Programming the Interface between Computation and Communication

Wim Vanderbauwhede
Department of Computing Science
University of Glasgow

Heterogeneous Systems

	Homogeneous	**Heterogeneous**
Multicore Processor	n-Core Intel	Cell
Multiprocessor System	SGI Altix (NUMA)	(multicore) processor + GPGPU + FPGA
System-on-Chip	homogeneous arrays (Ambric, MORA, AsAP)	heterogeneous multicore system on a single chip

Heterogeneous Multicore SoC

- Advances in integrated circuit technology and customer demands lead to increasing integration: entire systems on a single chip (**SoC**)
- Traditional system architecture (CPU, memory, peripherals connected over shared bus) can't scale
  - Synchronisation over large distances is impossible
  - Shared resource is performance bottleneck
- **On-chip networks** provide a solution
  - globally asynchronous/locally synchronous
  - flexible connectivity
  - parallel processing

Heterogeneous Multicore SoC

- **Heterogeneous** $\Longrightarrow$ “core” means any computational core: IP core, microprocessor, DSP, GPGPU, FPGA fabric
- No reason to treat von Neumann-style architecture different $\Longrightarrow$ all cores are “first-class” nodes on the network

Problem Definition

- Programming Systems where computation requires communication between cores – if all computations are independent, there is no multicore programming problem.
- Very large numbers of cores $\Longrightarrow$ communication issues
- Heterogeneous cores$\Longrightarrow$ integration issues
- How to govern the data flows in a heterogeneous multicore system?

OS support for Parallel Programming

- Threads, processes (OpenMP, MPI)
  - current OS'es are centralised $\Longrightarrow$ bottleneck
  - slow, large overhead (a program should not require the assistance of an OS to run)
  - assumes cores are von-Neumann processor, not suitable for heterogeneous systems
- OpenCL
  - abstracts the specifics of underlying hardware
  - many good ideas
  - but deals with the HW architecture as a given
  - and relies heavily on the OS

Challenges for Multicore SoC programming

- Language and compiler developers have for years focussed on von Neumann machines (sequential memory-based processor) or on low-level (RTL) hardware description (HDLs)
- We need languages and compilers for parallel hardware
  - Support for parallelism
  - Separation of data flow from control flow
- The hardware should actively support the programming model

Challenges for Multicore SoC programming

- HW manufacturers don’t design multicore systems with programming in mind (divide between HW and SW developers)
- How to design a heterogeneous multicore SoC infrastructure that will support high-level programming?

Challenges for Multicore SoC programming

- We propose an interface layer (HW) between **arbitrary** computational cores and **arbitrary** communication infrastructures

High-Level SoC View

- Cores $\Longrightarrow$ computation, data capture
- Network $\Longrightarrow$ communication
- No reasons for system to be globally synchronous
- In fact, lots of reasons not to: GALS paradigm

Terminology

Terminology (to avoid confusion with OS etc)

- A task is a distributed computation executed by a set of communicating cores
- a subtask is any part of the task executed on a particular core
- a core provides one or more services to the system

SoC Programming Model

- At low level (conceptually similar to OpenCL)
  - program the computations to be done by the cores (fixed cores have a fixed “program”).
  - program the communication between the cores
- At high level (the ideal)
  - use a common language for computation and communication
  - let the compiler work out the subtasks for every core and hence the communication

