MCGREP

A Predictable Architecture for Embedded Real-Time Systems

Jack Whitham and Neil Audsley

`jack@cs.york.ac.uk`

Real Time Systems Group

Department of Computer Science

University of York

Overview

- A real-time problem,
- The proposed solution and its features,
- Initial implementation and experiments,
- Results,
- Conclusion and future work.

A Real-Time Problem

We always need more computing power!

Requirements for a processor for a **real-time** system:

A Real-Time Problem

We always need more computing power!
Requirements for a processor for a **real-time** system:

- Be Predictable

  (part of ensuring correctness)

A Real-Time Problem

We always need more computing power!
Requirements for a processor for a **real-time** system:

- Be Predictable

  (part of ensuring correctness)

- Be Fast

  (run more demanding applications)

These conflict!

A Processor

Parallel problem: Portable embedded devices have limited electrical power.

Requirements for a processor for an **embedded** system:

A Processor

Parallel problem: Portable embedded devices have limited electrical power.

Requirements for a processor for an **embedded** system:

- Be Low Power Consumption
  (conserve battery life)

A Processor

Parallel problem: Portable embedded devices have limited electrical power.

Requirements for a processor for an **embedded** system:

- Be Low Power Consumption

  (conserve battery life)
- Be Fast

  (run more demanding applications)

These also conflict!

Our Problem

Real-time systems need the following in the future:

- Low Power Consumption
  (conserve battery life)
- Predictability
  (ensure correctness)
- Execution Speed
  (run more demanding applications)

Custom Hardware

Application-specific hardware:

- Accelerates processing bottlenecks (“hotspots”),

Custom Hardware

Application-specific hardware:

- Accelerates processing bottlenecks (“hotspots”),
- Not reusable in general,

Custom Hardware

Application-specific hardware:

- Accelerates processing bottlenecks (“hotspots”),
- Not reusable in general,
- Not scalable in general,

Custom Hardware

Application-specific hardware:

- Accelerates processing bottlenecks (“hotspots”),
- Not reusable in general,
- Not scalable in general,
- Introduces co-design problem.

Reconfigurable hardware

Create “virtual hardware” devices on one reprogrammable array.

- Adapt to mode changes,

Reconfigurable hardware

Create “virtual hardware” devices on one reprogrammable array.

- Adapt to mode changes,
- Support many more hotspots,

Reconfigurable hardware

Create “virtual hardware” devices on one reprogrammable array.

- Adapt to mode changes,
- Support many more hotspots,
- Introduces **configuration generation** problem,

Reconfigurable hardware

Create “virtual hardware” devices on one reprogrammable array.

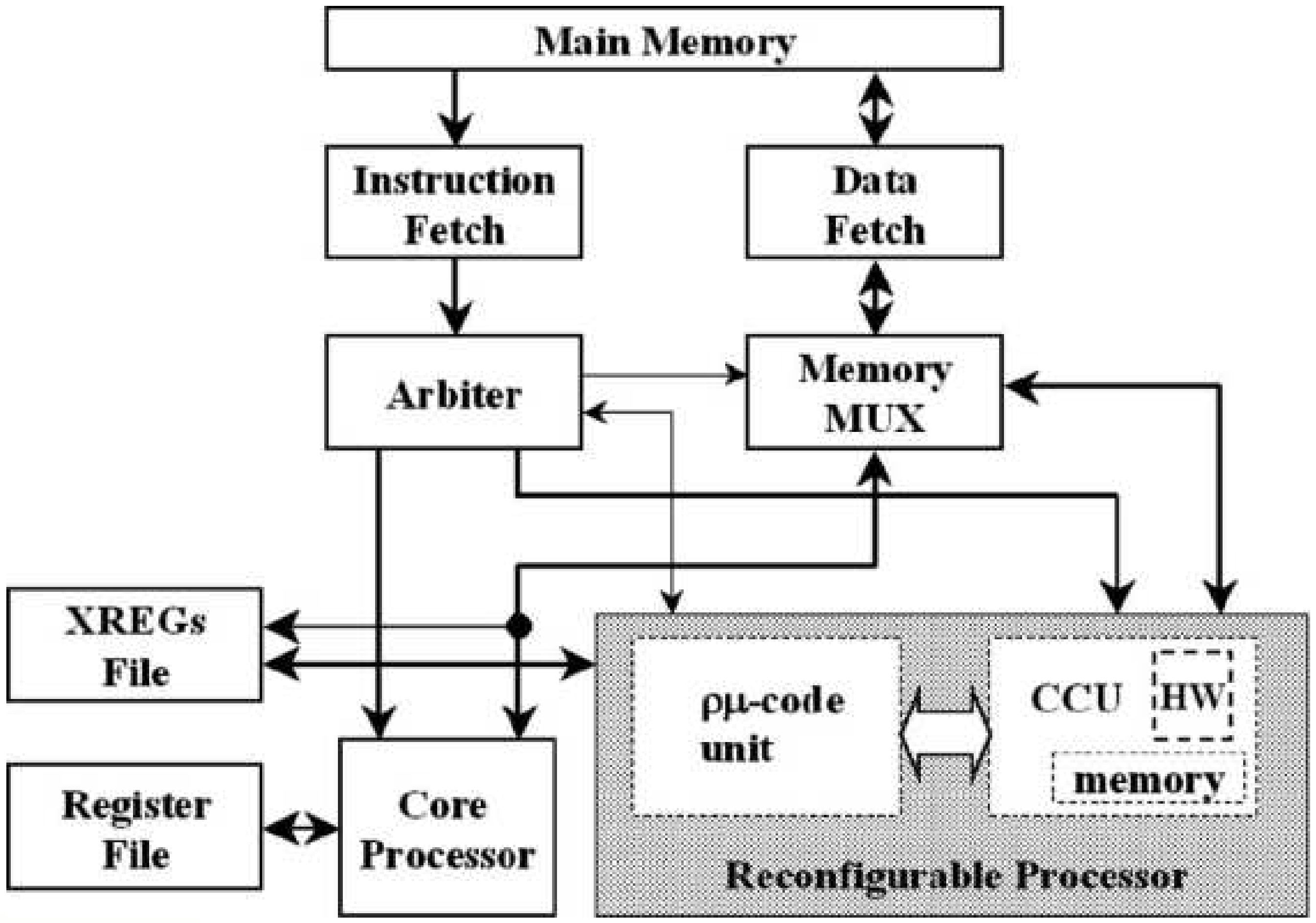
- Adapt to mode changes,
- Support many more hotspots,
- Introduces **configuration generation** problem,
- Introduces **compatibility** problem,

Reconfigurable hardware

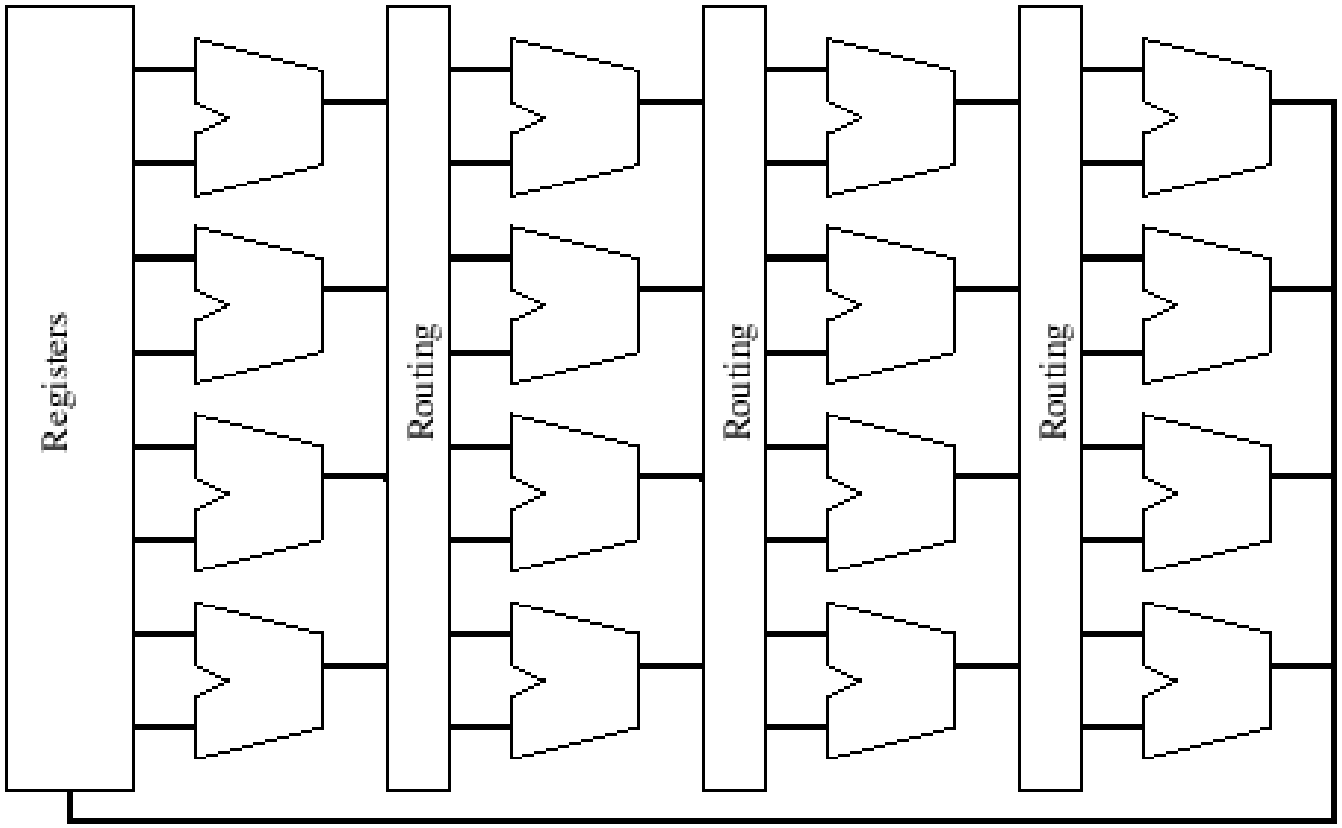
Create “virtual hardware” devices on one reprogrammable array.

