Analysis of Memory-Contention in Heterogeneous COTS MPSoCs

Mohamed Hassan and Rodolfo Pellizzoni

https://gitlab.com/FanusLab/memory-contention-analysis
Outline

1. Motivation
2. Big Picture
3. Motivation
4. Proposed
5. Results
DRAM Consists of multiple banks
The memory controller (MC) manages accesses to DRAM
A request in general consists of:
  • ACTIVATE (A) command:
    • Bring data row from cells into sense amplifiers
  • Read/Write (R/W) commands:
    • To read/write from specific columns in the sense amplifiers
  • PRECHARGE (P) command:
    • to write back a previous row in the sense amplifiers before bringing the new one
Request Types

Intra-bank interfering requests:

1. Intra-bank conflict requests
Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests
Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Request Types
Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests
Latency Buckets (Components)

Big Picture

ACT Latency  CAS Latency  Conflict Latency
Latency Buckets (Components)

Big Picture

Conflict Latency

CAS Latency

ACT Latency

Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests
Request-Dr Analysis

- What is the worst-case of each of these components can be suffered by a single request? \( \rightarrow WCL_{\text{per-req}} \)
- Assuming nothing at all about interfering tasks
  - (i.e., infinite number of interfering requests)
- Then obtain total memory latency assuming we know the total number of interfered requests
  \( \rightarrow WCL_{\text{tot}} = \#\text{Reqs} \times WCL_{\text{per-req}} \)
Request-Driven Analysis

- What is the worst-case of each of these components can be suffered by a single request? → $WCL_{\text{per-req}}$
- Assuming nothing at all about interfering tasks
  - (i.e., infinite number of interfering requests)
- Then obtain total memory latency assuming we know the total number of interfered requests
  → $WCL_{\text{tot}} = \#\text{Reqs} \times WCL_{\text{per-req}}$

Job-Driven Analysis

- What is the worst-case of each of these components can be suffered by the total task assuming we know the number of interfering requests?
- Assuming nothing at all about # interfered requests
  - (i.e., infinite number of interfered requests)
Request-Driven vs Job-Driven Analysis

Motivation
Request-Driven vs Job-Driven Analysis

Motivation

Relatively small number of interfered requests → Req-Dr wins
Relative small number of interfered requests → Req-Dr wins

Relative small number of interfering requests → Job-Dr wins

Request-Driven vs Job-Driven Analysis
  - Both request- and job-driven analysis
  - A specific COTS platform

  - Both request- and job-driven analysis
  - A specific COTS platform

• [Hassan Req-Dr] Mohamed Hassan and Rodolfo Pellizzoni. Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems, EMSOFT, 2018
  - Explores a wide variety of COTS possible configurations (144 platform instances)
  - Only request-driven analysis
What do we do?

- A task-aware
- COTS-aware
- Hybrid analysis

This work

Proposed

- Inter-bank open requests
- Inter-bank close requests
- Intra-bank reorder requests
- Conflict Latency

- Intra-bank conflict requests
- Conflict Latency

- Write batching requests
- Inter-bank close requests
- ACT Latency
What do we do?

• A task-aware COTS-aware Hybrid analysis

• **Task-aware:**
  • Account for different levels of knowledge we have about running tasks:
    • Total number of requests
    • Total number of reads + writes
    • Total number of open (row hits) + close (row misses) requests
COTS-Aware Analysis

Memory Behavior Depends on?:

- **OS Configuration**
  - Priority:
    - PEs can be given priorities
    - COTS platforms support different priority levels
    - Existing analysis does not account for this
  - Intra-bank scheduling
    - FR-FCFS
    - COTS also supports a threshold on reordering to prevent starvation
  - Inter-bank scheduling
    - RR across banks
    - Two flavors:
      - Always schedule ready commands of any type (high performance)
      - Reorder only commands of different type (prevent starvation)
  - Read/Write arbitration, two flavors:
    - Reads and writes have same priority
    - Serve in batches, where reads have higher priority

