High-level Programming for Specifying Run-time Reconfiguration in Processor Arrays

Authors: Zain-ul-Abdin and

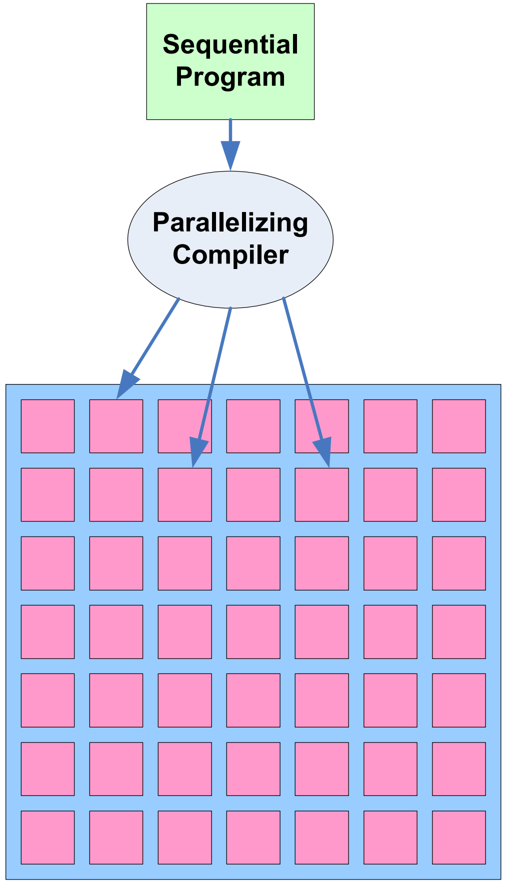
Bertil Svensson

Motivation

- Challenges in the design of High-performance Embedded Systems
  - Real-time performance
  - Energy Efficiency
  - Adaptable functionality
- Emergence of coarse-grained reconfigurable processor arrays
  - Optimized functional units with low silicon cost
  - Adaptable for new functionalities
  - Energy Efficient implementation
    - Controlling clock frequency of individual cores
    - Switching off unused cores

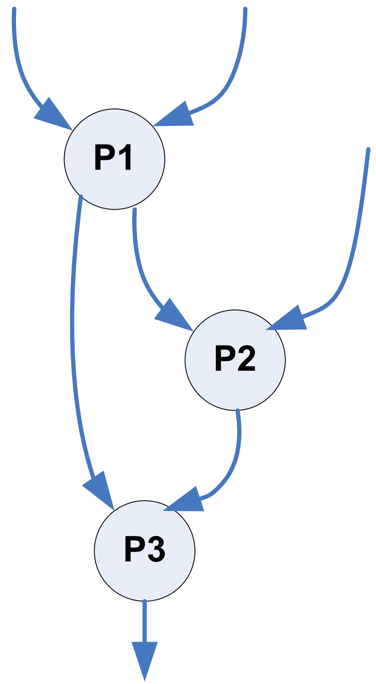
Traditional Approaches

- Imperative languages (C, Pascal)
  - Rely on sequential control flow
  - Intended for algorithm specification
  - Use annotations to adapt to target architecture
- Traditional methods
  - Automatic parallelization by compilers
  - Use of advanced synthesis tools
- Lack of support for expressing dynamic reconfiguration



Our Approach

- Use of Concurrent Programming Model
  - Expresses computations in a productive manner by matching it to target hardware
  - Supported by a compiler for allowing portability
- Occam-pi
  - CSP dataflow
  - Mobility features of pi-calculus
  - Expression of Reconfigurability



Occam-pi Significance

- Explicit concurrency
  - Explicit control of granularity of parallelism and data locality
- Strong encapsulation
- Asynchronous or untimedness
- Language support hides control flow
- Backpressure assures no loss of data
- Abstractions for underlying hardware
  - Processes
  - Channels (Unbuffered message passing)

Mobile data

- Variables can only be written by the owner process i.e., data is strictly private
- In occam-pi, only one name can ever refer to the same object
  - Thus avoiding aliasing
- Automatically guarantees against ***parallel race hazards*** on data access
- Occam-pi introduces `MOBILE` data
  - Ownership of data can be exported between different processes
  - Only one process can hold a given mobile data