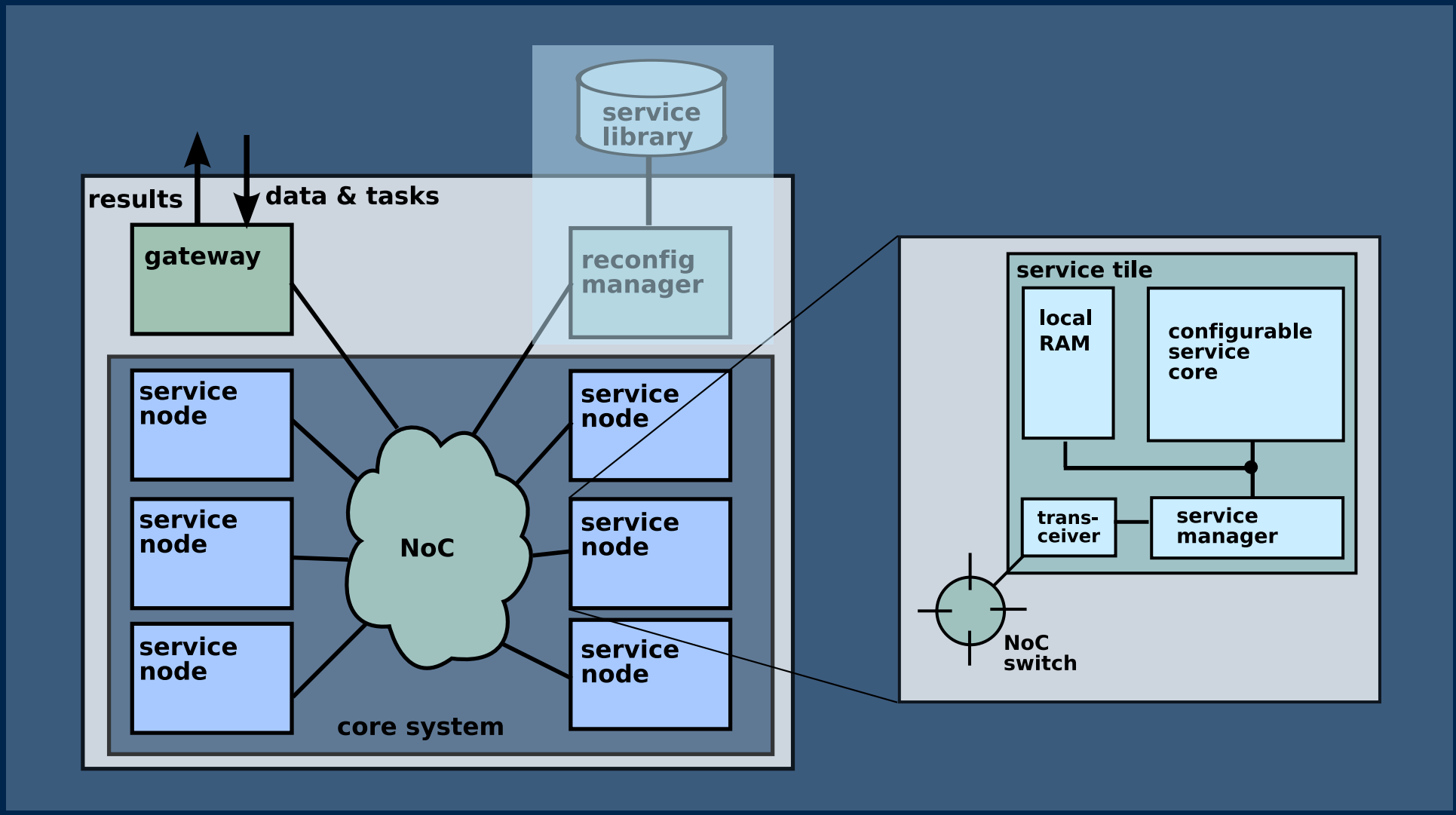
Gannet Platform

- interface layer between NoC and cores
- functional interface with stream support
- HW implementation but also VM
- capable of dynamic reconfiguration

