A Novel Parallel Deadlock Detection Algorithm and Architecture

Pun H. Shiu[2], Yudong Tan[2],

Vincent J. Mooney III[1]

{ship, ydtan, mooney}@ece.gatech.edu

http://codesign.ece.gatech.edu

[1,2]Hardware/Software RTOS Group

[1]Low Power Compiler Group

[1]Assistant Professor, [1,2]Electrical and Computer Engineering

[1]Adjunct Assistant Professor, [1]College of Computing

Georgia Institute of Technology

Atlanta, GA USA

[1]http://crest.ece.gatech.edu

Georgia Tech

Overall Outline

- Motivation - Technology Trends
- Background - Deadlock Detection
- Parallel Algorithm
- Parallel Architecture
- Experimental Results
- Conclusion

Motivation - Technology Trends

- Many of today's chip designs contain 2 processors, e.g., a DSP and a microcontroller
- Future SoC designs are likely to include
  - 4-40 heterogeneous processors
  - 10-50 on-chip hardware resources
    - FFT, Viterbi filter, wireless communication
  - Multithreaded software which dynamically requests and uses the resources

SoC Software

- Ideally, programmers of such future SoC designs would only write deadlock-free code
- If not, we provide a way to detect deadlock very fast
- User can write code to recover from deadlock

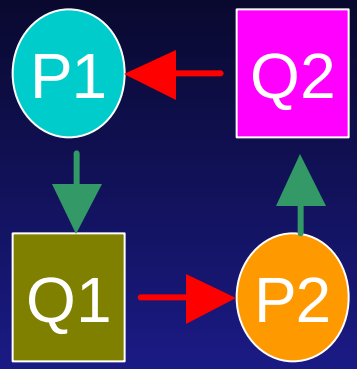
Deadlock Detection Unit (DDU)

- Small & scalable parallel hardware unit
- Multiple requestors & resources
  - In this paper, the only requestors are processors and the only resources are specialized hardware units like FFT

Overall Outline

- Motivation - Technology Trends
- Background - Deadlock Detection
- Parallel Algorithm
- Parallel Architecture
- Experimental Results
- Conclusion

Background: Deadlock Condition



- Properties of Resources
  - Mutual Exclusion: Any resource can be held exclusively, making it unavailable to other processors
  - Non-preemption: Any resources can be released only by the processors holding the resource.
- Behavior of processors
  - Partial Allocation: a processor may hold some resources while the processor requests additional resources.
  - Blocked Wait: processor must wait for unavailable resources to become available.

Previous Algorithms' Run Time

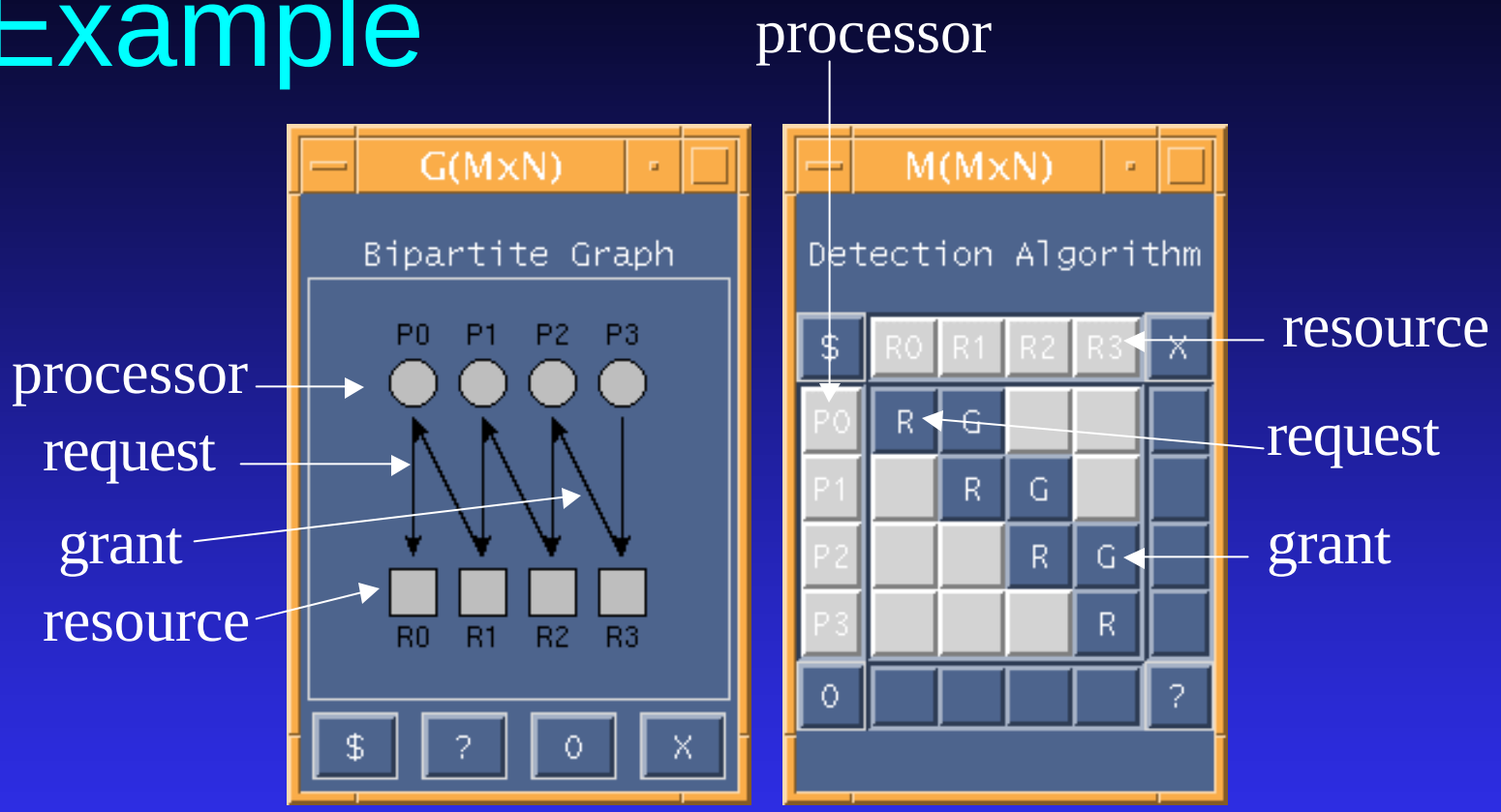
Generally the run time is O(m*n), where m is the number of processors and n is the number of resources.

