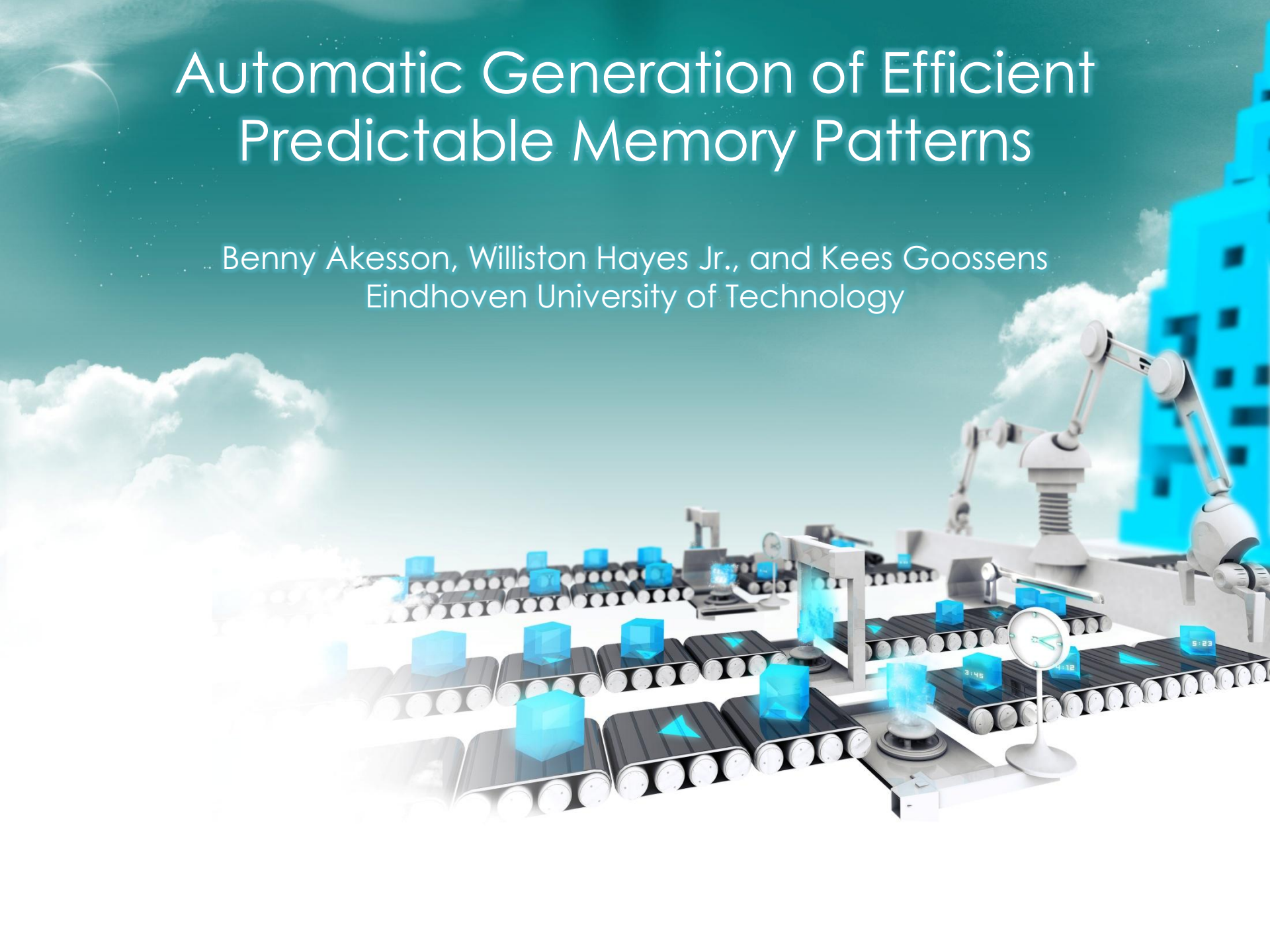
Automatic Generation of Efficient Predictable Memory Patterns