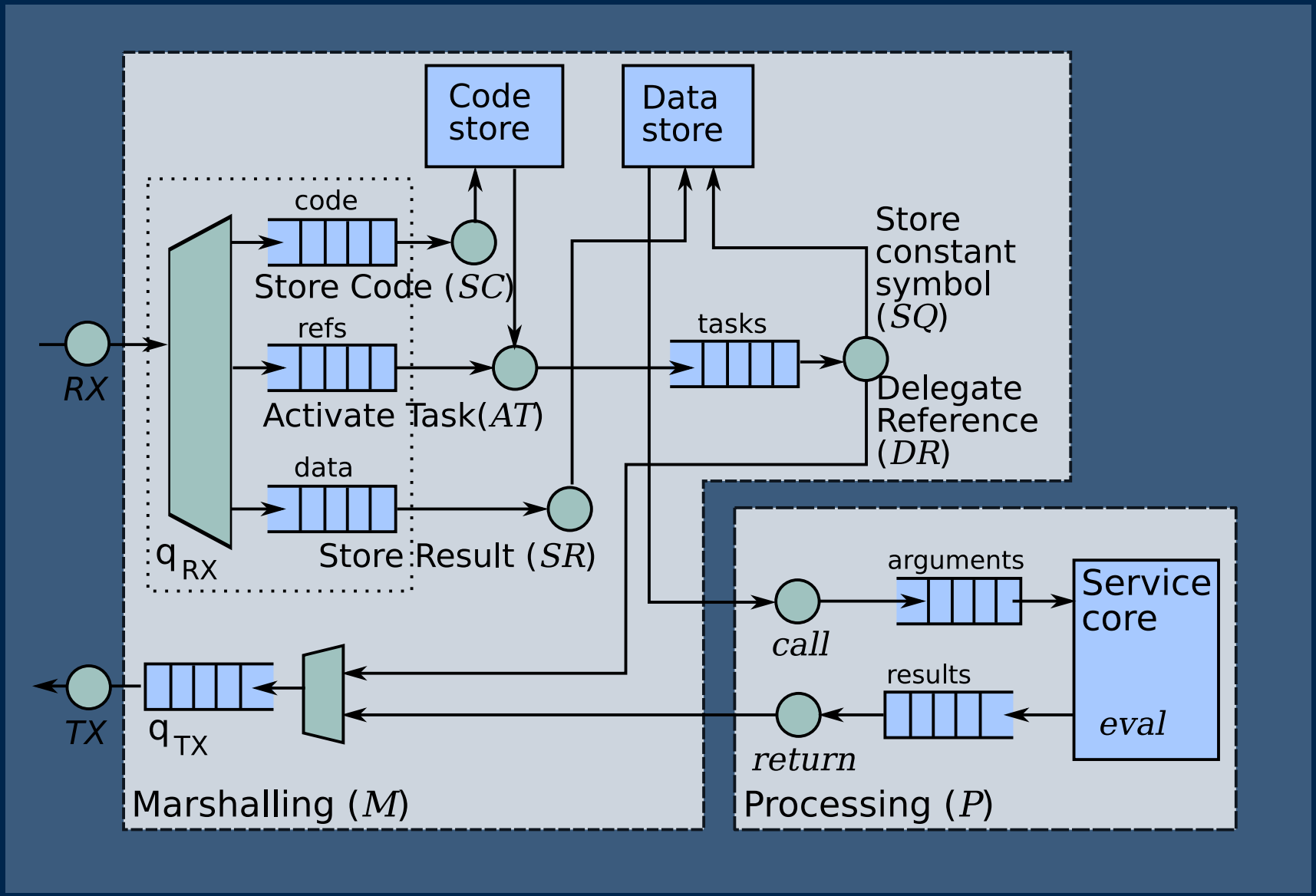
Gannet System Architecture

- A **service-based** architecture for **heterogeneous Multicore SoCs:**
  - a collection of IP cores (HW/SW).
  - each IP core offers a a specific **service.**
  - IP cores acquire service behaviour through a generic data marshalling interface, the **Service Manager**
  - services interact through a Network-on-Chip (NoC)
- High abstraction-level design: high-level program governs behaviour of complete system