- Path Based, O(e), or O(e≤m*n), where e is the set of edges.
- Tree Based, O(m*n)
- Matrix Based, O(m*n)
- Message Passing Based, O(m*n)

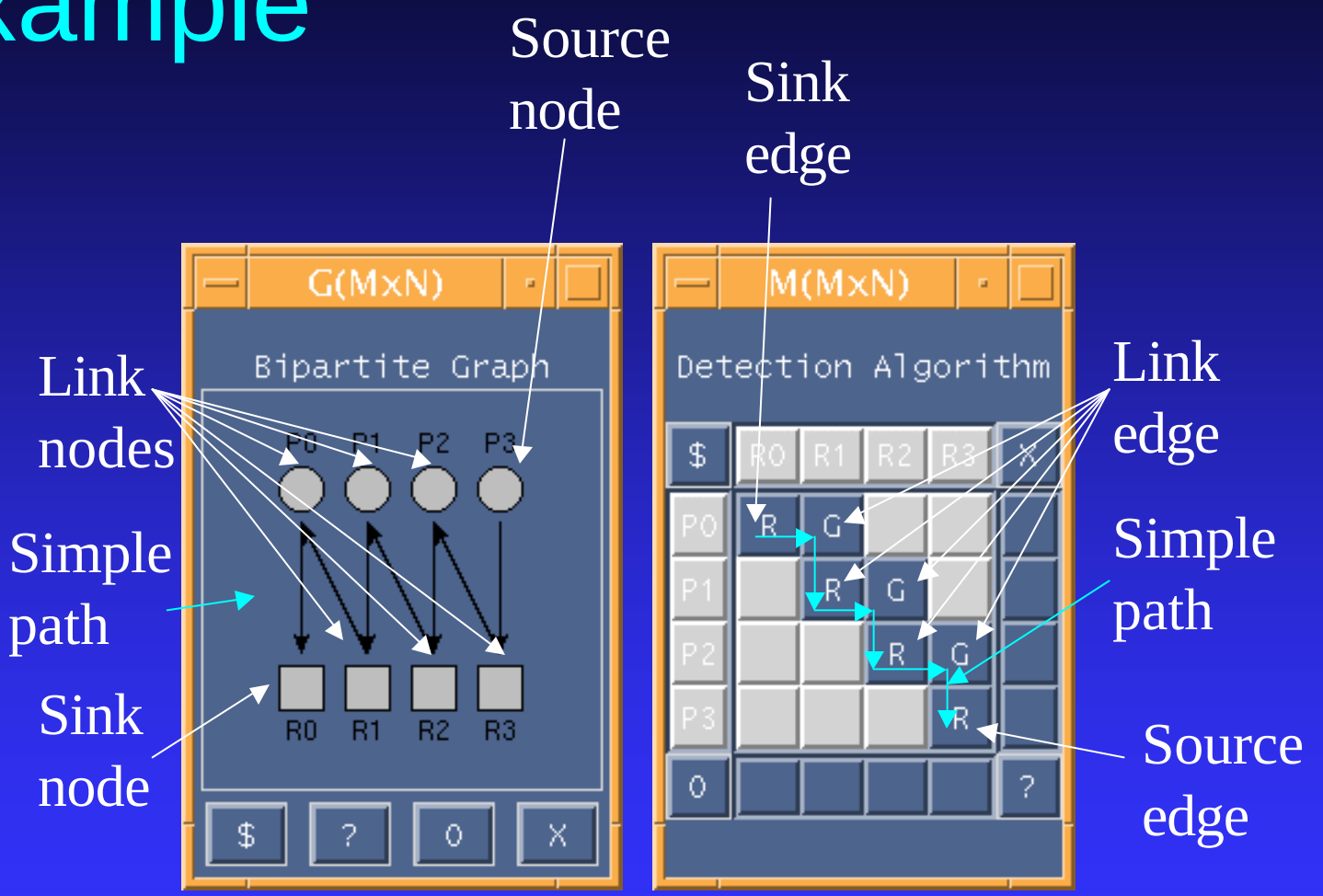
Overall Outline

- Motivation - Technology Trends
- Background - Deadlock Detection
- Parallel Algorithm
- Parallel Architecture
- Experimental Results
- Conclusion

Example


processor
G(MxN)
Bipartite Graph
P0 P1 P2 P3
R0 R1 R2 R3
$? 0 X
M(MxN)
Detection Algorithm
$ R0 R1 R2 R3 X
P0 R G
P1 R G
P2 R G
P3 R
0 ?
processor
request
grant
resource
resource
request
grant

Example


Source node
Sink edge
Link nodes
Simple path
Sink node
Link edge
Simple path
Source edge
G(MxN)
Bipartite Graph
P0 P1 P2 P3
R0 R1 R2 R3
$? O X
M(MxN)
Detection Algorithm
$ R0 R1 R2 R3 X
P0 P1 P2 P3
O ?
R G

Matrix Representation

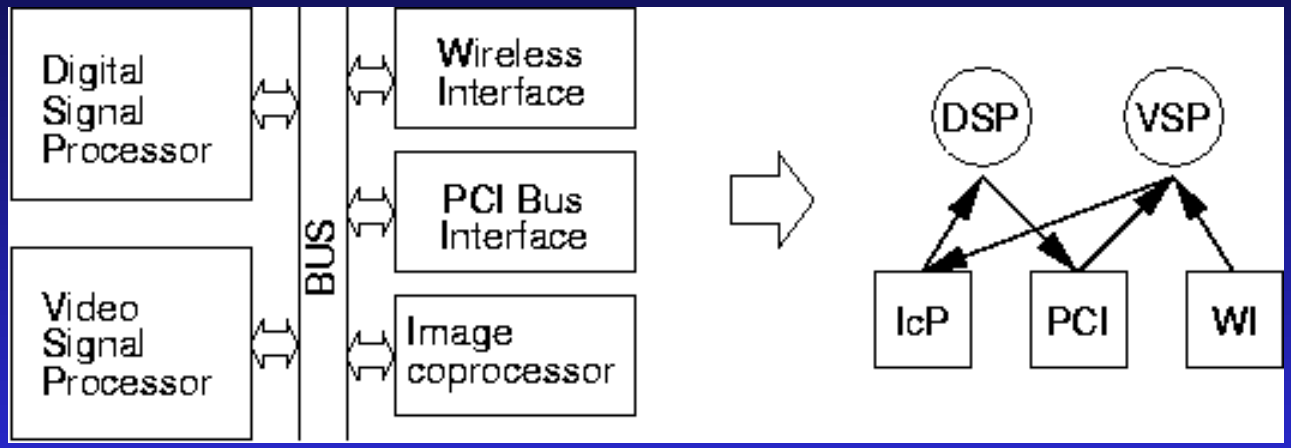
- Each row corresponds to a requestor (processor)
  - p_i represents requestor (processor) i
- Each column corresponds to a resource
  - q_j represents resource j
- Entries in the matrix
  - r (r_{ij}) represents a request
  - g (g_{ij}) represents a grant
  - 0 represents no action (neither request nor grant)

Georgia Tech

Properties

- Proposed Algorithm
  - Matrix Based
  - Modified Reduction Technique
  - Handling multiple requests, and grants at the same time.
  - Requires simple bit-wise boolean operations.