- Adapt to mode changes,
- Support many more hotspots,
- Introduces **configuration generation** problem,
- Introduces **compatibility** problem,
- Introduces **predictability** problem.

Fine-grained array [25]



Coarse-grained array [9, 24]



Proposed Solution

Microcoded **c**oarse-**g**rained **re**configurable **p**rocessor: software controlled CPU

- Based on Coarse-grained Array,

Proposed Solution

Microcoded **c**oarse-**g**rained **re**configurable **p**rocessor: software controlled CPU

- Based on Coarse-grained Array,
- Many reprogrammable processors,

Proposed Solution

Microcoded **c**oarse-**g**rained **re**configurable **p**rocessor: software controlled CPU

- Based on Coarse-grained Array,
- Many reprogrammable processors,
- Each can run programs from external memory and from internal microcode.

MCGREP Paradigm

- Applications execute from external memory, **except** when hotspots are reached.

MCGREP Paradigm

- Applications execute from external memory, **except** when hotspots are reached.
- Hotspot tasks are distributed to all or part of the reconfigurable array.

MCGREP Paradigm

- Applications execute from external memory, **except** when hotspots are reached.
- Hotspot tasks are distributed to all or part of the reconfigurable array.
- ILP exploited as multiple processors may be used to execute each task.

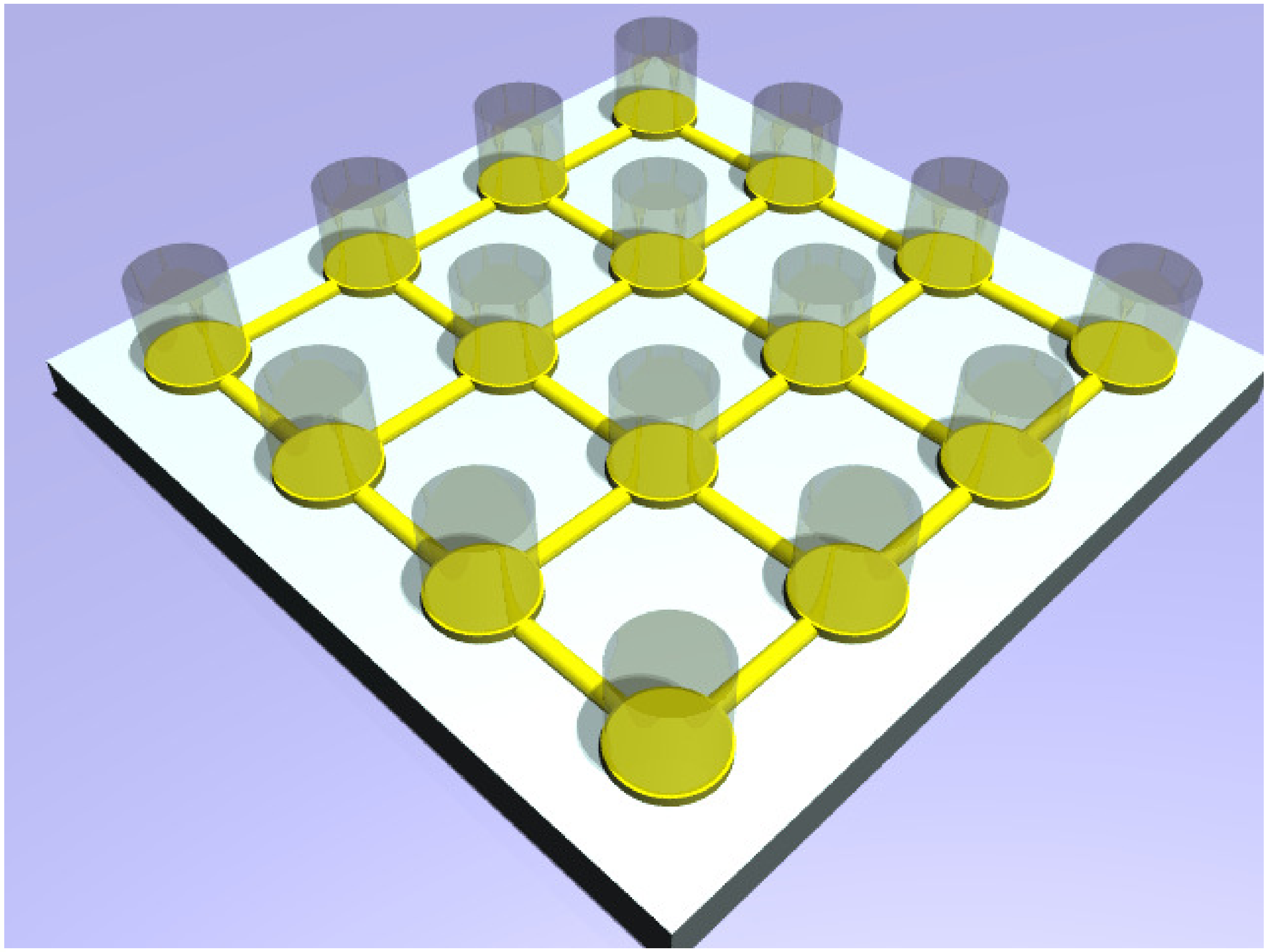
MCGREP Paradigm

- Applications execute from external memory, **except** when hotspots are reached.
- Hotspot tasks are distributed to all or part of the reconfigurable array.
- ILP exploited as multiple processors may be used to execute each task.
- Applications are speeded up!