Gannet System Architecture



Gannet Service Architecture

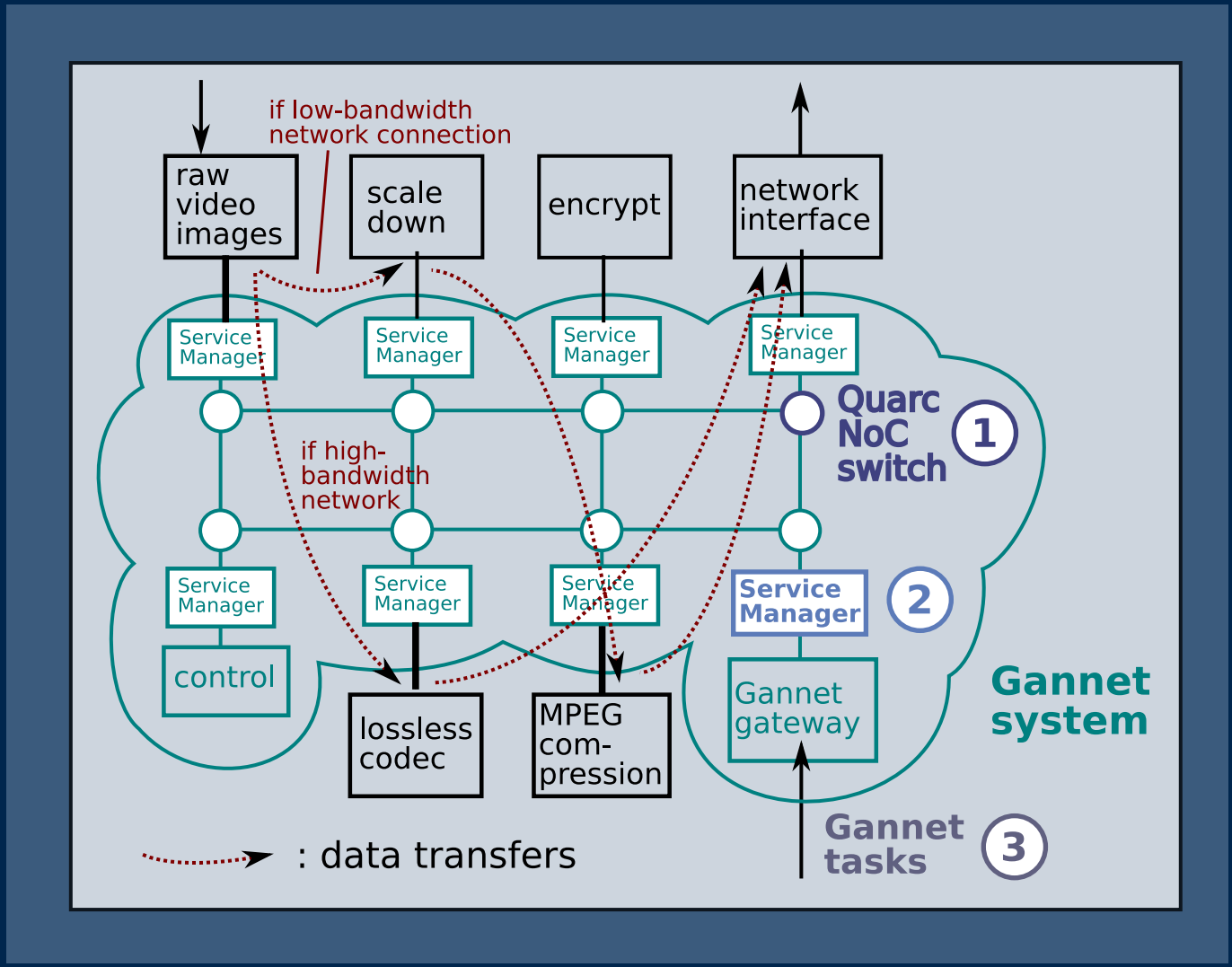


Gannet Service Abstraction

- Service = service manager + core (+ local memory + TRX)
  - Service core => function body, result computation
  - Service manager => function call, argument evaluation
- Gannet Services
  - computational (pure functions)
  - flow control (if, lambda,...)

Example

Simple video capure system



Gannet Language

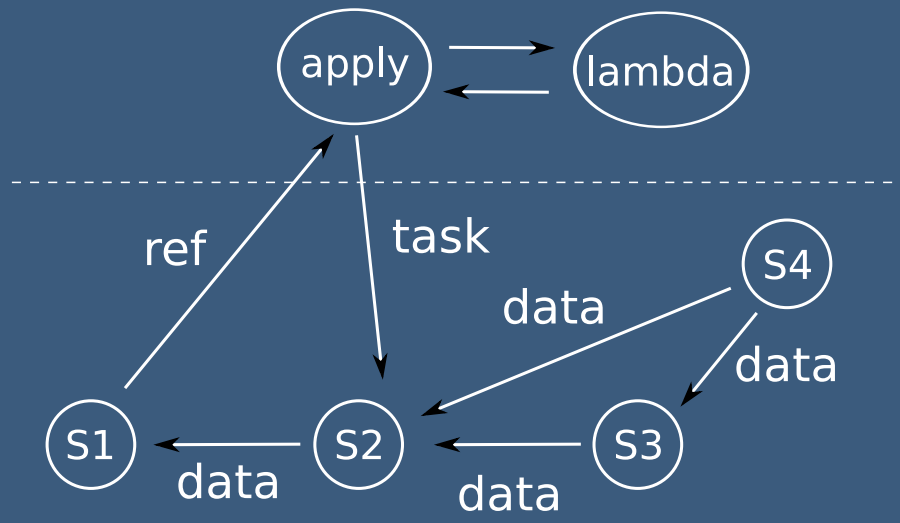
- The “assembly” (or IR) language to program the Gannet system
- Intended as compilation target, not HLL
- A functional language, every service is mapped to an opaque function

