no significant operational difference with bus-based mechanism.
Non-task-level reconfigurable system:

- microcontroller only sends control or configuration information to each core;
- all data can flow between the cores.
Task-level reconfigurable system:

- data paths are determined at run time by a program running on microcontroller;
- all data pass via the microcontroller.
Data* NoC_TRX(CoreAddress&, ...); // variadic function prototype
/* variable declarations omitted */
data1=NoC_TRX(C1, IN1);
if (condition1) {
    Data* data2=NoC_TRX(C2, data1);
data5=NoC_TRX(C5, data2);
} else {
    Data* data2=NoC_TRX(C5, data1);
data5=NoC_TRX(C2, data2);
}
result=NoC_TRX(C3, data5);
Example

![Diagram showing a microcontroller with memory and connections to various data points such as data1, data2, data5, result, and condition1 (true).]
■ Overview

■ Task control in NoC-based SoCs

■ Gannet system

■ Gannet language

■ Separation of data flow and control flow
Gannet architecture

- A service-based architecture for very large SoCs:
  - a collection of processing cores (HW/SW).
  - each core offers a specific service.
  - tasks are defined by the interaction pattern of the services.

- Task-level reconfigurability
  - task description programs, configurable at run time

- High abstraction-level design
  - single program governs behaviour of complete system
The Gannet machine is a distributed computing system where every computational node consumes packets and produces packets and can store state information between transactions. We denote a Gannet packet as $p(\text{Type}, \text{To}, \text{Ret}, \text{Id}; \text{Payload})$.

The semantics of a Gannet service (computational node) can be described in terms of:

- the task code
- the internal state
- the result packet(s) produced by the task
Gannet system operation

1. Service $S_i$ receives a code packet $p(Code, S_i, S_j, R_{task}; task)$ where $task = (S_i a_1...a_n)$. The task is stored with reference $R_{task}$. Service $S_i$ is in $state_i$.

2. Service $S_i$ receives a task reference packet $p(Ref, S_i, S_j, R_{id}; R_{task})$

3. The service activates the task referenced by $R_{task}$: $(S_i a_1...a_n)$. This results in evaluation of the arguments $a_1..a_n$:

4. The service produces a result packet $p(Type_i, S_j, S_i, R_{id}; Result_i)$ and the state changes to $state_i'$.

5. This packet is sent to $S_j$ where $Result_i$ is stored in a location referenced by $R_{id}$.
Gannet system operation

- Calling Service
  - Reference packet
  - Result packet
- Called Service
  - Code packet
  - Reference packet
  - Result packet
- Gateway
- System = all services
Control services in Gannet

- Any run-time reconfigurable system requires **control constructs** to be effective.

- In Gannet, these constructs (if/then, functions, blocks, variables, ...) are provided by **services**.

- Such **control services** can be efficiently implemented on an embedded microcontroller.

- Interleaving the services provided by the HW cores with control services can cause bottleneck due to memory bandwidth.
Control services in Gannet

- Ideally, the microcontroller would only exchange control information with the cores.
  - technically not impossible to realise using compiled code but would require
    - a language with functional characteristics (no side-effects, undetermined execution order, laziness, concurrency)
    - access to absolute memory addresses of the data structures
  - program would need to contain a JIT compiler to create bytecode for the service managers at run time.
The Gannet Virtual Machine

- A Virtual Machine (VM) which interacts with the hardware service managers:
  - software implementation of the service managers, control service cores and a ’virtual NoC’
  - small, portable C++ application
  - runs byte-compiled programs in the Gannet language
  - same bytecode used by VM and HW
The Gannet system
Overview

- Task control in NoC-based SoCs
- Gannet system
- **Gannet language**
- Separation of data flow and control flow
Gannet language

■ The “assembly” language to program the Gannet system
■ Intended as compilation target, not HLL
■ A functional language, every service is mapped to an opaque function
■ Gannet is a distributed machine for running this language
■ Service = service manager + service core
■ Service cores can be implemented in HW or SW
Previous example in S-expressions syntax:

\[
(S8 \ (S4
\ (S3
\ (S2 \ (S1 \ IN1) )
\ (S6 \ (S5 \ IN2))
)
\ (S7
\ (S6 \ (S5 \ IN2))
\ (S2 \ (S1 \ IN1))
)))
\]
Example with control services (factorial):