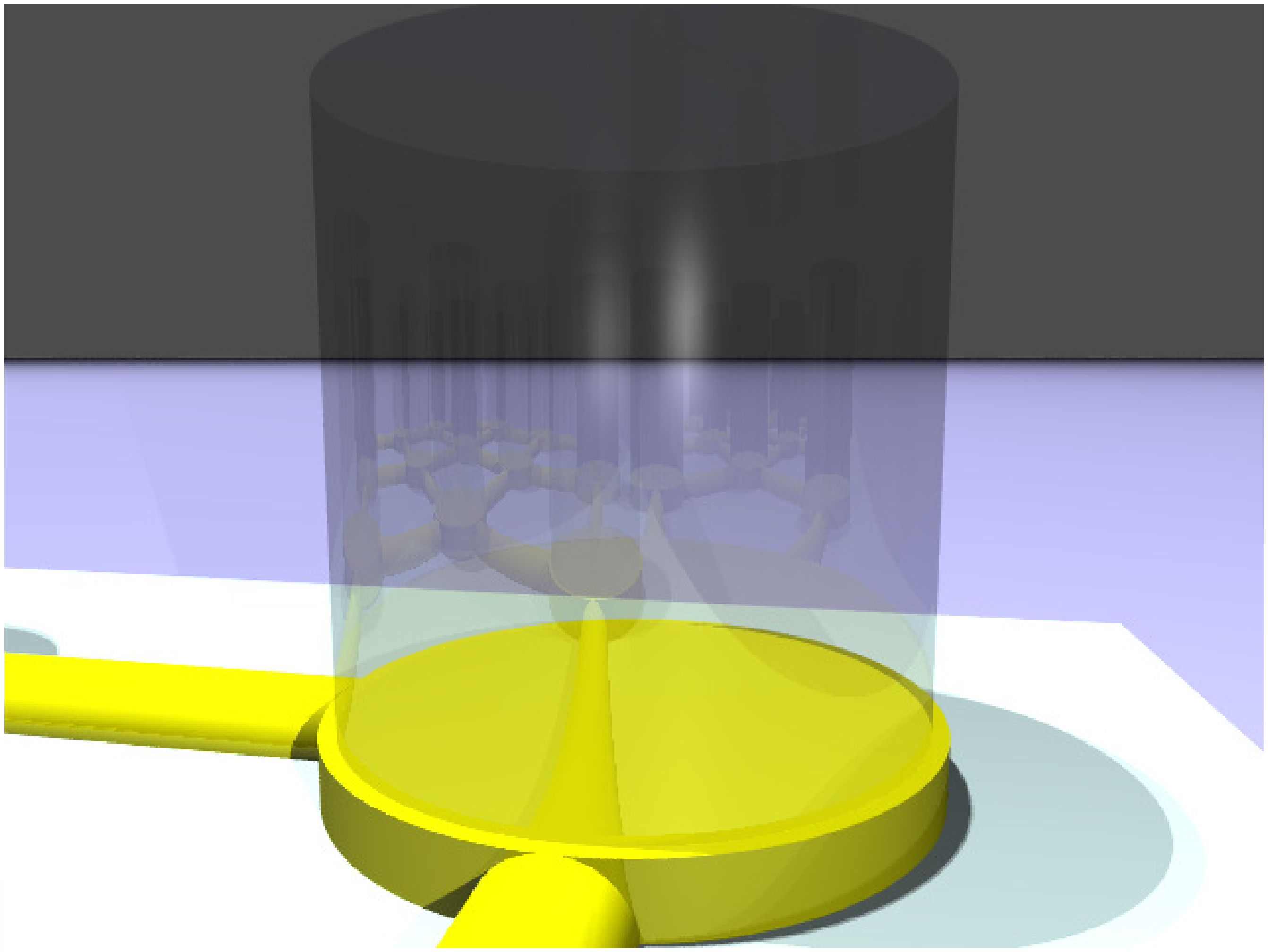
MCGREP Paradigm

- Applications execute from external memory, **except** when hotspots are reached.
- Hotspot tasks are distributed to all or part of the reconfigurable array.
- ILP exploited as multiple processors may be used to execute each task.
- Applications are speeded up!
- Predictability and scalability retained.

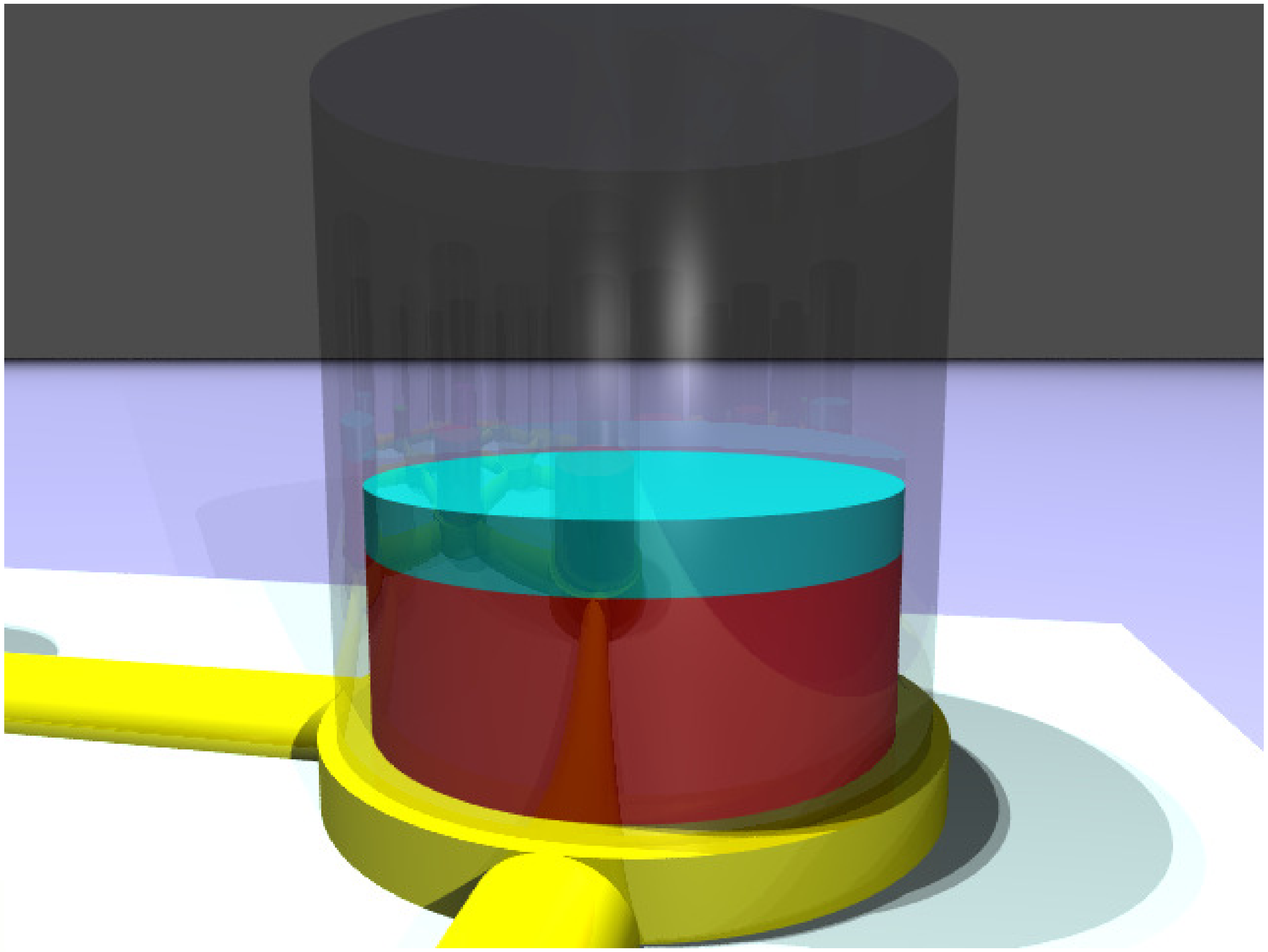
Distributing Applications (1)