Gannet Language

- Some key properties of the Gannet language:
  - the evaluation order is unspecified
  - eager by default but deferring evaluation is possible
  - no side effects across services
- These properties
  - make the language fully concurrent (maximise parallelism)
  - and enable separation of control flow from data flow
  - facilitate support for stream processing

Example: Function Application

```
(S1
   (apply
      (lambda 'x
         '(S2 (S3 ... x ...) ... x ...)
      )
      '(S4 ...)
   )
   ...
)
```

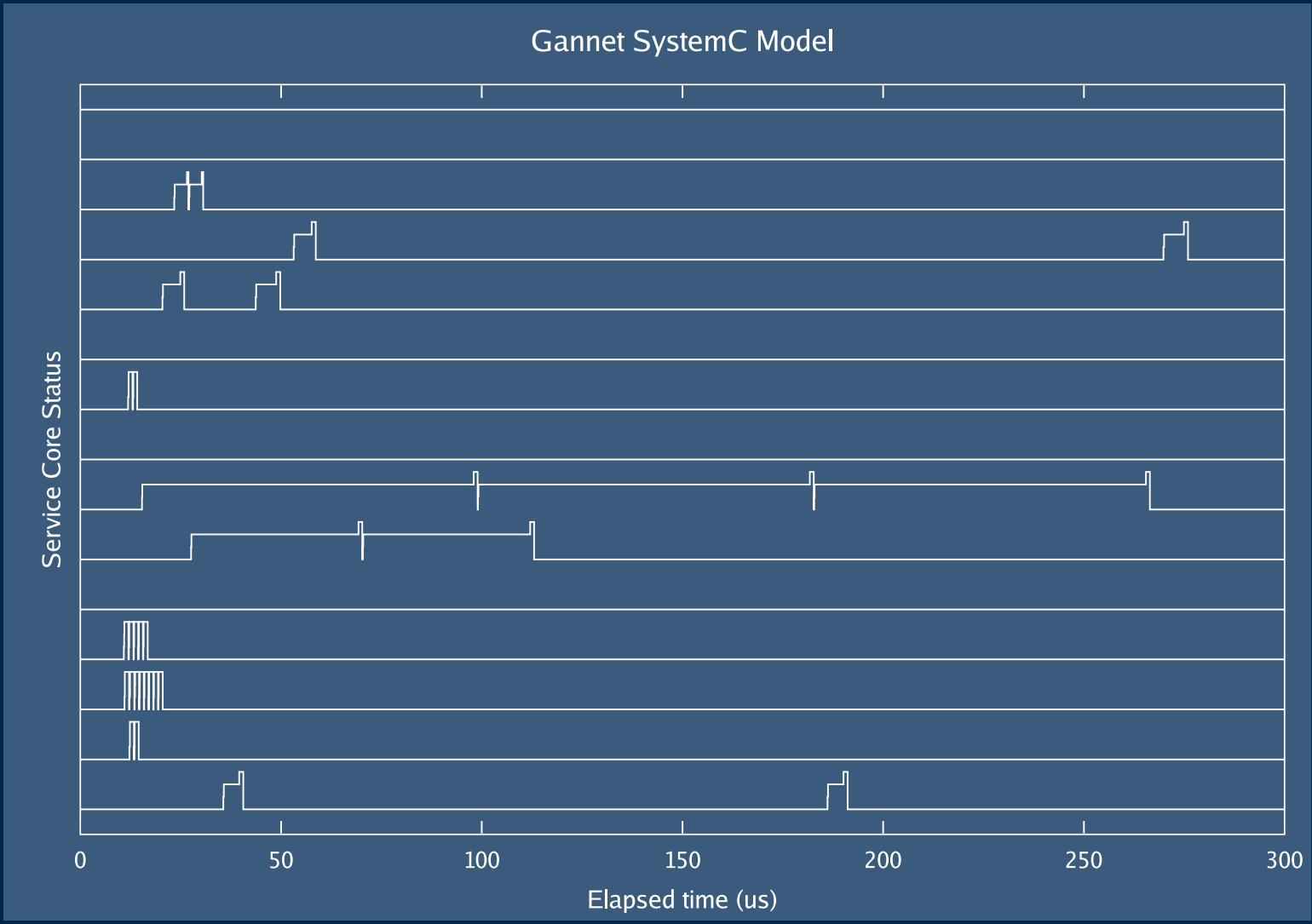


$$(S_1\,(\lambda x \rightarrow (S_2\,(S_3 \ldots x \ldots)\ldots x \ldots))\,(S_4 \ldots))\ldots)$$