- **PE Architecture**
- **MC Policies**
R/W Reorder
- 1: write batching
- 0: no write batching

Inter-bank Reorder
- 1: Reorder across all commands
- 0: Reorder commands of diff types

FR-FCFS Threshold
- 1: FR-FCFS is capped
- 0: no cap on FR-FCFS

Priority
- 1: Critical PEs are higher priority
- 0: no priority

Pipeline
- IO-All: All PEs are In-order
- IO-Cr: Critical PEs are in-order
- OOO-All: All PEs are OOO

Partitioning
- No-Part: No Partitioning
- Part-Cr: Partition among critical apps
- Part-All: Partition among all apps

COTS-Aware Analysis
Proposed
COTS-Aware Analysis

### MPSoC Platform Instances

1. **R/W Reorder**
   - 1: write batching
   - 0: no write batching

2. **FR-FCFS Threshold**
   - 1: FR-FCFS is capped
   - 0: no cap on FR-FCFS

3. **Priority**
   - 1: Critical PEs are higher priority
   - 0: no priority

4. **Inter-bank Reorder**
   - 1: Reorder across all commands
   - 0: Reorder commands of diff types

5. **Pipeline**
   - **IO-All**: All PEs are In-order
   - **IO-Cr**: Critical PEs are in-order
   - **OOO-All**: All PEs are OOO

6. **Partitioning**
   - **No-Part**: No Partitioning
   - **Part-Cr**: Partition among critical apps
   - **Part-All**: Partition among all apps

144 different platform instances!
What do we do?

- A task-aware COTS-aware Hybrid analysis
- **Hybrid:**
  - **State-of-the-art:** only running request- or job-Dr analysis or run both and take the min
  - **This work:** construct an optimization framework that blends both request-level and task-level per-core constraints to obtain tighter bounds
Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests

Write batching requests
Inter-bank open requests
Inter-bank close requests
Intra-bank reorder requests
Intra-bank conflict requests

ACT Latency
CAS Latency
Conflict Latency
Conflict Req

Optimization problem:
1. Write Latency components as functions on those requests

Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests

ACT Latency
CAS Latency
Conflict Latency
Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests

Optimization problem:
1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis

ACT Latency
CAS Latency
Conflict Latency

Write batching requests
Inter-bank open requests
Inter-bank close requests
Intra-bank reorder requests
Intra-bank conflict requests
Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests

Optimization problem:
1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)
<table>
<thead>
<tr>
<th></th>
<th>Priority</th>
<th>Interfering Reqs from Cr Core</th>
<th>Interfering Reqs from nCr Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part-All</td>
<td>No thr</td>
<td>FR-FCFS thr</td>
<td>None</td>
</tr>
<tr>
<td>No-Priority</td>
<td>No thr</td>
<td>FR-FCFS thr</td>
<td>None</td>
</tr>
<tr>
<td>Part-Cr</td>
<td>No thr</td>
<td>FR-FCFS thr</td>
<td>* unbounded</td>
</tr>
<tr>
<td>No-Priority</td>
<td>No thr</td>
<td>FR-FCFS thr</td>
<td>* Unbounded</td>
</tr>
<tr>
<td>No-Part</td>
<td>Priority</td>
<td>FR-FCFS thr</td>
<td>* Unbounded</td>
</tr>
<tr>
<td>No Priority</td>
<td>Nothr</td>
<td>Unbounded</td>
<td>None</td>
</tr>
</tbody>
</table>

**Constraint:**
- If FR-FCFS is with threshold: no more than $N^{thr}$ can cause reorder-interference with request under analysis →
- Total # reorder interfering requests from all cores $\leq N^{thr} \times \# \text{Interfered Requests}$