Dynamic Process Invocation

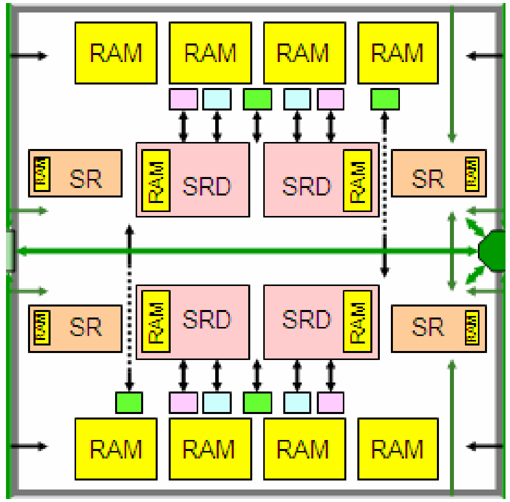
- Occam-pi offers dynamic spawning of processes
- Occam-pi introduces two new keywords – `FORKING` and `FORK`
- Inside a `FORKING` block, we can use `FORK` at any time to spawn a new process
- When the `FORKING` block exits, it'll wait for all the spawned processes to finish
- The arguments to a `FORK`ed process should be:
  - Passed by value (i.e.,VAL)
  - Mobile – where they are then owned by new process

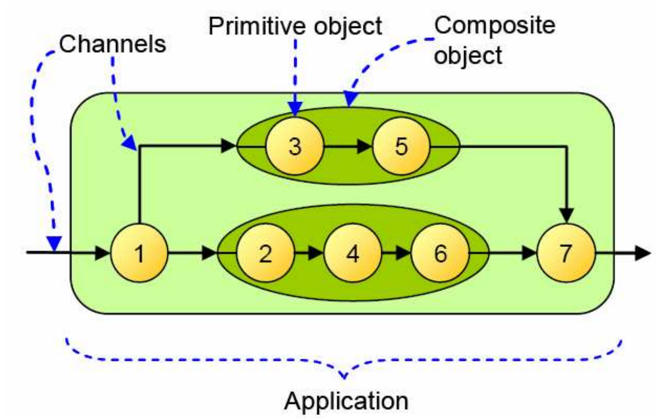
Ambric Arch. & Programming Model

- Occam-pi Code

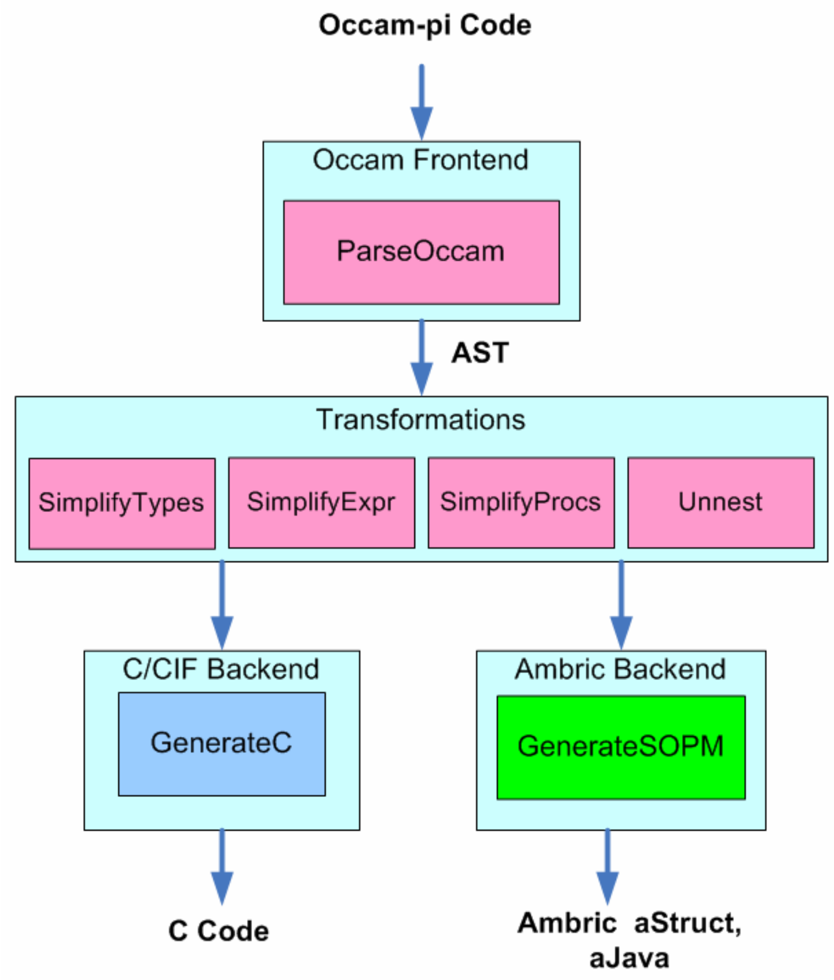
```
PROC SimpleEx()
  INT j,k:
  CHAN INT a,b,c,d:
  PAR
    SEQ j=1 FOR 8
      a ! j
    Square(a?,b!)
    Square(c?,d!)
    SEQ j=1 FOR 8
      d ? k
:

PROC Square(CHAN INT a?,b!)
  INT x,y:
  SEQ
    a ? x
    y = x * x
    b ! y
:
```