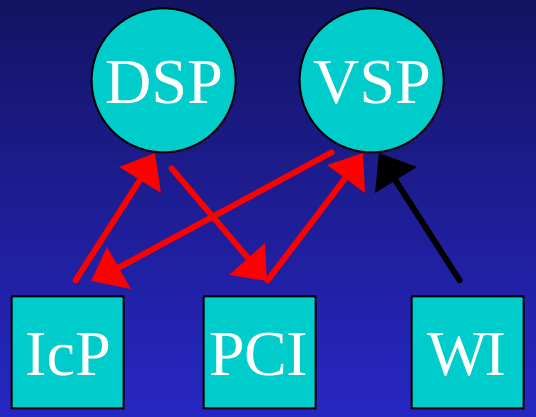
SoC Example



P\Q	q1(IcP)	q2(PCI)	q3(WI)
p1(DSP)	g	r	0
p2(VSP)	r	g	g

Deadlock and Cycle Relation

- Deadlock ⇒ ∃ cycles
- Cycles ⇒ ∃ Deadlock
  (As shown in the red)



P\Q	q1(IcP)	q2(PCI)	q3(WI)
p1(DSP)	g →	r ↓	0
p2(VSP)	r ←	g	g

Matrix Representation

$$M = \begin{bmatrix} g & r & 0 \\ r & g & g \end{bmatrix}$$

$$g_c = [01] \quad r_c = [10]$$

$$M_c = \begin{bmatrix} 01 & 10 & 00 \\ 10 & 01 & 01 \end{bmatrix}$$

$$g_r = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad r_r = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$M_r = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Matrix Representation: calculation of M_{rbo} and XOR_{right}

$$M_r = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad M_{rbo} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad XOR_{right} = \begin{bmatrix} 1 \oplus 1 \\ 1 \oplus 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Matrix Representation: calculation of M_{cbo} and XOR_{below}

$$M_c = \begin{bmatrix} 01 & 10 & 00 \\ 10 & 01 & 01 \end{bmatrix}$$

$$M_{cbo} = [11 \quad 11 \quad 01]$$

$$XOR_{below} = [1 \oplus 1 \quad 1 \oplus 1 \quad 0 \oplus 1] = [0 \quad 0 \quad 1]$$

Georgia Tech

Result of first iteration

$$XOR_{\text{below}} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

$$XOR_{\text{right}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Based on result, we set all entries in column 3 to zero:

$$M = \begin{bmatrix} g & r & 0 \\ r & g & 0 \end{bmatrix}$$

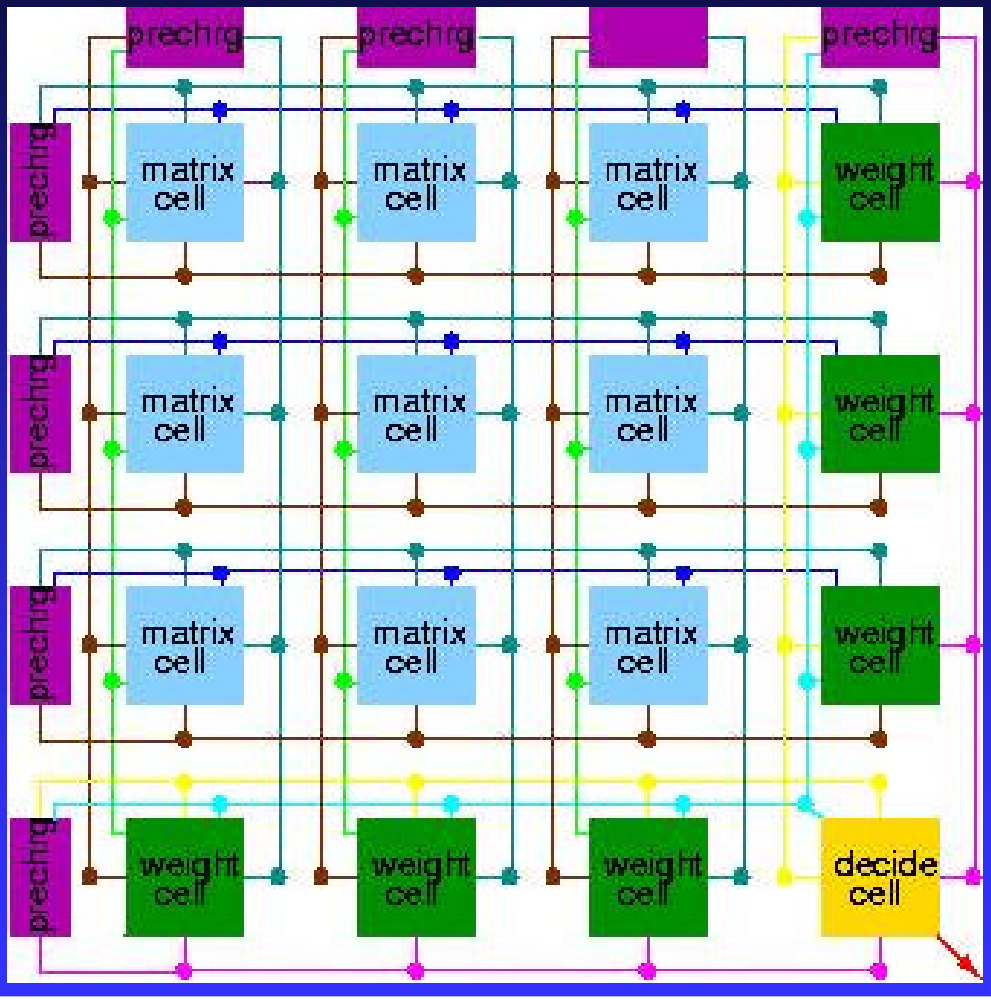
Multiple Iterations

- Continuing in this way, we continue iterating until no more changes
- When finished, if M is all zeros, we have no deadlock; otherwise, we do have deadlock
- This algorithm requires at most 2*min(m,n) iterations

Overall Outline

- Motivation - Technology Trends
- Background - Deadlock Detection
- Parallel Algorithm
- Parallel Architecture
- Experimental Results
- Conclusion

3 Processors/3 Resources: Architecture



Overall Outline

- Motivation - Technology Trends
- Background - Deadlock Detection
- Parallel Algorithm
- Parallel Architecture
- Experimental Results
- Conclusion

Experiments

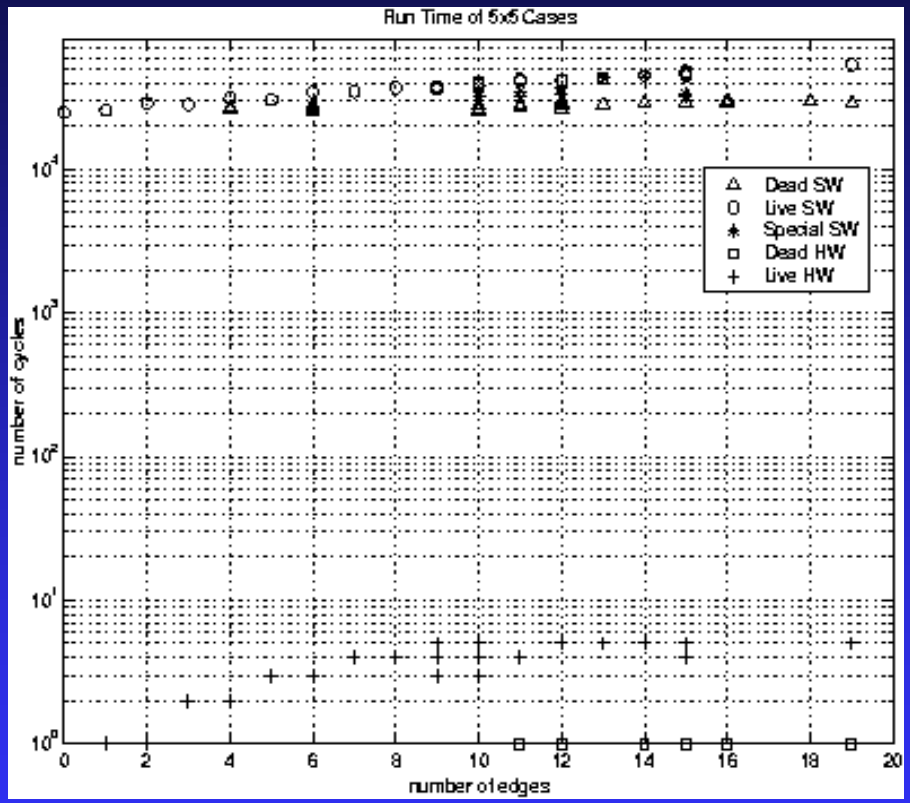
- Assumption
  - Software Cycle: 83.3 MHz processor
  - Hardware Cycle:
    - Synthesized from gate-level description
    - Clock as fast as critical path (e.g., 4.12 ns ⇒ 242 MHz Clock)
    - Clock same as CPU clock 83.3 MHz clock (12 ns cycle time)
- Simulation
  - Previous Algorithm: PowerPC 750 runs .c in Seamless CVE
  - Proposed Algorithm: Synopsys VCS runs .v
- ~100 – 1000 times faster
  - 99% run time reduction

Area and Delays of DDU

|P| Times |Q|	Lines of Verilog	Area AMI 0.3u	Delay/ Step (ns)	Worst Case (# steps)	Worst Case Custom Clk (ns)	Worst Case 83.3Mhz (ns)
2x3	49	186	0.91	2	1.82	24ns
5x5	73	264	2.21	5	11.05	60ns
7x7	102	455	2.51	7	17.57	84ns
10x10	162	622	3.66	10	36.6	120ns
50x50	2682	14142	4.12	50	206	600ns