Benny Akesson, Williston Hayes Jr., and Kees Goossens
Eindhoven University of Technology



Trends in Embedded Systems

→MPSoC design gets **increasingly complex**

- Number of applications in a device is increasing
- More processors, hardware accelerators, and memories
- Many applications execute concurrently

- Some applications have **(hard) real-time requirements**
  - Missing a deadline results in significant quality degradation

- **Resources are shared** between applications to reduce cost
  - Results in temporal interference between sharing applications
  - Makes it difficult to satisfy real-time requirements

Formal Verification

- Formal RT verification requires **predictable systems**
  - Have performance models of both applications and hardware

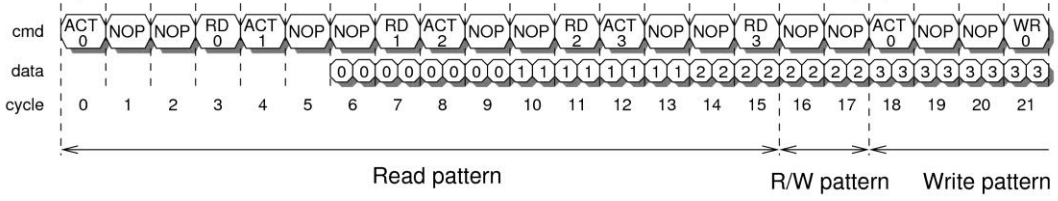
- We have proposed a predictable platform **(CoMPSoC)**
  - Processor tile with MicroBlaze processor and RTOS
  - Æthereal network-on-chip
  - Memory tiles with SRAM controller or **Predator SDRAM controller**

Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Problem Statement

- SDRAM bandwidth is **scarce** and must be **efficiently** utilized
  - Off-chip pins are expensive in terms of area and power
- Our SDRAM controller is based on **predictable memory patterns**
  - Statically computed sequences of SDRAM commands
  - Dynamically scheduled at run-time
  - Enable **bandwidth** and **response times** of requests to be bounded
- Memory patterns are **computed manually**
  - Time-consuming and error-prone process
  - Five patterns required per memory device / configuration
  - Manually computed patterns may not use bandwidth efficiently



Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Contributions

→This paper presents **three algorithms** for pattern generation
- Branch and bound scheduling
- As-soon-as possible scheduling
- Bank scheduling

→Algorithms are **experimentally evaluated**
- For a range of memories and configurations
- Run-time of algorithm vs. efficiency (bandwidth)

Presentation Outline

Introduction

SDRAM overview

Predictable SDRAM controller

Generation algorithms

Experiments

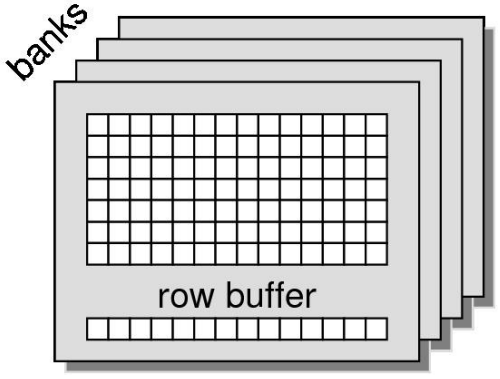
Conclusions

Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

SDRAM Architecture

- An SDRAM is organized in **banks**, **rows** and **columns**
  - A **row buffer** in each bank stores a currently active (open) row
- Interface has a **command bus**, **address bus**, and a **data bus**
  - Buses shared between banks to reduce the number of off-chip pins
- SDRAM cells suffer from leakage
  - Needs to be **refreshed** regularly to retain data