(let
    (assign 'fact
        (lambda 'n 'a 'f
          '(if (< n '2)
             'n
             '(apply f (- n '1) (* a n) 'f)
           )))
    (apply fact '4 '1 'fact)
)
Gannet language properties

- Some key properties of the Gannet language:
  - the evaluation order is unspecified
  - eager by default but lazy evaluation is possible
  - no side effects across services
  - updates of variables are atomic (no race conditions)

- These properties
  - make the language fully concurrent (maximise parallelism)
  - and enable separation of control flow from data flow
Gannet language properties

- **Unspecified execution order:**
  - In a given function call it is not possible to predict the evaluation order of the arguments.
  - In practice, all arguments are evaluated in parallel; call blocks until all arguments are ready.

```lisp
(let
  (assign 'a (S1 ...))
  (assign 'b (S2 ...))
  (S3 ... b ...)
  (S4 ... a ...)...
)
```
Lazy evaluation:

- By default, Gannet is **eager**, i.e. it always evaluates all arguments before passing them on to the service core.
- It should be possible to evaluate arguments at need ("lazy").
- Laziness is expressed by prefixing an expression or symbol with a single quote:

  \[
  \text{(assign 'a (S1 ...) )}
  \]

- Quoting causes the evaluation of the symbol \(a\) to **deferred** to the service core.
Language properties

- **No side effects across services:**
  - A call to a given service should not result in a modification of the state of the rest of the system.

- **Updates of variables are atomic:**
  - No race conditions if several services simultaneously try to modify shared data.
  - The service manager processes all task requests in FIFO order.
  - Not possible to update an unassigned variable.
■ Overview

■ Task control in NoC-based SoCs

■ Gannet system

■ Gannet language

■ Separation of data flow and control flow
The mechanism for separation of control and data flows: **deferred evaluation and redirection:**

$S_1$ sends $p(R_{ref}, S_{ctl}, S_1, R'_1; R_{ctl})$; $S_{ctl}$ sends $p(R_{ref}, S_3, S_1, R'_1; R_3)$
Conditional branching

(S1
  (if cond
    '(Strue ...)
    '(Sfalse ...)
  )
  ...
)

$S_{if}$ sends $p(Ref, S_{false}, S_1, R'_1; R_{false})$
Revisiting the earlier example:

\[(S3
   (if condition1
      '(S5 (S2 (S1 IN1)))
      '(S2 (S5 (s1 IN1)))
   )
)
\]

\[S_{if} \text{ sends } p(Ref, S_5, S_3, R'_3; R_5)\]
Function definition and application

\[(S1\ 
  (apply\ 
    (lambda 'x\ 
      '(S2 (S3 ... x ...) ... x ...)\ 
    )\ 
    '(S4 ...)\ 
  )\ 
  ...
)
\]

\[S_1((\lambda x \rightarrow S_2(S_3(...,x,...),...,x,...))S_4(...),...)]
- **list**: list constructor
- **head**: first element of the list
- **return**: unquotes its argument

```lisp
(let
  (assign 'l (list '(S2 ...) '(S3 ...)))
  (S4 (return (head l)) (S1 ...))
)
```
Conclusion

- Gannet: a service-based SoC architecture for high-level design of reconfigurable heterogeneous multi-core SoCs.
- Alleviate bottleneck resulting from memory bandwidth limitation:
  mechanism for the separation of control flow and data flow based on deferred evaluation and packet redirection.
- Gannet system
  - provides full control over data paths in multi-core SoC;
  - provides full concurrency;
  - ensures that data can flow directly between the cores.
Virtex II-Pro

PowerPC 405

Gannet Virtual Machine (C++)

OPB-NoC bridge

NoC packet switch (Xilinx ref design)

Service Manager
RAM
IP core

Service Manager
RAM
IP core

Service Manager
RAM
IP core

Service Manager
RAM
IP core

Service Manager
RAM
IP core