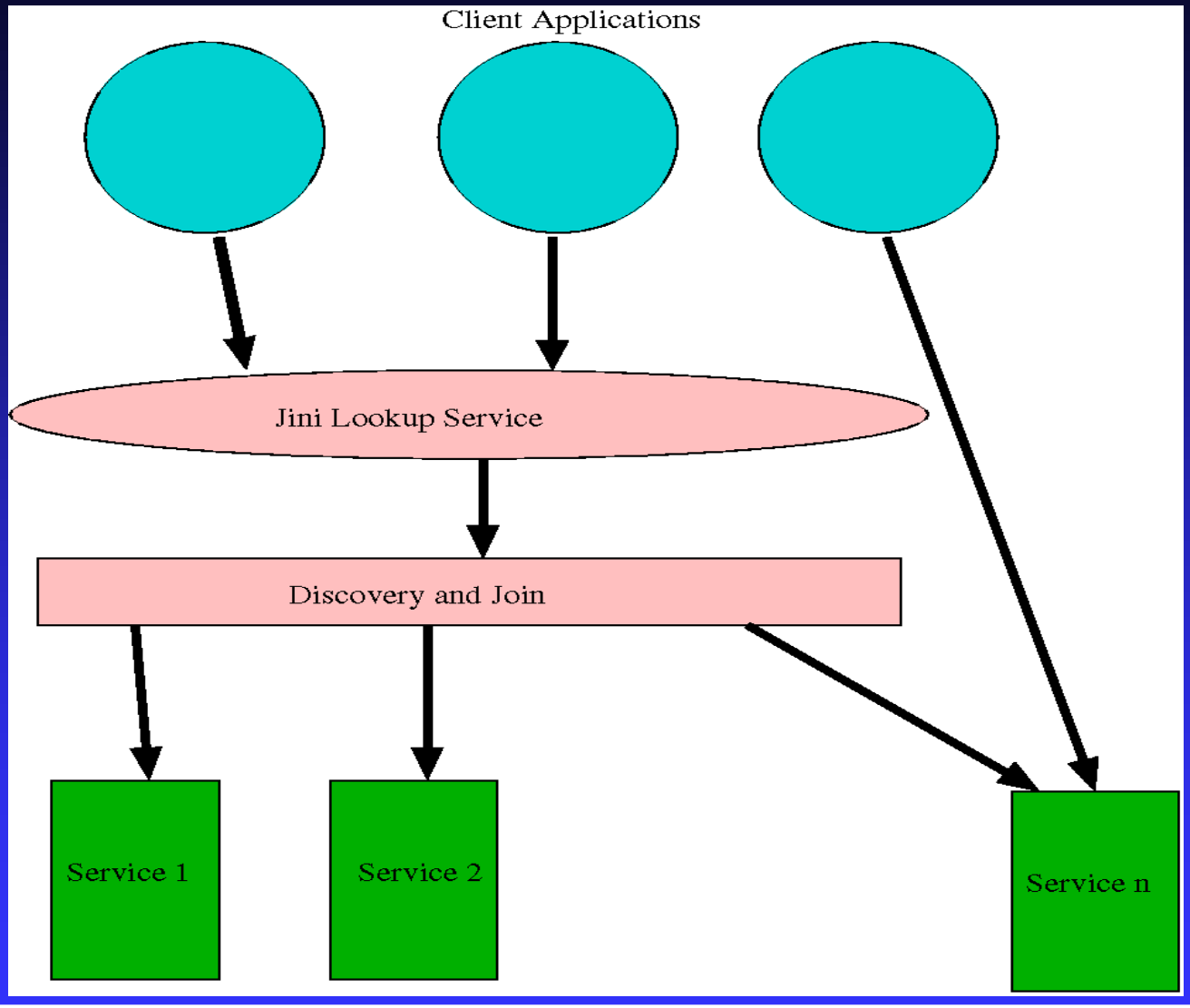
Hardware vs. Software Performance

Number of Cycles

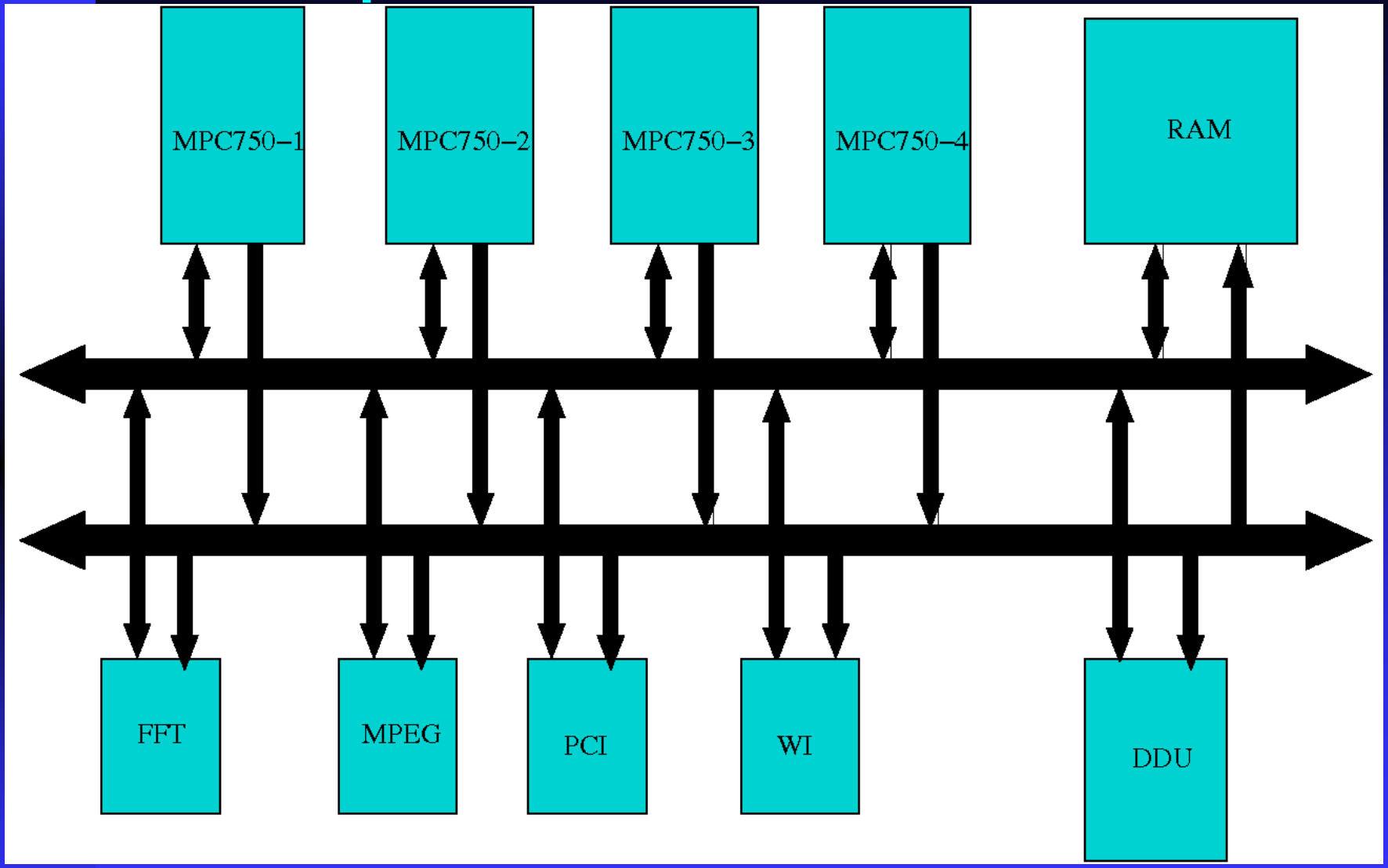


Number of Edges

Example: Lookup Service



Example SoC Architecture



Event Sequence of the Example

Time	Event No.	Events
t1	e1	MPC750-1 requests FFT, MPEG are granted to MPC750-1 immediately
t2	e2	MPC750-3 requests FFT, PCI; PCI is granted to MPC750-3 immediately.
t3	e3	MPC750-2 requests FFT, MPEG.
t4	e4	FFT is released by MPC750-1
t5	e5	FFT is granted to MPC750-2.

Adjacency Matrices

	MPC750-1	MPC750-2	MPC750-3	MPC750-4
FFT	g	0	0	0
MPEG	g	0	0	0
PCI	0	0	0	0
WI	0	0	0	0

t_1

	MPC750-1	MPC750-2	MPC750-3	MPC750-4
FFT	g	0	r	0
MPEG	g	0	0	0
PCI	0	0	g	0
WI	0	0	0	0

t_2

	MPC750-1	MPC750-2	MPC750-3	MPC750-4
FFT	g	r	r	0
MPEG	g	0	0	0
PCI	0	r	g	0
WI	0	0	0	0