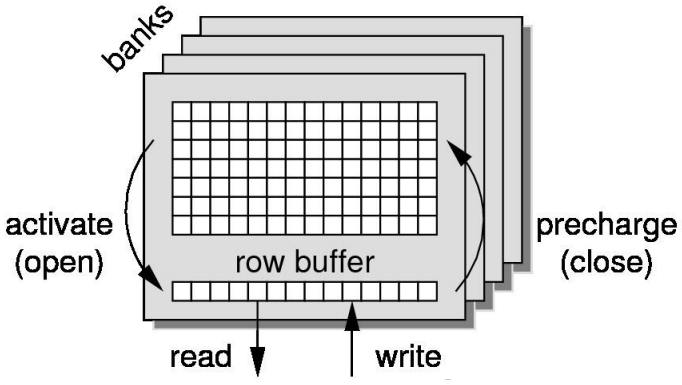


Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Basic SDRAM Operation

- Memory map decodes address to bank, row, and column
- Row is **activated** and copied into the row buffer of the bank
- Read bursts and/or write **bursts are issued to the active row**
  - Programmed **burst length** (BL) of 4 or 8 words
- Row is **precharged** and stored back into the memory array



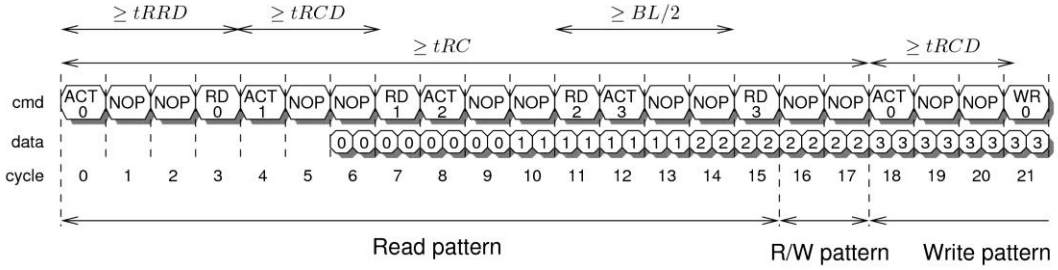
Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Timing Constraints

→**Timing constraints** determine schedulability of commands
- More than 20 constraints on minimum time between commands
  - E.g. activate-to-activate, activate-to-read/write, read/write-to-precharge, read-to-write, write-to-read, etc.
- Constraints reduce bandwidth provided by the memory

→**Memory efficiency**
- The fraction of clock cycles when requested data is transferred
- Determines the guaranteed **net bandwidth**



Presentation Outline

Introduction

SDRAM overview

Predictable SDRAM controller

Generation algorithms

Experiments

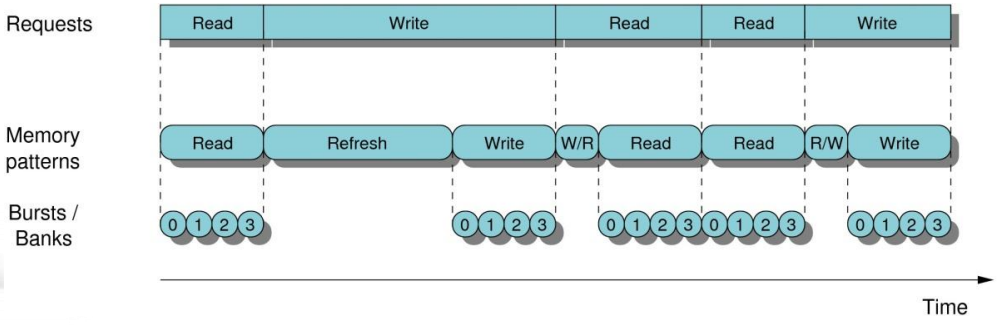
Conclusions

Predictable SDRAM

→Predictability through predictable **memory patterns**
- Statically computed sequences of SDRAM commands
- Dynamically scheduled at run-time

→There are **five types** of memory patterns
- Read, write, r/w switch, w/r switch, and refresh patterns