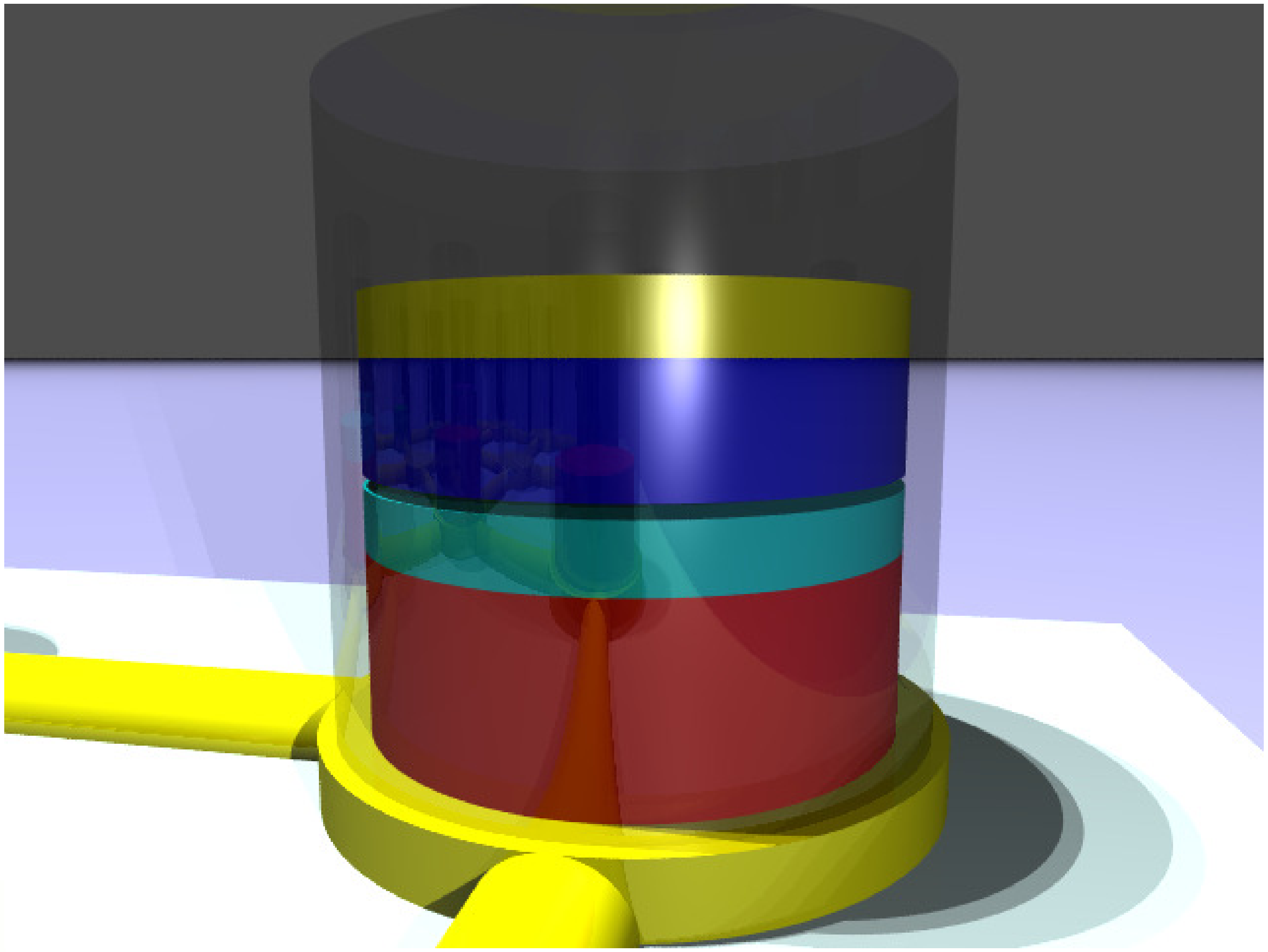
Distributing Applications (2)



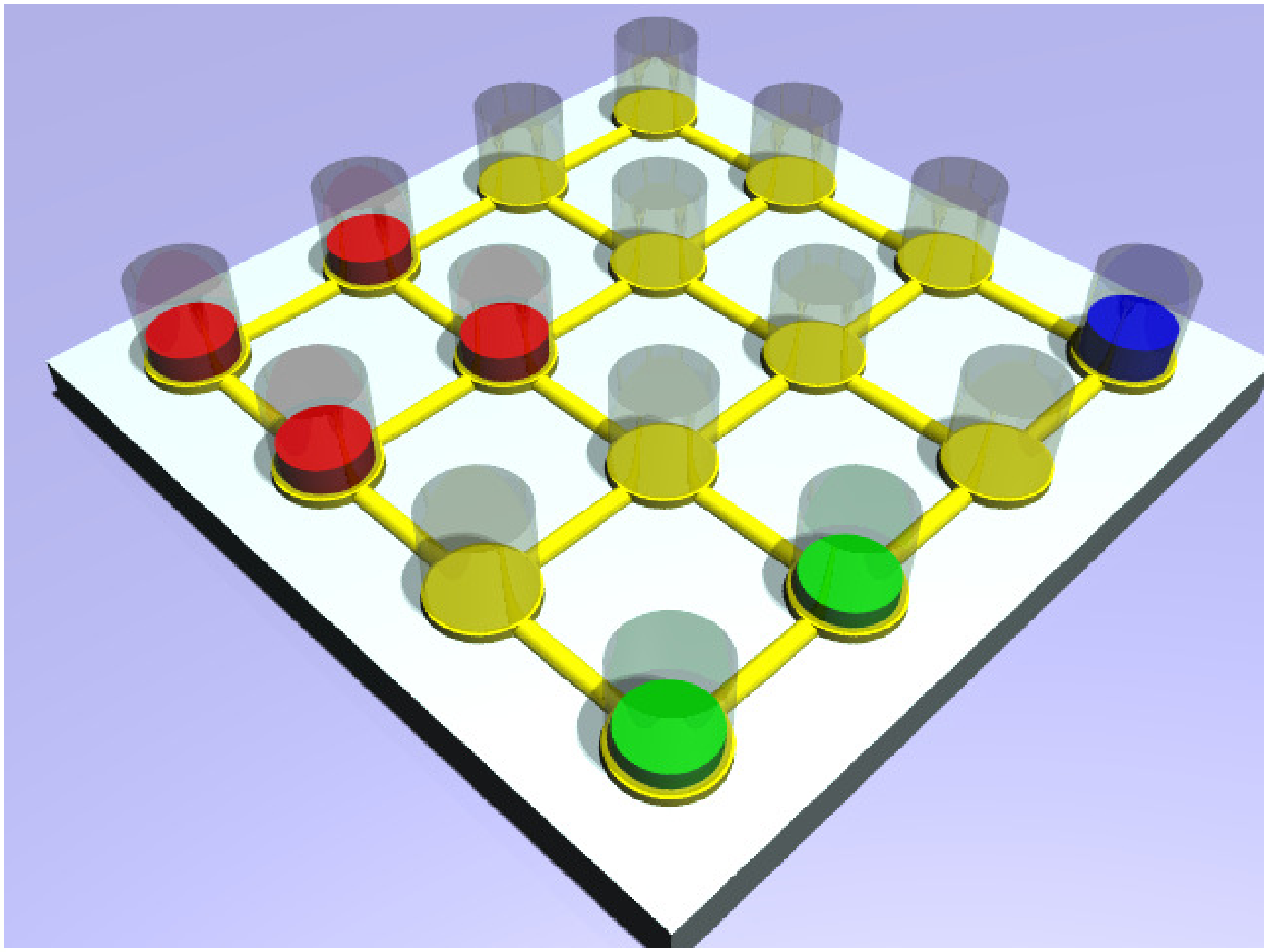
Distributing Applications (3)



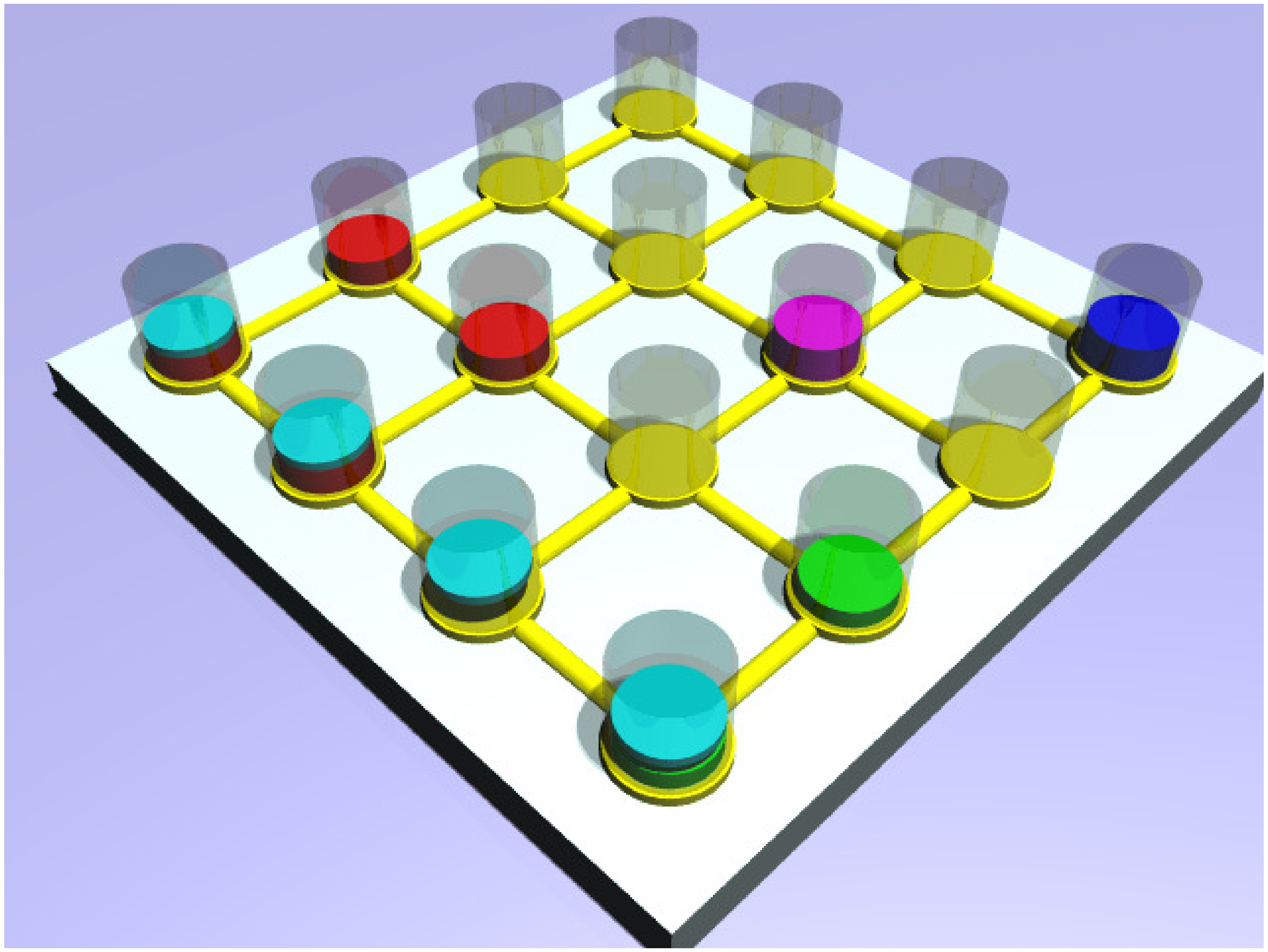
Distributing Applications (4)