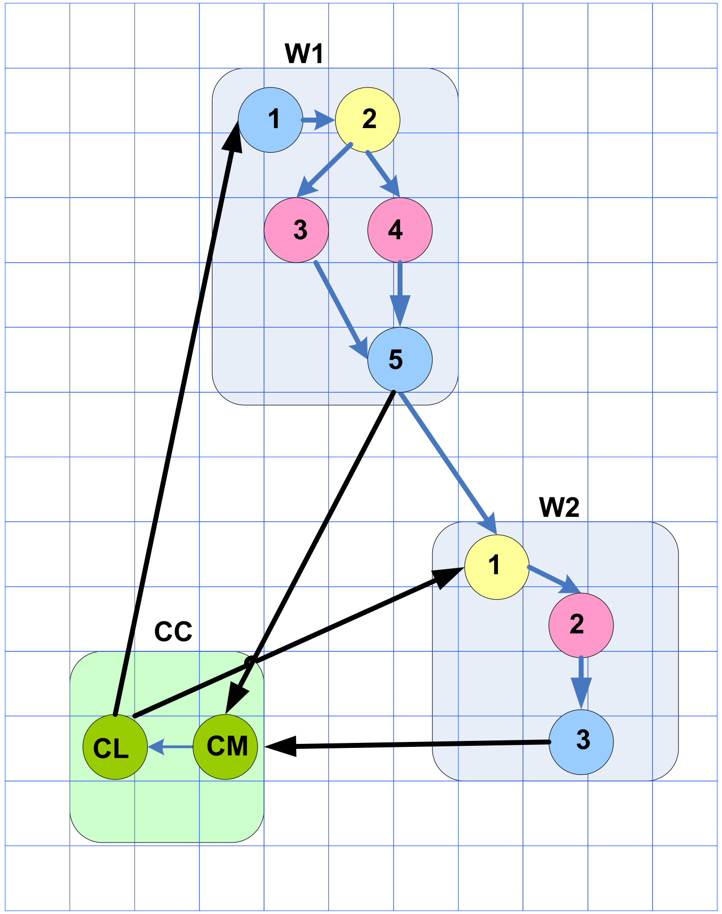
Occam-Ambric Compilation



CERES
CENTRE FOR RESEARCH ON EMBEDDED SYSTEMS

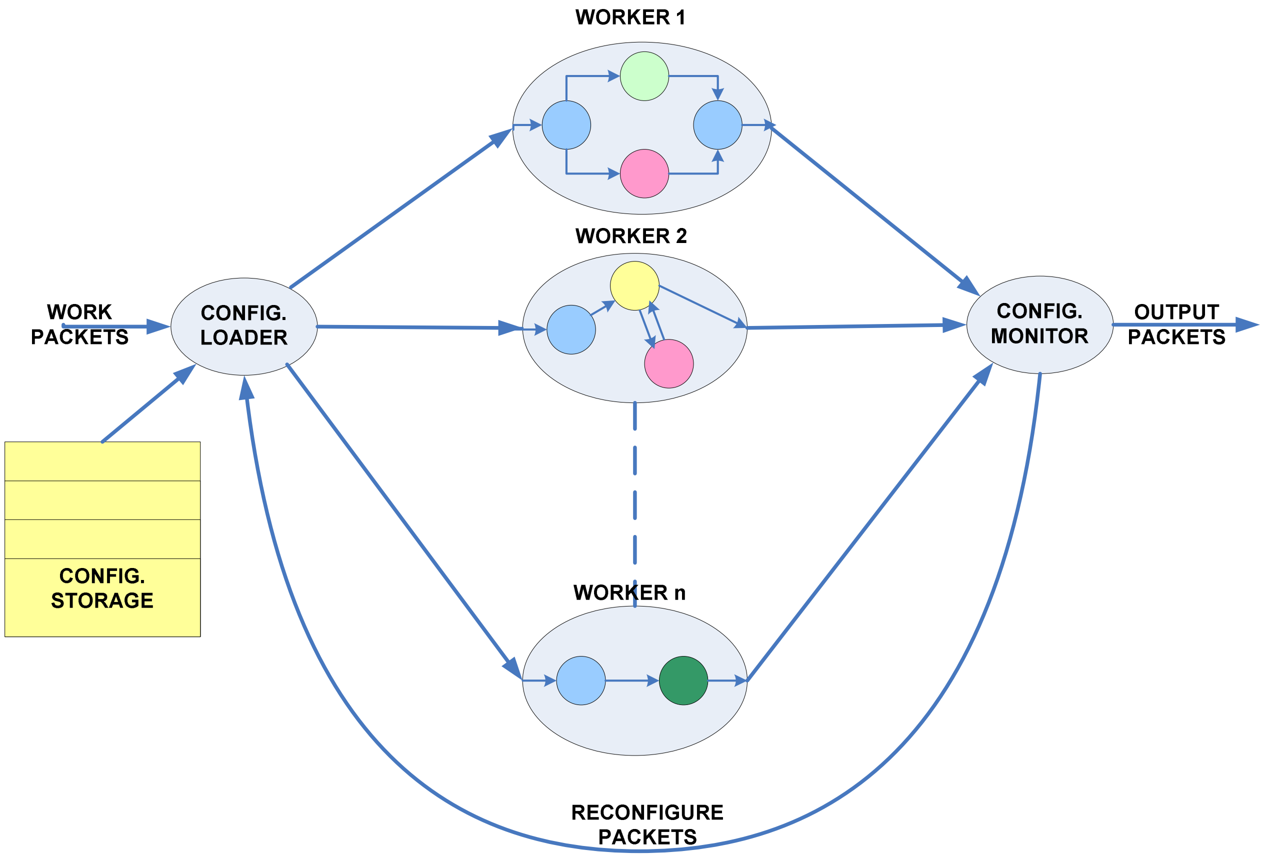
I SAMARBETE MED KK·stiftelsen

Reconfigurable Workers Mapping



W1,W2 → Workers
CC → Configuration Controller
CL → Configuration Loader
CM → Configuration Monitor