→Request to pattern mapping:
- Read request → read pattern (potentially first w/r switch)
- Write request → write pattern (potentially first r/w switch)
- Refresh pattern issued periodically to retain data



Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Memory Patterns

→Patterns enable scheduling at higher level than commands
- **Less state** and **fewer constraints**, making them **easier to analyze**

→Bounding memory efficiency (bandwidth)
- Worst sequence of patterns is known (scheduling rules & pattern lengths)
- Data transferred by patterns is known (by definition)

→Bounding response times
- Number of interfering requests is known (arbiter analysis)
- Request to pattern mapping is known (scheduling rules)
- Pattern to cycle mapping is known (pattern lengths)

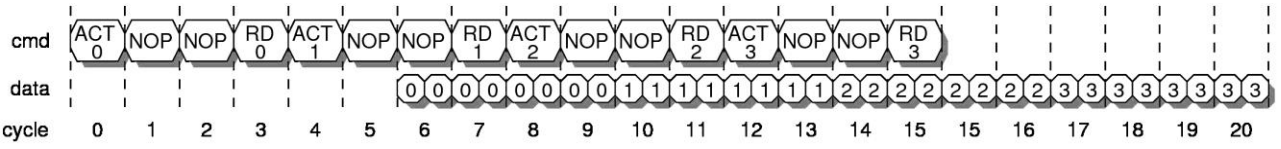
Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Pattern structure

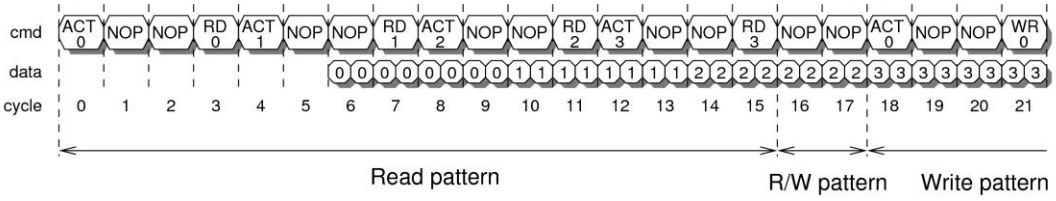
- There is a general structure for memory patterns
  - **Valid patterns** implement this structure and satisfies all timing constraints of the memory device

- Structure of **access patterns** (read and write patterns)
  - At least one activate and precharge command per bank
    - Access patterns must be independent
    - Incorrect rows are open in banks in worst case
    - Banks are precharged immediately after access (close-page policy)
    - Improves worst-case memory efficiency
  - Fixed number of bursts to each bank, called **burst count** (BC)
    - Memory efficiency increases with burst count



Structure of auxiliary patterns

- Switching patterns
  - Purpose is to allow data bus to switch direction
  - Consists of zero or more NOP commands
- Refresh patterns
  - First consists of NOP command to allow all banks to precharge
  - Then has a refresh command follow by NOPs to finish refresh
- Auxiliary patterns are easy to derive given access patterns
  - Shown in paper, not discussed further in this presentation



Benny Åkesson
k.b.akesson@tue.nl

Technische Universiteit Eindhoven
University of Technology

Presentation Outline

Introduction

SDRAM overview

Predictable SDRAM controller

Generation algorithms

Experiments

Conclusions

Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Design decisions

→ Huge design space reduced using **five design decisions**

1. Shorter access patterns are assumed to be more efficient
   - Enables finding shortest read and write patterns independently
   - Auxiliary patterns are generated afterwards
   - Assumption usually valid, but may reduce efficiency with up to 1%

2. Identities of banks are not distinguished
   - Patterns identical if all commands to two bank are swapped
   - Reduces set of valid patterns considerably
   - No impact on efficiency or response time

Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Design decisions

3. Access patterns start with an activate command
   - Rationale: must activate before reading or writing
   - Ignores patterns starting with one or more NOP commands
   - Initial NOPs typically reduce bandwidth
   - No impact on efficiency or response time