Example Matrix Operations

```
(madd
  (cross (scale '0.5 (inv
    (if (< (det (a)) '0)
      '(mmult (a) (c))
      '(mmult (a) (d)) )))
    (tran (a)))
  (cross (scale '0.5 (inv
    (if (< (det (b)) '0)
      '(mmult (b) (d))
      '(mmult (b) (c)) )))
    (tran (b))) )
```

Example Matrix Operations



Hardware Implementation

- Cycle-approximate System-C model
- FPGA (Xilinx Virtex-II Pro) prototypes of
  - service manager
  - NoC switch and TRX (Quarc)
- Clock speed and slice count comparable with Xilinx Microblaze processor

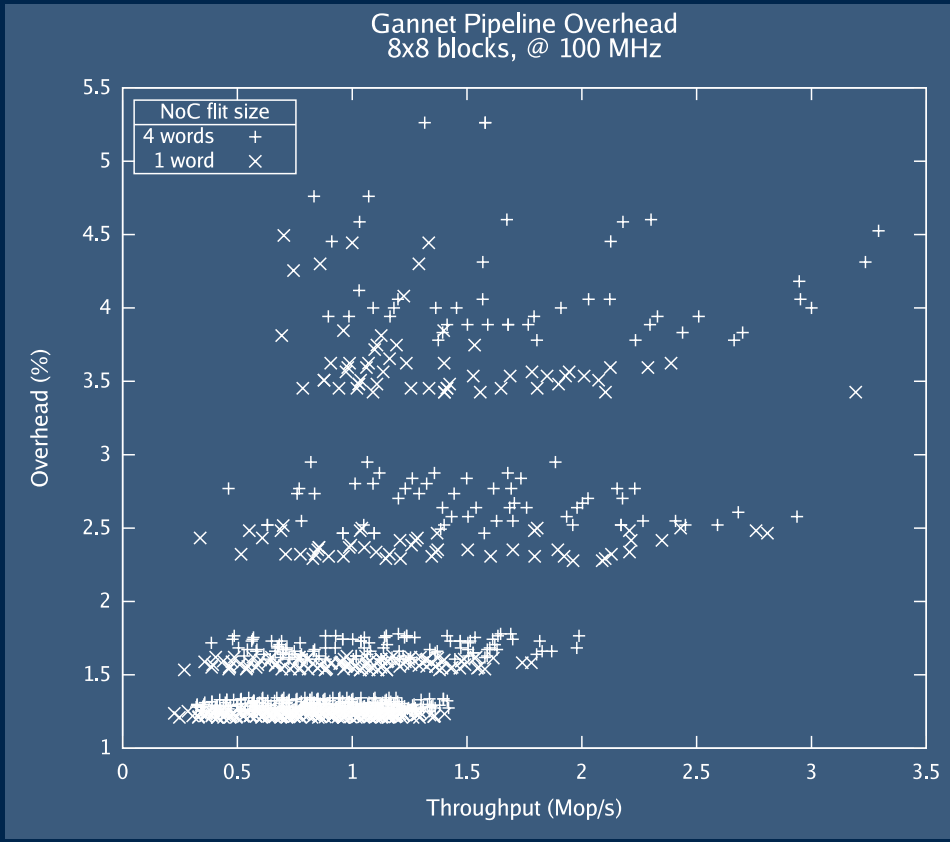
Software Implementation

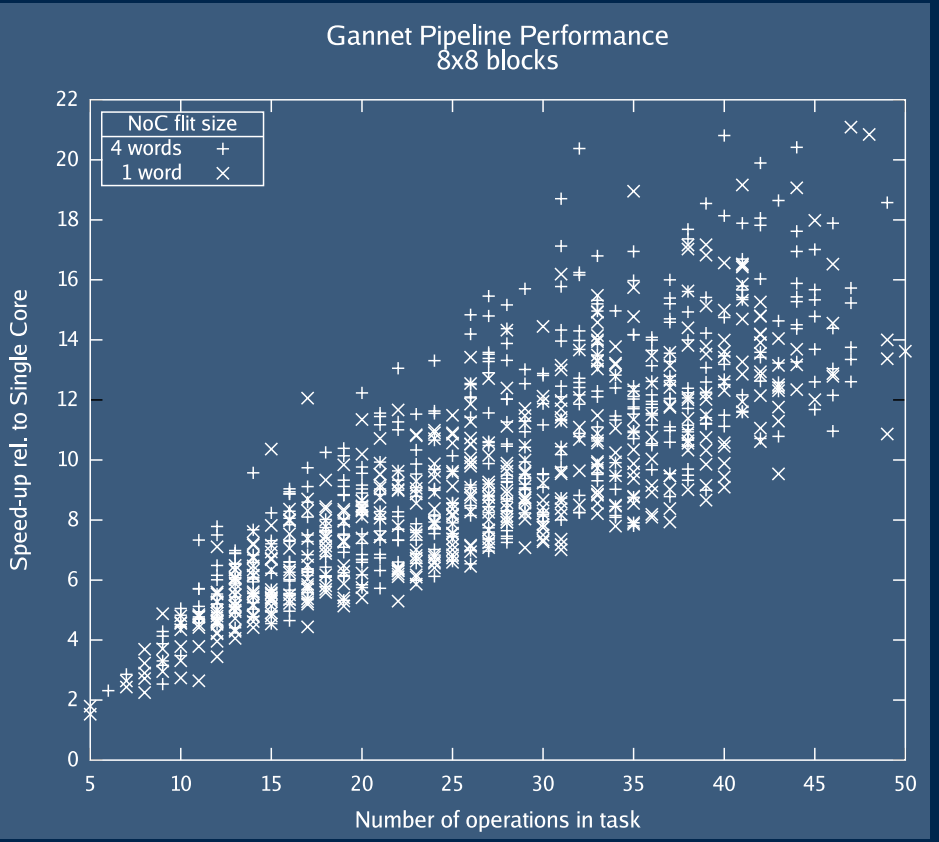
- Gannet Virtual Machine, a stand-alone VM for embedded processors
- Runs same Gannet bytecode as hardware service managers
- Running VM on e.g. Xilinx Microblaze processor is 2-3 orders of magnitude slower than HW
- But very flexibe, easy HW/SW codesign

Gannet Performance

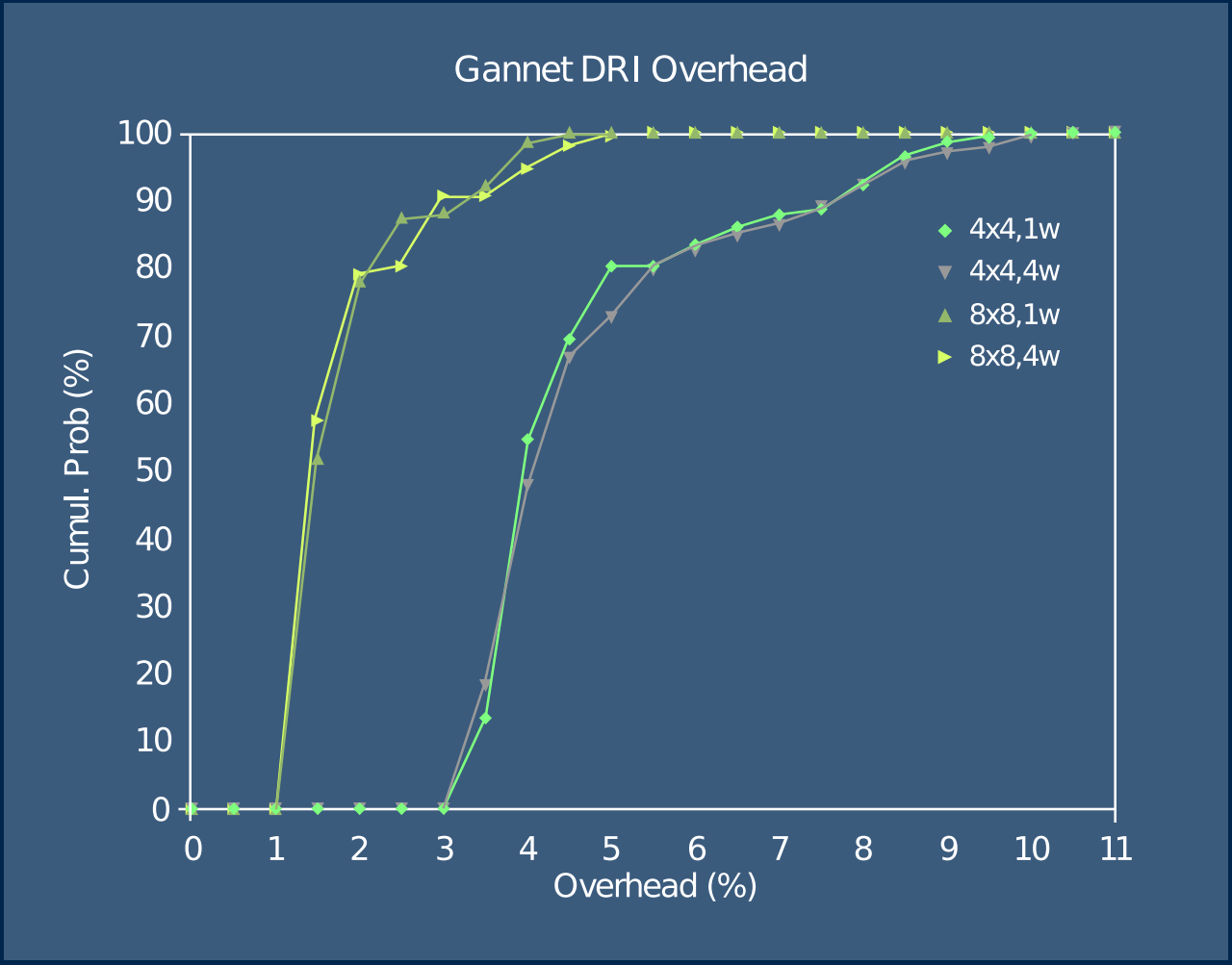
- Monte-Carlo DOE
  - Matrix operations on 8x8 blocks
  - Random valid expressions