Communication b/w Configuration Controller & Workers

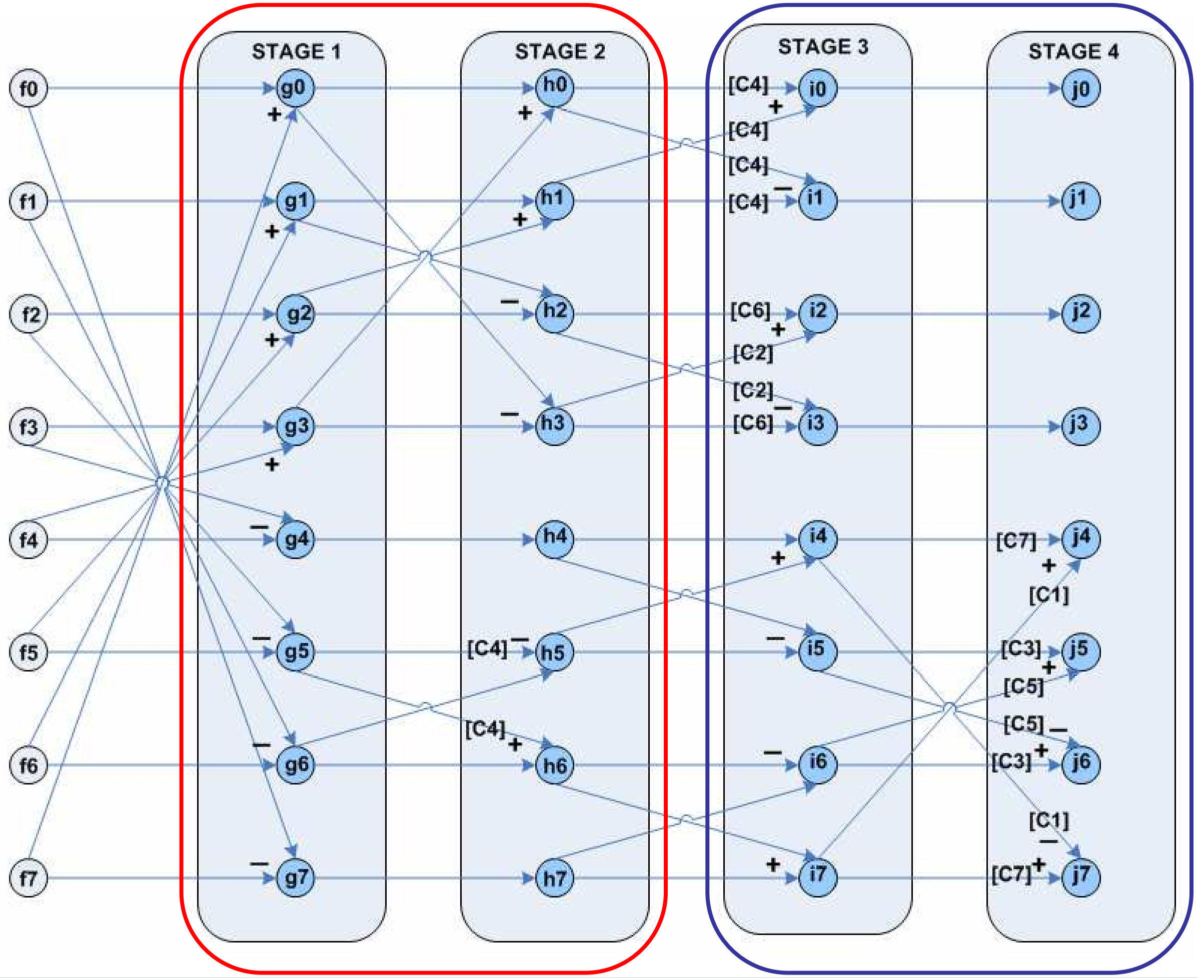


CENTRE FOR RESEARCH ON EMBEDDED SYSTEMS

CERES

I SAMARBETE MED KK·stiftelsen

1D-DCT Case Study



Example Code

```
PROC loader(CHAN INT inp?, CHAN MOBILE INT cnf!,
                CHAN INT ack?)
  INT cstatus, value, id:
  MOBILE [100]INT config:
  CHAN MOBILE INT cnf:
  CHAN INT res:
  VAL RECONFIG IS 255:
  SEQ
    FORKING
      WHILE TRUE
        SEQ
          inp ? value
          cnf ! value
          ack ? cstatus
          IF
            cstatus = RECONFIG
              SEQ
                ack ? id
                IF
                  id = 1
                    FORK task2(config, cnf?, res!)
                  id = 2
                    ...
:
```

(a)

```
PROC monitor(CHAN INT res?, CHAN INT ack!,
                            CHAN INT outp!)
  INT status:
  VAL RECONFIG IS 255:
  WHILE TRUE
    SEQ
      res ? status
      IF
        status = RECONFIG
          ack ! RECONFIG
        status <> RECONFIG
          outp ! status
:
```

(b)

```
PROC task2(MOBILE [100]INT config,
           CHAN MOBILE INT cnf?, CHAN INT res!)
  CHAN INT ch:
  PLACED PAR
    PROCESSOR 1,1
      stage3(config,cnf?,ch!)
    PROCESSOR 1,2
      stage4(config,ch?,res!)
:
```

(c)

Implementation Results

No. of Configuration words = 97

DCT Implementations	Cycle Counts
4-Processor DCT	1340
2-Processor Reconfigurable (including reconfiguration time)	2612
Reconfiguration time	550

Conclusions & Future Work

- Application development based on concurrent computation models (Streaming/CSP/KPN)
  - Raises the abstraction level while not compromising the performance
  - Able to express dynamic reconfiguration
- Extend the Compiler framework to provide:
  - Target-specific partitioning techniques
- Evaluation of the framework with multiple signal processing applications and for multiple target architectures

Thank you for your attention!