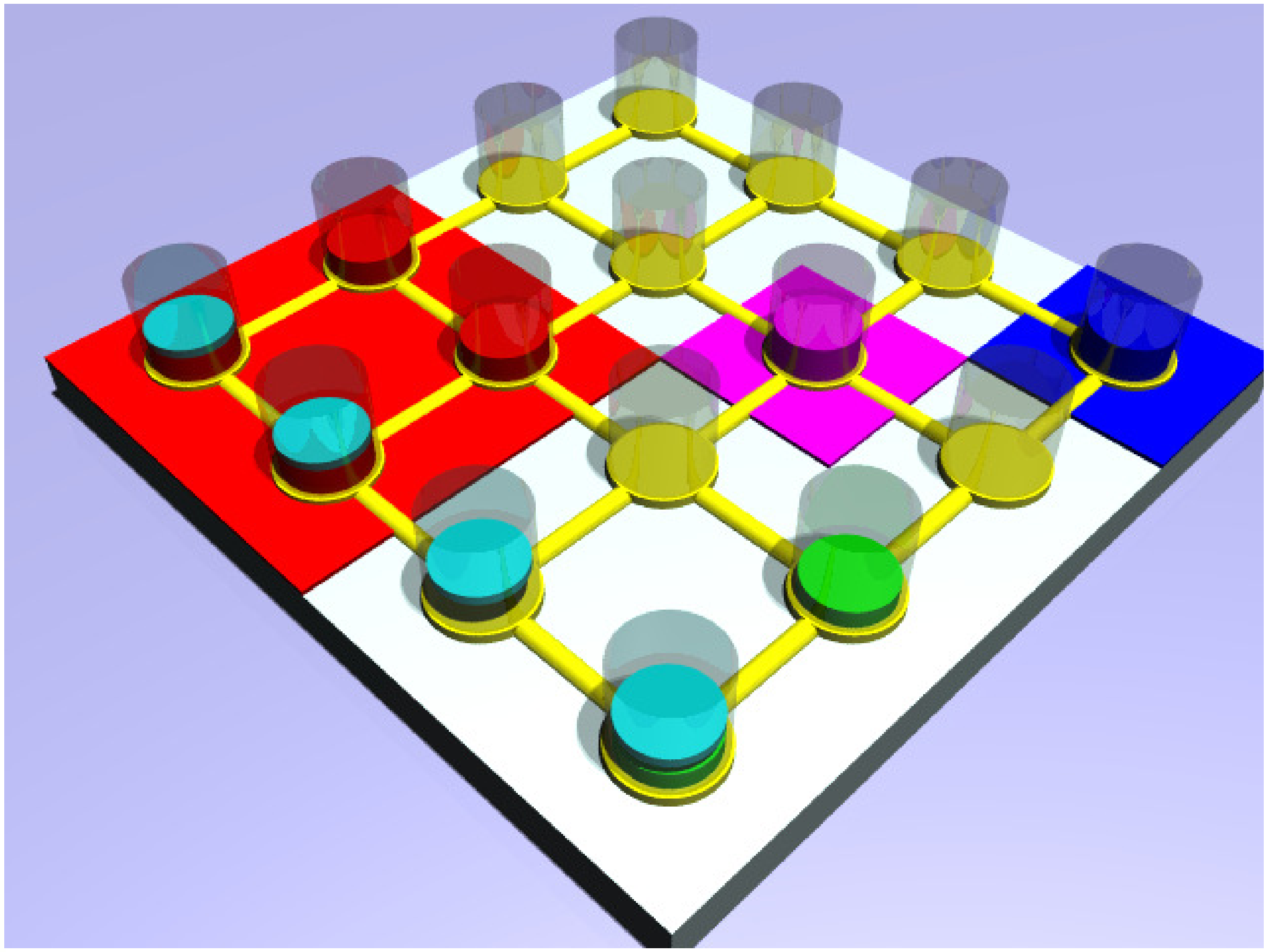
Distributing Applications (5)



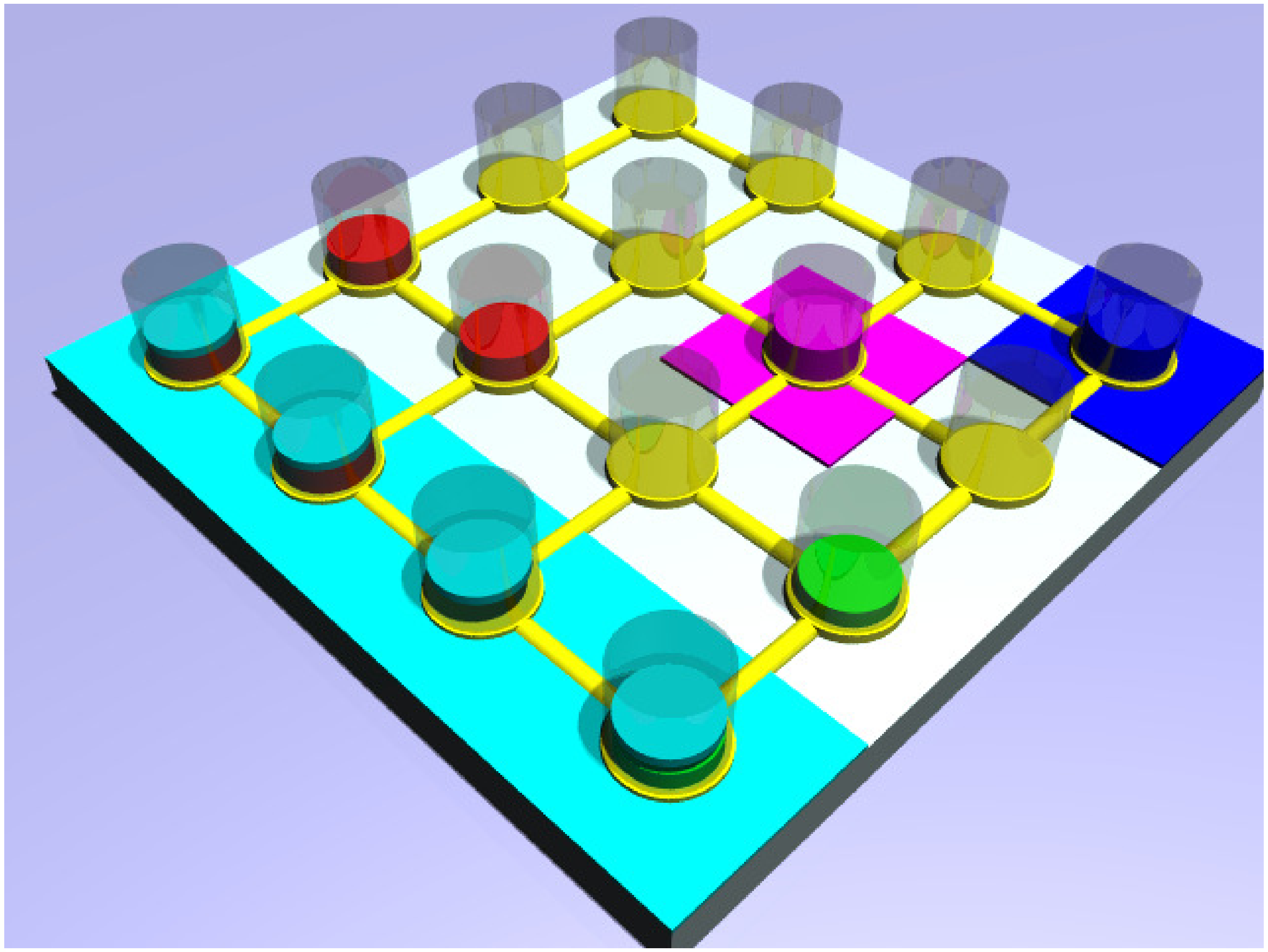
Distributing Applications (6)