4. Issue last burst to a bank with auto-precharge flag
   - Less commands to schedule, limiting the design space
   - Less contention on command bus, which may improve efficiency

5. Issue all bursts to a bank before moving to next
   - Gives more time to activate and precharge between accesses
   - Improves efficiency

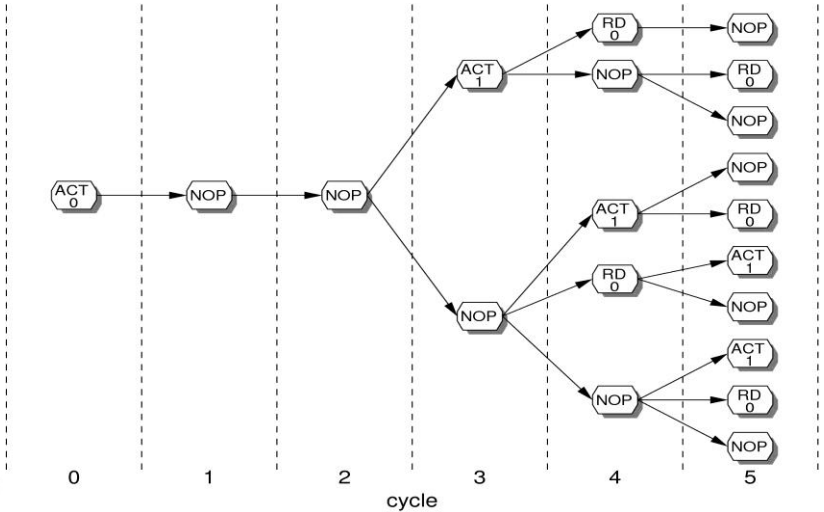
Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Branch and bound scheduling

- Algorithm is based on depth-first traversal of valid patterns
  - Guaranteed to find shortest patterns
  - Optimal given our design decisions

- Run-time of algorithm is a problem due to large search space
  - 10000 optimal read patterns of 32 cycles for DDR2-400 BC=2
  - Three orders of magnitude more patterns with length 37!



Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Pruning the search space

→Search space is pruned to reduce run-time

→Two bounding conditions determine if branch can be discarded

1. If pattern is longer than current shortest pattern
2. If pattern is will be longer after scheduling remaining commands
   - Determined based on timing constraints between successive activate commands and read/write commands

→Neither of these conditions can discard an optimal solution

→Run-time may be **hours or days** despite pruning

– Faster algorithm required for faster memories or high burst counts

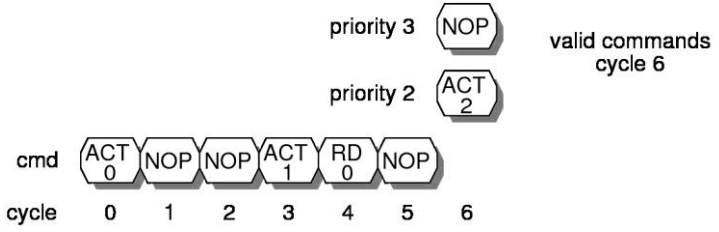
Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

ASAP scheduling

- ASAP scheduling is a **heuristic** that aims to reduce run-time
  - Simple intuitive algorithm
  - Schedule commands as early as possible to find short schedules

- Algorithm works **cycle-by-cycle**
  - Determine set of valid commands
  - Use simple priority mechanism to schedule command
    1. Read/write command (puts data on bus)
    2. Activate command (enables future data transfer)
    3. NOP