### Request-Dr Constraints

**Example: Reorder Requests**

- Intra-bank reorder
- Read (Write) requests from p
Overall Approach

Proposed

Information about Requests of Running tasks

- Req-Dr Constraints
- Job-Dr Constraints

Optimization Problem to maximize Total Delay

Total Delay Value
Values of all request variables
<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Pcr</th>
<th>2</th>
<th>Pncr</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nthr</td>
<td>8</td>
<td>Wbtch</td>
<td>16</td>
<td>PR</td>
<td>4</td>
</tr>
<tr>
<td>NB</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBp</td>
<td></td>
<td></td>
<td>8</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>noPart</td>
<td>PartAll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PartCr and p is Cr</td>
<td>PartCr and p is nCr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBcr</td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>noPart of PartCr</td>
<td>PartAll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBncr</td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>noPart of PartCr</td>
<td>PartAll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>#Reads</td>
<td>#writes</td>
<td>Total</td>
<td>BM</td>
<td>#Reads</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>matrix</td>
<td>280000</td>
<td>38428</td>
<td>318428</td>
<td>rspeed</td>
<td>2000</td>
</tr>
<tr>
<td>a2time</td>
<td>166000</td>
<td>21751</td>
<td>187751</td>
<td>pntrch</td>
<td>2000</td>
</tr>
<tr>
<td>aiffrt</td>
<td>101000</td>
<td>77234</td>
<td>178234</td>
<td>basefp</td>
<td>2000</td>
</tr>
</tbody>
</table>
Comparison with [CMU] across its supported platforms
CMU-JobDr is achieving better performance than Req-Dr in these 3 scenarios.

Comparison with [CMU] across its supported platforms.
CMU-JobDr is achieving better performance than Req-Dr in these 3 scenarios.

However, Proposed approach achieves the tightest bound: 24% and up to 22.6× than Job-Dr.

Comparison with [CMU] across its supported platforms.
Comparison with [CMU] across its supported platforms
However, Proposed approach achieves the tightest bound (34% tighter than Hassan-ReqDr)

Comparison with [CMU] across its supported platforms
Comparison with [YUN] across its supported platforms
YUN-JobDr is achieving better performance than Req-Dr in these 3 scenarios.

Comparison with [YUN] across its supported platforms.
YUN-JobDr is achieving better performance than Req-Dr in these 3 scenarios.

However, Proposed approach achieves the tightest bound: 25% and up to 42% than Job-Dr.

Comparison with [YUN] across its supported platforms.
Comparison with [YUN] across its supported platforms

- Req-Dr achieves tighter bound than Job-Dr in this scenario.
However, Proposed approach achieves the tightest bound (15% tighter than Hassan-ReqDr).

Comparison with [YUN] across its supported platforms.

Req-Dr achieves tighter bound than Job-Dr in this scenario.
Proposed provides up to 98% and 24% on average tighter bounds across all platform instances.

Comparison with Req-Dr across platforms

*Low-High case*
Comparison with Req-Dr across platforms

High-Low case
Proposed provides up to 71× and 18× on average tighter bound across all configurations!

Two main reasons are behind such significant gap: **no partitioning** (noPart) and **write batching** (WB). Both features, if considered, forces ReqDr to consider a pathological overly pessimistic scenario.

Comparison with Req-Dr across platforms

*High-Low case*
Previous Comparison with Req-Dr is for cases that are bounded by Req-Dr. Out of the 144 platform instance $\rightarrow$ 63 were proven to be unbounded.

Proposed Analysis is able to bound those leveraging the Job-Dr constraints in the optimization framework.

Comparison with Req-Dr across platforms

Unbounded cases by Req-Dr
Back to these two figures

How does the proposed hybrid analysis perform?
Summary

Intra-bank interfering requests:
1. Inter-bank conflict requests
2. Inter-bank reorder requests

Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests

Write batching
5. Write batching requests

Write batching requests
- Inter-bank close requests
- Inter-bank reorder requests

Inter-bank open requests
- Inter-bank close requests

ACT Latency
CAS Latency
Conflict Latency
Optimization problem:
1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job-driven analysis
3. Maximize total latency (summation of all components)

Summary

https://gitlab.com/FanusLab/memory-contention-analysis
Optimization problem:
1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)
Optimization problem:
1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)