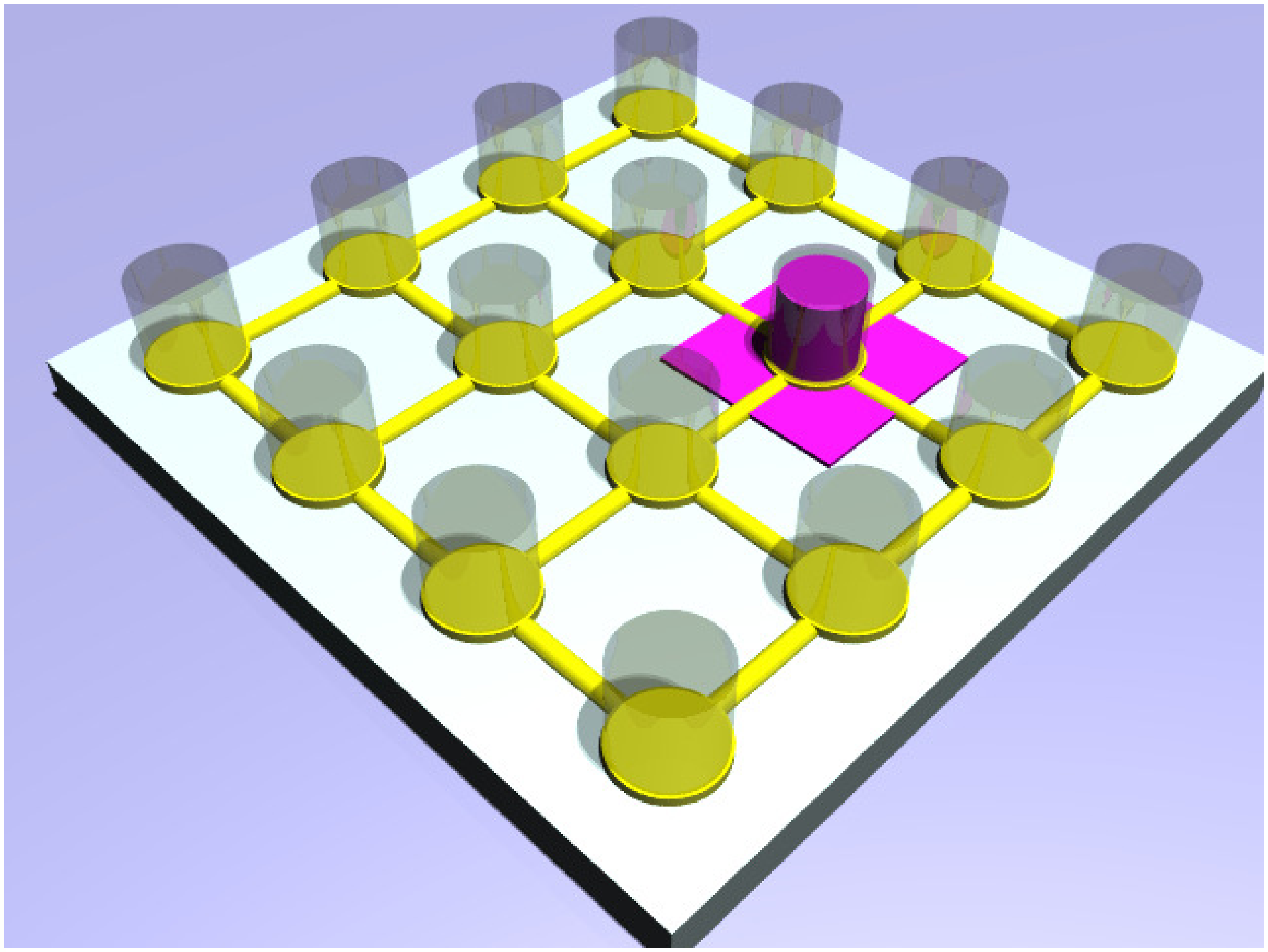
Distributing Applications (7)



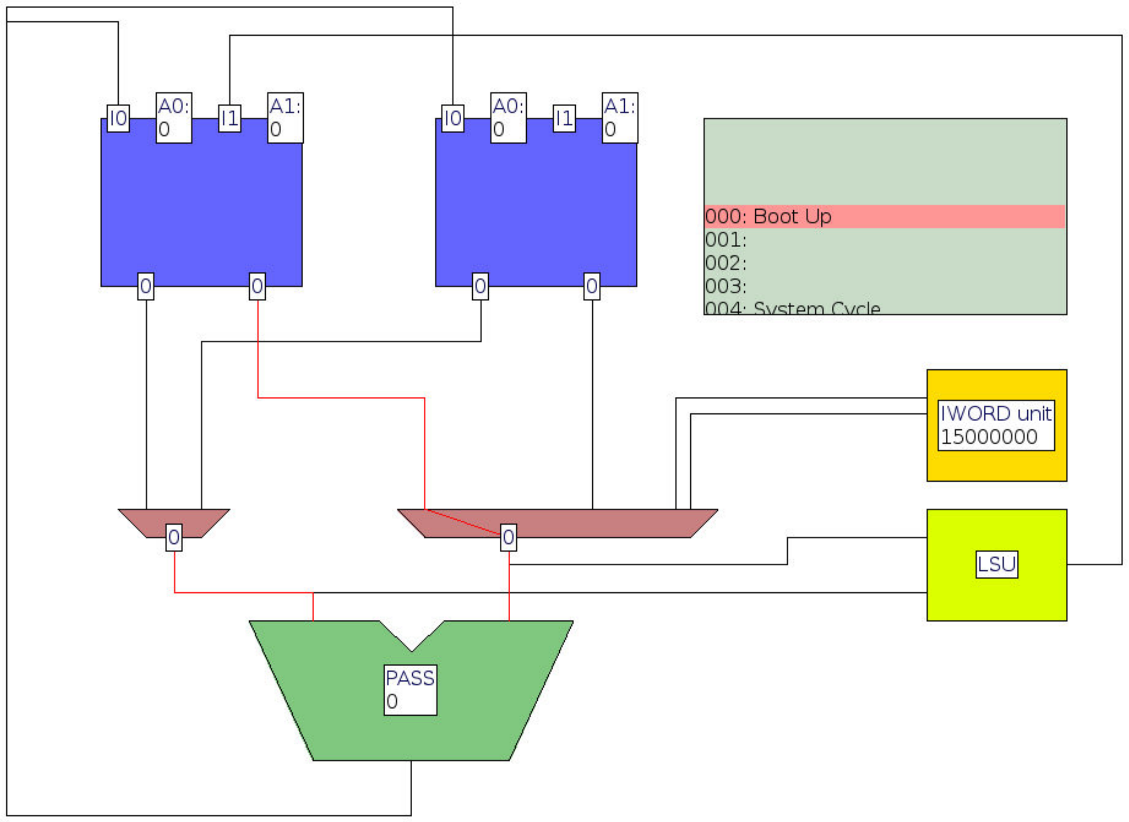
Distributing Applications (8)