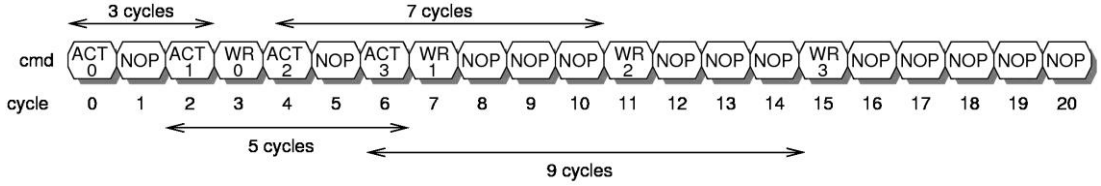
Benny Åkesson
k.b.akesson@tue.nl

Technische Universiteit Eindhoven University of Technology

Problem with ASAP scheduling

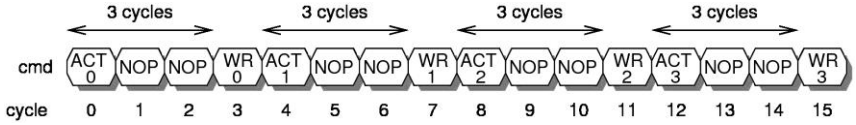
→It executes in a second, but patterns are **not always efficient**
- Activates scheduled increasingly far from their read/writes
- Additional NOPs required to satisfy precharge conditions
- Reduces memory efficiency up to 10% compared to B&B

→This motivates looking for a better heuristic



ASAP read pattern for DDR2-400

Balanced read pattern for DDR2-400



Benny Åkesson
k.b.akesson@tue.nl

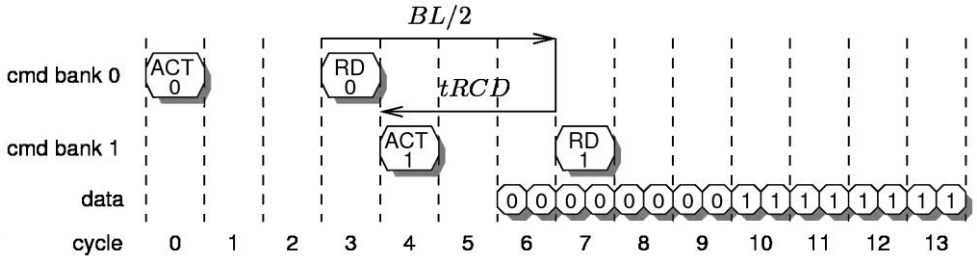
Technische Universiteit Eindhoven University of Technology

Bank scheduling

→Bank scheduling is a **heuristic** that aims for high efficiency
- Builds on lessons from ASAP algorithm
- Aims to keep activates close to their read/write commands

→Algorithm works **bank-by-bank**
- Schedules first bank according to minimum timing constraints
- Tries scheduling read/write at BL/2 cycles from last access
  - Successful if its activate can be scheduled tRCD cycles earlier
  - Otherwise move read/write one cycle later and try again

→It executes in a second and provides high efficiency!



Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Presentation Outline

Introduction

SDRAM overview

Predictable SDRAM controller

Generation algorithms

Experiments

Conclusions

Benny Åkesson
k.b.akesson@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology

Experimental Setup

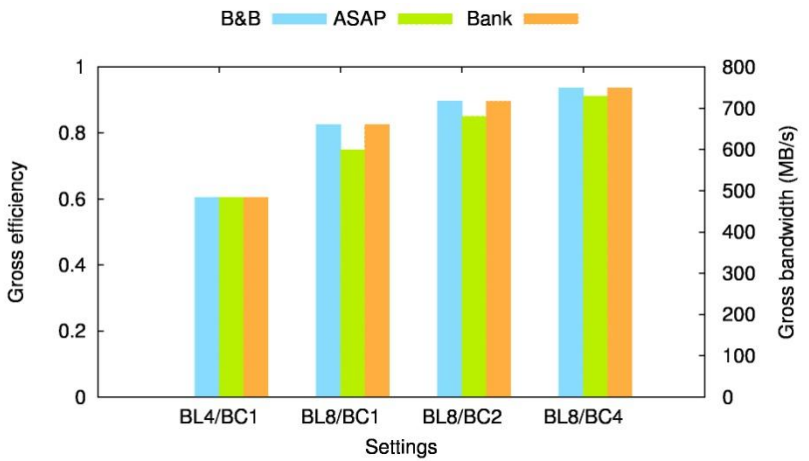
→Experiments consider a range of memories and configurations
- DDR2-400 (DDR2-800 and DDR-1600 in paper)
- 16 bit interface, 4 banks, 512 Mb capacity
- Burst count (BC) 1, 2, and 4
- Programmed burst length (BL) of 4 and 8 words