Gannet Performance





Gannet Performance



Future Work

- Current service manager is functional, i.e. demand-driven
- Alternative models:
  - Data-drive execution
    (but results in unnecessary processing)
  - Actor model
    (but is more complex, so requires more area)

Future Work

- The Gannet platform can be viewed as a lightweight hardware distributed operating system
- GannetVM can be developped into a fully featured software distributed operating system

Future Work

- High-level language compiler
- Integration of core programs
- Ideally a single language for everything

Summary

- Gannet platform for heterogeneous multicore SoC design
  - programmable interface between cores and communication medium
  - high-level programming of data flows, sophisticated flow control
- Hardware implementation
  - small
  - fast
  - low overhead
- Software implementation (VM)
  - facilitates HW/SW codesign
  - can be developped into a distributed OS

www.gannetcode.org

Gannet System Operation

- The Gannet machine is a distributed computing system where every node (**service**) **consumes packets** and **produces packets** and can store state information between transactions.
- We denote a Gannet packet as $p(Type, To, Ret, Id; Payload)$
  - packet $Types$ are $code$, ref or $data$
- The operation of a Gannet **service** can be described in terms of
  - the **task code**
  - the internal state
  - the **result packet(s)** produced by the task

Gannet System Operation

- SC**: Store code:** service S_i receives a **code** packet $p(code, S_i, S_j, R_{task}; t)$ where $t = (S_i\ a_1 ... a_n)$ and stores it referenced by R_{task}.
- AT**: Activate task :** the service S_i in $state_i$ receives a task **reference** packet $p(ref, S_i, S_j, R_{id}; R_{task})$

  the service activates the task referenced by R_{task}: $(S_i\ a_1 ... a_n)$. This results in evaluation of the arguments $a_1 .. a_n$:
  - DR**: Delegate reference:** the service manager delegates subtasks referenced by **reference sy**mbols via **reference** packets
  - SQ**: Store quoted symbol:** all **quoted** (i.e. constant) symbols in the code ares stored in the local store.
  - SR**: Store returned result:** result data from subtasks are stored in the local store.

Gannet System Operation

- P: **Processing:** When all arguments of the subtask have been evaluated,
  - the data are passed on to the service core (**call**);
  - The core performs processing on the data (**eval**);
  - the service, now in $state'_i$, produces a result packet p_{res} (**return**) $p_{res} = p(Type_i, S_j, S_i, R_{id}; Payload_i)$ where both $Payload_i$ and the state change to $state'_i$ are the result of processing the evaluated arguments $a_1..a_n$ by the core of S_i.
- p_{res} is sent to S_j where $Payload_i$ is stored in a location referenced by R_{id}.