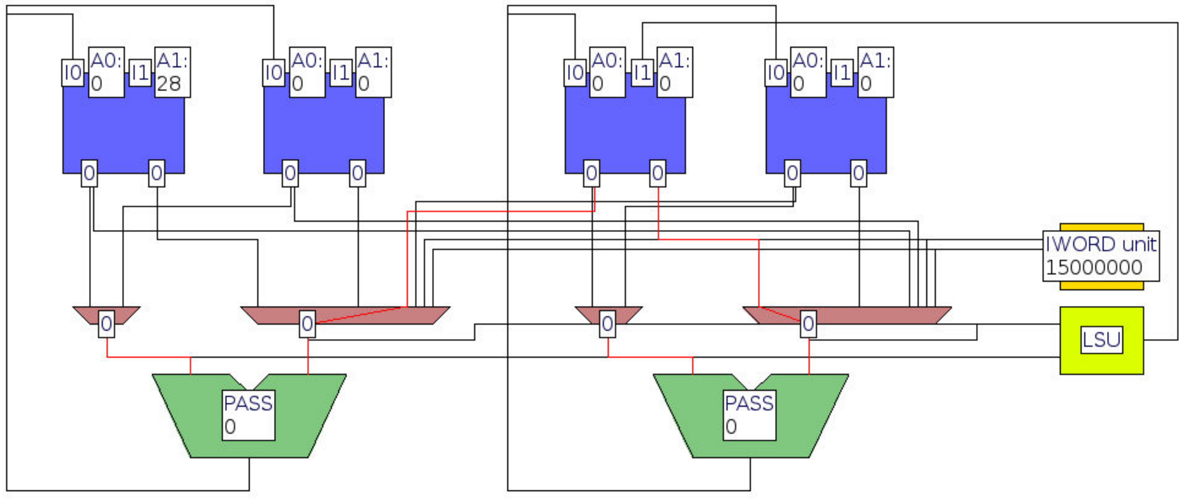
Distributing Applications (9)



How it works


I0
A0:
0
I1
A1:
0
I0
A0:
0
I1
A1:
0
0
0
0
0
000: Boot Up
001:
002:
003:
004: System Cycle
IWORD unit
15000000
0
0
LSU
PASS
0

How it works



000: Boot Up
001:
002:
003:
004: System Cycle
005: Reenter Machine Code

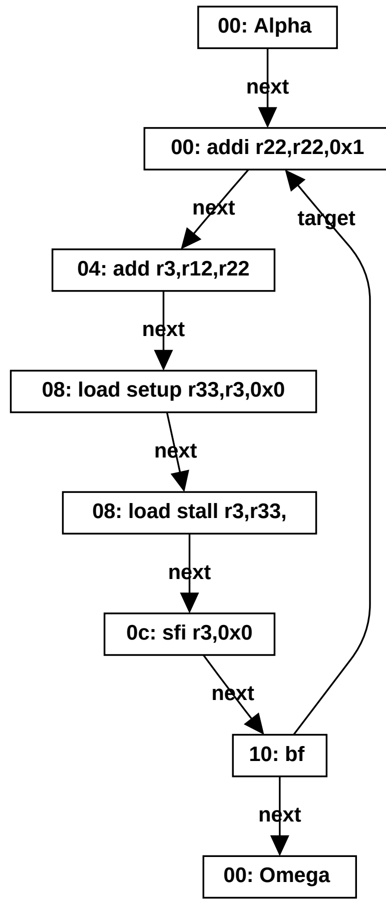
Making microcode

Input: **machine code**,

1. Generate operation graph,
2. Re-schedule operations to make best use of n processors,
3. Convert schedule to microcode for each processor.

Making microcode (1)

1: Generate operation graph.



```
 0: 9e d6 00 01 l.addi r22,r22,0x1
 4: e0 6c b0 00 l.add r3,r12,r22
 8: 90 63 00 00 l.lbs r3,0x0(r3)
 c: bc 23 00 00 l.sfnei r3,0x0
10: 13 ff ff fc l.bf 0x0
14: 15 00 00 00 l.nop 0x0
```

Making microcode (2)

2: Re-schedule operations to make best use of n processors.
Sample ($n = 2$):

	Node 0	Node 1
Time 0	load setup	nop
	address $\leftarrow r252$	
Time 1	load stall	nop
	$r250 \leftarrow$ output	
Time 2	add	compare
	$r254 \leftarrow r254 + 1$	$r244 \leftarrow r250 \neq 0$
Time 3	add	branch
	$r252 \leftarrow r248 + r254$	$r244$

Making microcode (3)

3: Schedule converted to microcode for each processor.

Each micro-instruction includes:

- Settings for multiplexers,
- Register file commands,
- ALU commands,
- Branch unit commands.

Sample microcode

```
              node 0                     node 1

04    16000245 03b801f9  00000200 00b80000
05    0c000245 03b801fb  00000200 00b80000
06    0cfe8245 03b801fd  00fe8245 00b801fd
07    00ff8245 03b861ff  00ff8245 00b801ff
08    00000200 03b86000  00fc824d 00b801f9
09    00000200 03b80000  00fd824d 00b801fb
0a    00000200 03b85e4e  00000200 00b80000
```

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches,

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches, ...or at least cache replacement strategies that always lead to known states,

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches, ...or at least cache replacement strategies that always lead to known states,
- No dynamic branch prediction,

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches, ...or at least cache replacement strategies that always lead to known states,
- No dynamic branch prediction,
- No shortcuts in hardware implementation,

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches, ...or at least cache replacement strategies that always lead to known states,
- No dynamic branch prediction,
- No shortcuts in hardware implementation,
- In-order execution,

Predictability retained...

At the **microarchitecture** level...

(from Heckmann *et. al.*[10])

- No caches, ...or at least cache replacement strategies that always lead to known states,
- No dynamic branch prediction,
- No shortcuts in hardware implementation,
- In-order execution,
- Simple pipeline.

Predictability retained...

And at the **array** level:

- Array composed of predictable parts,

Predictability retained...

And at the **array** level:

- Array composed of predictable parts,
- Applications scheduled in predictable fashion,

Predictability retained...

And at the **array** level:

- Array composed of predictable parts,
- Applications scheduled in predictable fashion,
- Microcode for processors generated by predictable algorithms.

Predictability retained...

And at the **array** level:

- Array composed of predictable parts,
- Applications scheduled in predictable fashion,
- Microcode for processors generated by predictable algorithms.

Result: No special WCET analysis difficulty is introduced.

This Implementation

Implemented on FPGA and in software simulator.

- Only two processors, permanently locked together,

This Implementation

Implemented on FPGA and in software simulator.

- Only two processors, permanently locked together,
- Microcode task fragments generated by partly manual process,

This Implementation

Implemented on FPGA and in software simulator.

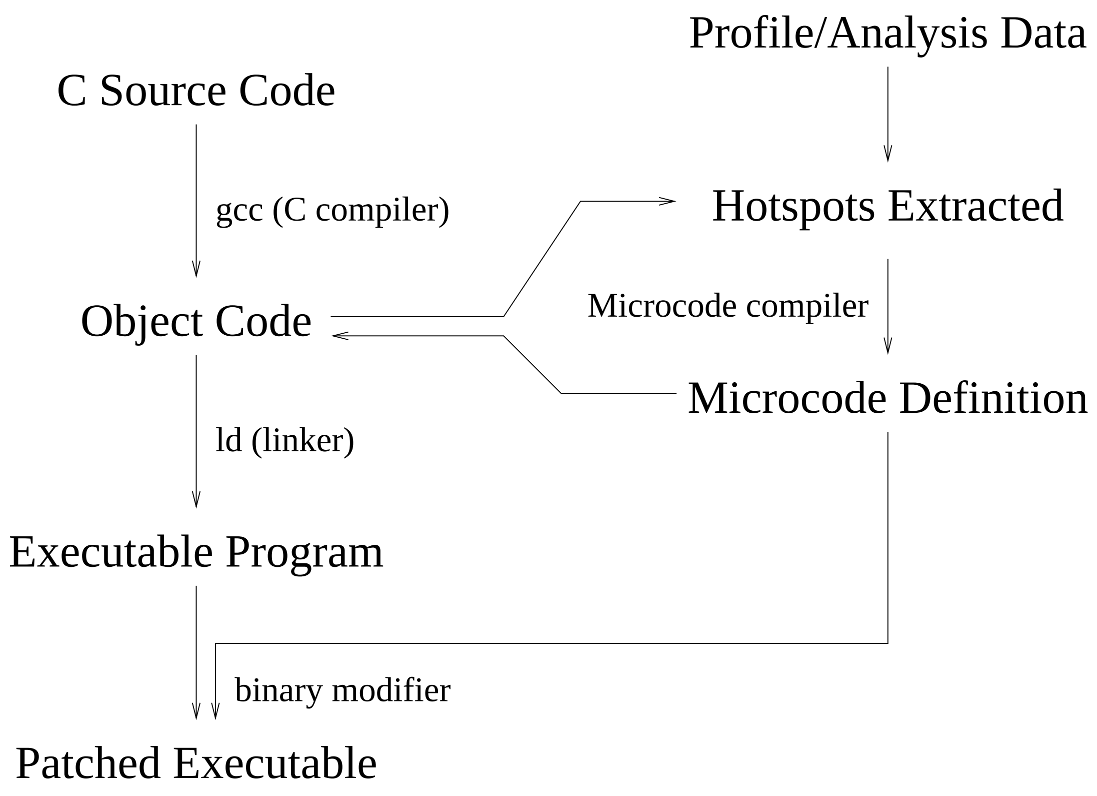
- Only two processors, permanently locked together,
- Microcode task fragments generated by partly manual process,
- No predication,

This Implementation

Implemented on FPGA and in software simulator.