→Experiment considers **worst-case** memory efficiency
- No simulation, exercises tooling
- Independent of input

DDR2-400

- Worst-case efficiency results
  - All patterns are identical with BL=4
    - Timing constraints give few options with small bursts
  - Efficiency of ASAP is up to 10.2% lower than for B&B
    - Longer write pattern due to precharge problem
  - Bank scheduling provides same efficiency as B&B for all settings

- Run-time results
  - typically in a second for all algorithms
  - B&B requires 8 days with BC=4
  - B&B does not finish in 10 days for DDR3-1600 BC=2,4



Benny Åkesson
k.b.akesson@tue.nl

Presentation Outline

Introduction

SDRAM overview

Predictable SDRAM controller

Generation algorithms

Experiments

Conclusions

Conclusions

→A predictable memory controller has been proposed
- Enables formal verification of SoCs with large storage requirements
- Based on memory patterns, which must be generated manually

→The paper presents **three pattern generation algorithms**

→We show that the choice of algorithm matters
- Difference between B&B and ASAP scheduling is up to 10.2%
- B&B is efficient, but is slow for faster memories with more banks
- Bank scheduling is fast and provides same efficiency as B&B

→Bank scheduling provides a favorable trade-off between run-time and efficiency

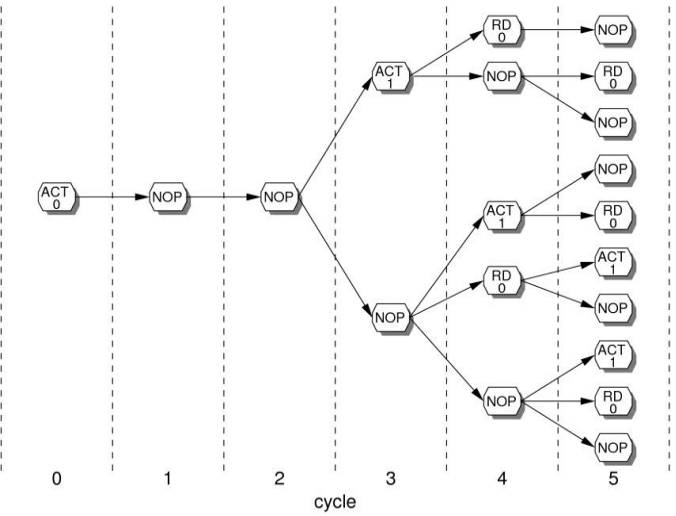
Benny Åkesson
k.b.akesson@tue.nl

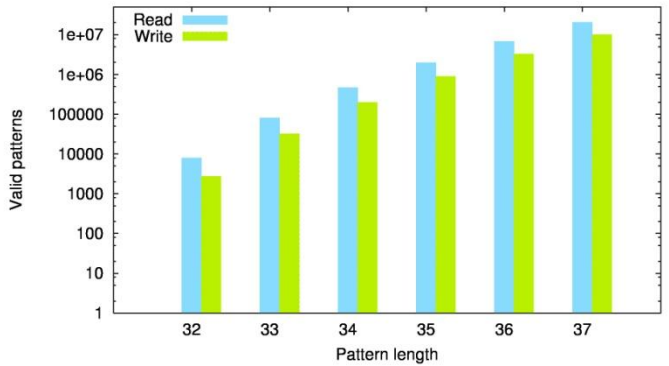
TU/e Technische Universiteit Eindhoven University of Technology

Branch and bound scheduling

- Algorithm is based on depth-first traversal of valid patterns
  - Guaranteed to find shortest patterns
  - Optimal given our design decisions

- Run-time of algorithm is a problem due to large search space
  - 10000 optimal read patterns of 32 cycles for DDR2-400 BC=2
  - Three orders of magnitude more patterns with length 37!