t_3

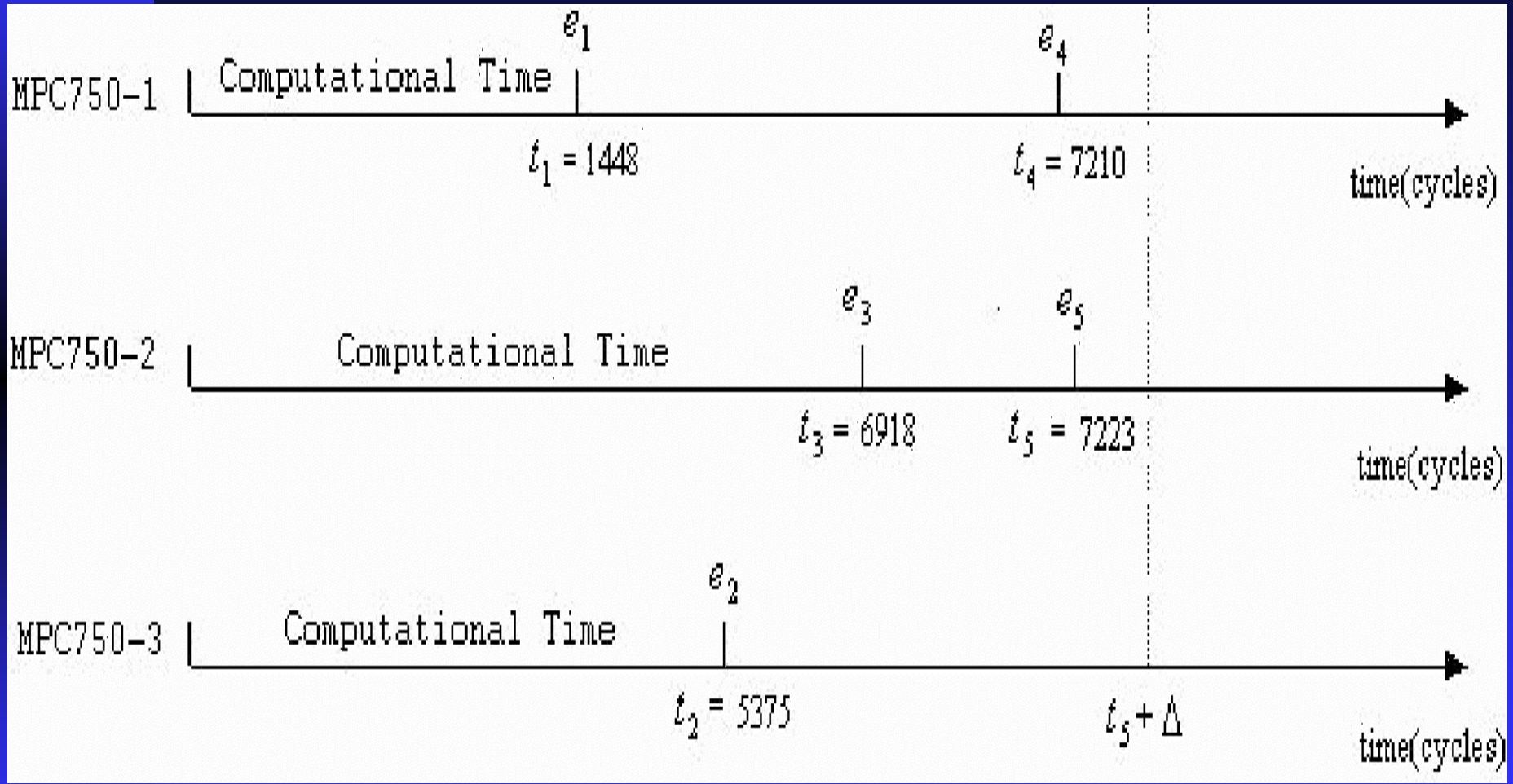
	MPC750-1	MPC750-2	MPC750-3	MPC750-4
FFT	0	r	r	0
MPEG	g	0	0	0
PCI	0	r	g	0
WI	0	0	0	0

t_4

	MPC750-1	MPC750-2	MPC750-3	MPC750-4
FFT	0	g	r	0
MPEG	g	0	0	0
PCI	0	r	g	0
WI	0	0	0	0

t_5

Sequence of Events



Deadlock Detection Time and Total Execution Time

Method of Deadlock Detection	Detection Time Δ (cycles)	$t_5 + \Delta$
Software	16,038	23,261
DDU	2	7,225

$$S_{overall} = \frac{23{,}261 - 7{,}225}{23{,}261} = 68.9\%$$

Conclusion

- Deadlock Detection Unit
  - very small area, even for 50x50
  - $O_{sw}(m*n)$ to $O_{hw}(min(m,n))$ speedup
  - Linearly scalability in min(m,n)
  - Handle simultaneous requests/grants
- DDU can be used by multiprocessor SoC sofware code to detect deadlock quickly and then, for example, release resources to get out of deadlock

Future Work

- Integrate DDU into an RTOS
  - Monitor DDU output
    - DDU API
  - Extend to handle multiple “blocked wait” threads on one CPU: RTOS on each processor aggregates requests which have the blocked wait property ⇒ each aggregate group is represented by a unique “processor” row
  - Try different recovery schemes
  - Perhaps some hardware assist in recovery