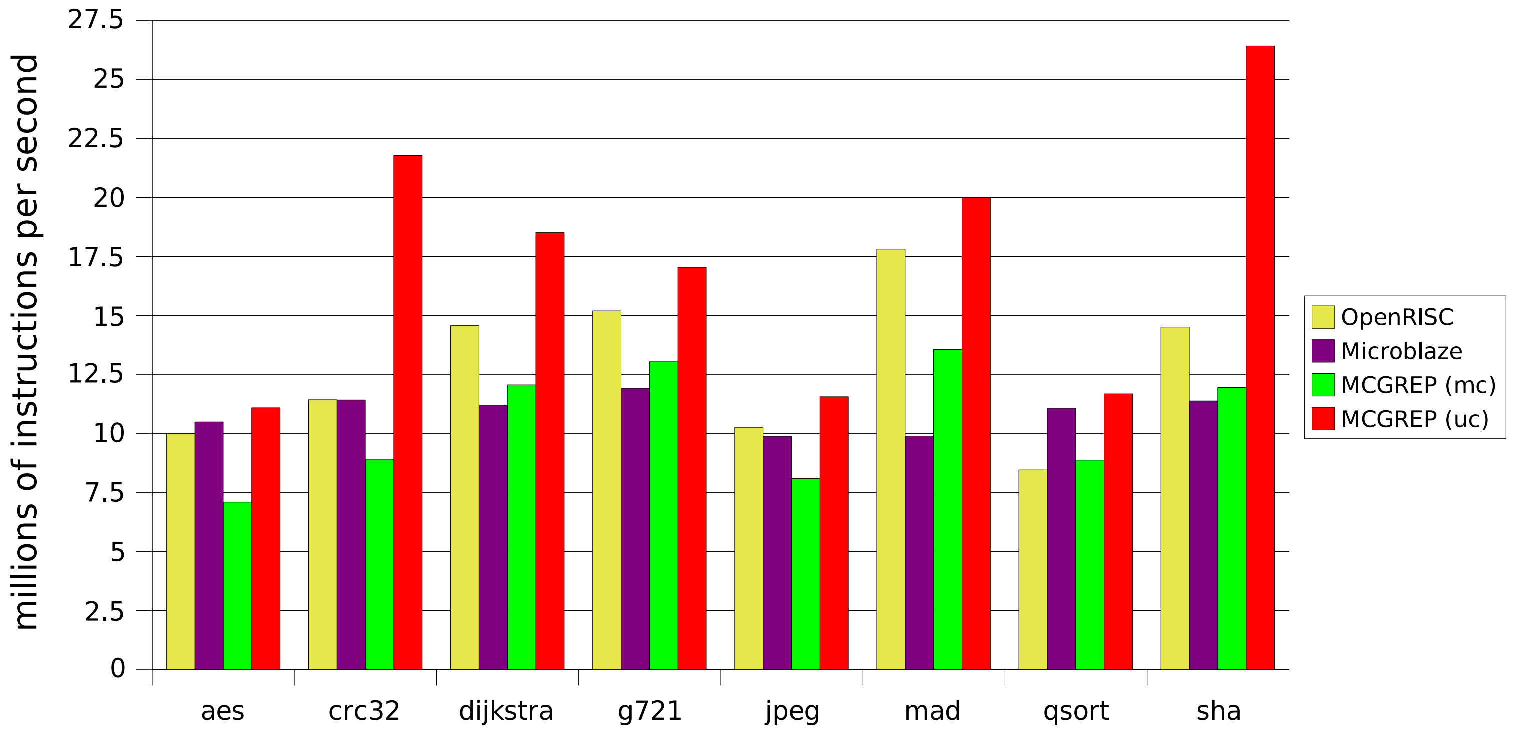
- Only two processors, permanently locked together,
- Microcode task fragments generated by partly manual process,
- No predication,
- Executes either RISC code (via interpreter microprogram) or **task fragments** written in microcode.

Toolchain


Profile/Analysis Data
C Source Code
gcc (C compiler)
Hotspots Extracted
Microcode compiler
Object Code
Microcode Definition
ld (linker)
Executable Program
binary modifier
Patched Executable

Evaluation

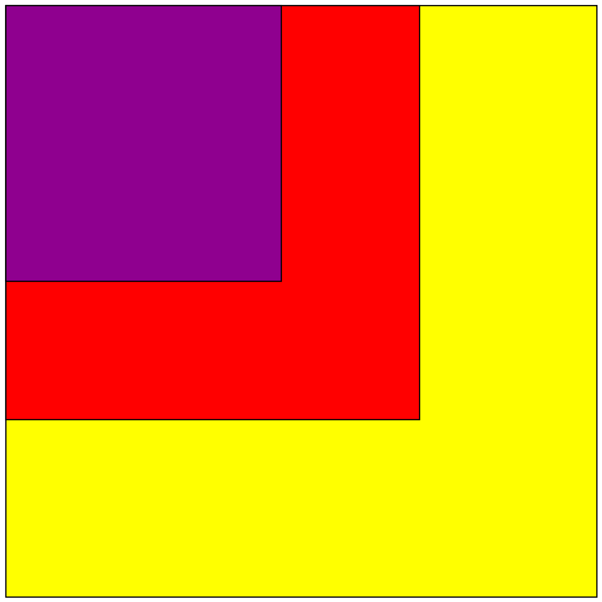
1. Instruction Throughput versus Hardware Size Comparison by experiment:



Evaluation

1. Instruction Throughput versus Hardware Size

Relative FPGA areas:



🟪	Microblaze
🟥	MCGREP
🟨	OpenRISC

Evaluation

2. Instruction Throughput versus Predictability -

- Task execution time independent of hardware state: all instructions have a fixed execution time.

Evaluation

2. Instruction Throughput versus Predictability -

- Task execution time independent of hardware state: all instructions have a fixed execution time.
- Execution highly resistant to interference: experimental evidence is given in the paper.

Evaluation

2. Instruction Throughput versus Predictability -

- Task execution time independent of hardware state: all instructions have a fixed execution time.
- Execution highly resistant to interference: experimental evidence is given in the paper.

WCET analysis does not need to track the hardware state.

Desirable Properties

There is a mechanism for generating very low level microcode from machine code...

can this be used for anything else?

Desirable Properties

Define critical RTOS components in assembly or C, then turn them into microcode, *e.g.*

Desirable Properties

Define critical RTOS components in assembly or C, then turn them into microcode, *e.g.*

- Microcoded context switcher,

Desirable Properties

Define critical RTOS components in assembly or C, then turn them into microcode, *e.g.*

- Microcoded context switcher,
- Microcoded interrupt service routine,

Desirable Properties

Define critical RTOS components in assembly or C, then turn them into microcode, *e.g.*

- Microcoded context switcher,
- Microcoded interrupt service routine,
- Any atomic operation,

Desirable Properties

Define critical RTOS components in assembly or C, then turn them into microcode, *e.g.*

- Microcoded context switcher,
- Microcoded interrupt service routine,
- Any atomic operation,
- Immediate priority ceiling protocol.

Paper includes example microcode for these.

Conclusion

- New implementation option for real-time/embedded systems,

Conclusion

- New implementation option for real-time/embedded systems,
- Compares well to existing soft cores,

Conclusion

- New implementation option for real-time/embedded systems,
- Compares well to existing soft cores,
- Predictable speedup and flexible architecture offered.

Conclusion

- New implementation option for real-time/embedded systems,
- Compares well to existing soft cores,
- Predictable speedup and flexible architecture offered.

Future plans - to experiment with...

- Larger versions of architecture,

Conclusion

- New implementation option for real-time/embedded systems,
- Compares well to existing soft cores,
- Predictable speedup and flexible architecture offered.

Future plans - to experiment with...

- Larger versions of architecture,
- Multitasking, dynamic compilation and 2D scheduling.

Questions?

`jack@cs.york.ac.uk`

Real Time Systems Group
Department of Computer Science
University of York