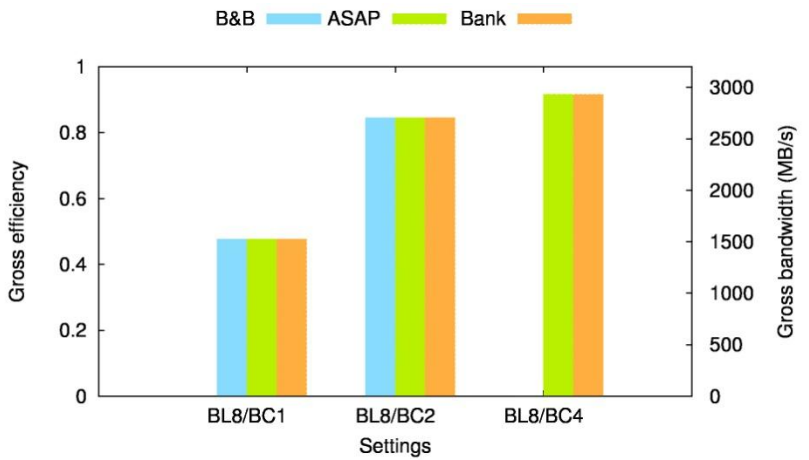




DDR3-1600

- Worst-case efficiency results
  - All algorithms perform identically for all settings
  - Write patterns are not longer with ASAP scheduling
    - Memory has eight banks
    - Four-activate window spreads out activates better

- Run-time results
  - ASAP and bank scheduling takes a second
  - B&B with BC=1 took 7 days to generate
  - B&B with BC=2 and 4 did not finish in 10 days!

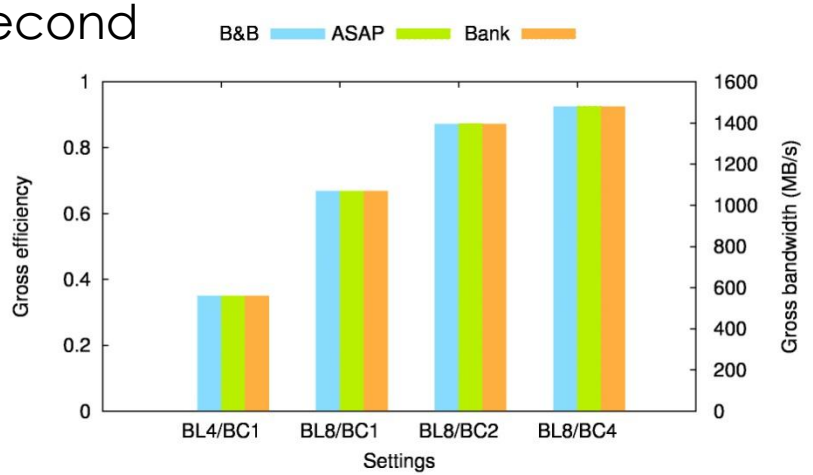


Benny Åkesson
k.b.akesson@tue.nl

DDR2-800

- Worst-case efficiency results
  - All algorithms perform almost identically for all settings
  - ASAP scheduling provides 0.1% high efficiency than others for BC=2
    - Write patterns three cycles longer than for other algorithms
    - Longer write pattern reduces lengths of auxiliary patterns
    - Benefit is negligible
    - Shows drawback of first design decision – shorter is not always better

- Run-time results
  - ASAP and bank scheduling takes a second
  - B&B with BC=4 took 32 minutes



Benny Åkesson
k.b.akesson@tue.nl