SystemC TLM-2.0 Loosely-Timed Contention-Aware Modeling

Emad Arasteh
arasteh@chapman.edu

Fowler School of Engineering
Chapman University, Orange, CA, USA

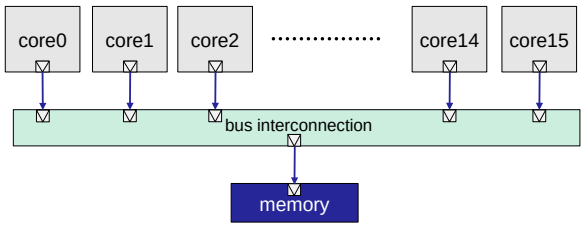
System Platform Exploration Lab (SPEL)

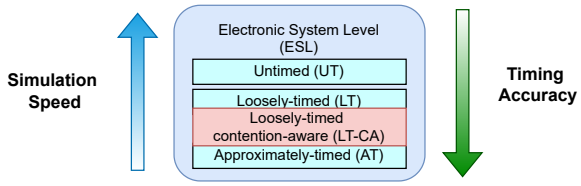
Outline

- Bus Contention
- Loosely-Timed Contention-Aware Modeling (LT-CA)
  - BusyUntil
  - BusyMap
  - TLM-2.0 LT-CA DNN
- Experimental Measurements and Results

Bus Contention

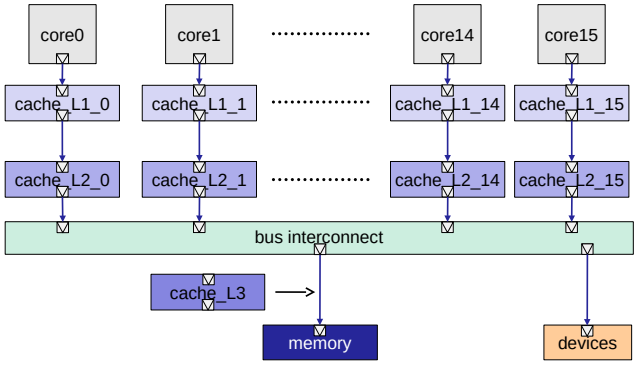
- Bus contention is a critical aspect in modeling modern multiprocessor system on a chip (MPSoC)
- SystemC TLM-2.0 aids system designers with performance estimation
  - Loosely-timed (LT)
    - adequate timing, fast simulation, no notion of contention
  - Approximately-timed (AT)
    - accurate timing, slow simulation, complex coding, model contention
- Can we model contention fast but accurately for early system design?
  - Should support different arbitration policies, temporal decoupling and multi-level interconnects
- We introduce Loosely-Timed Contention-Aware (LT-CA) modeling to model contention fast, accurate, and early in the design flow





Loosely-Timed Contention-Aware (LT-CA)

- LT-CA key features:
  - TLM-2.0 loosely-timed contention modeling with **high accuracy** at **high-speed** simulation
  - Early and efficient contention simulation and analysis supporting:
    - **First-come-first-served (FCFS)** and **round-robin (RR)** arbitration policies
    - **Temporal decoupling**
    - Multiple-level hierarchical interconnects, including **multi-level caches** or **multiple levels of buses**



LT-CA BusyUntil Contention

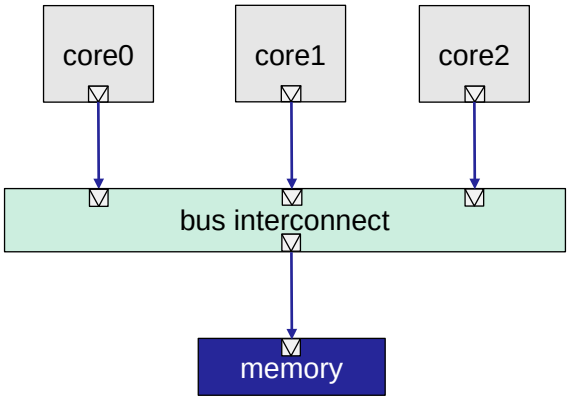
- We propose **BusyUntil**, which uses a state variable in the interconnect to keep track of contention
- By storing the contention in the timing annotation of the blocking transport interface, the transaction completes in a single function call
- Simple and effective approach

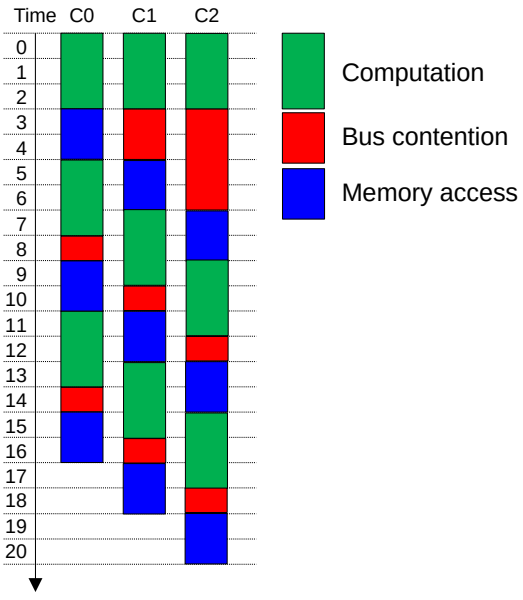
Algorithm 1: Modeling bus contention using a time stamp `busy_until` (adapted from [14])

```
Module Bus_BusyUntil begin
    target_socket S_in[NUM_TARGETS];
    initiator_socket S_OUT[NUM_INITIATORS];
    time bus_delay;
    time contention := 0;
    time busy_until := 0;
    Procedure ForwardRequest(trans, delay) begin
        // perform address translation
        socket := decode_and_translate(trans.address);
        // forward the transaction
        d1 := delay;
        socket→b_transport(transaction, delay);
        d2 := delay;
        memory_delay := d2 - d1;
        // maintain bus contention
        busy_span := bus_delay + memory_delay;
        busy_until := max(busy_until, global_time);
        busy_delay := busy_until - global_time;
        busy_until += busy_span;
        contention += busy_delay;
        delay += bus_delay + busy_delay;
    end
end
```


LT-CA BusyUntil Contention

- **BusyUntil** on synthetic SystemC model: **Bus3Init**

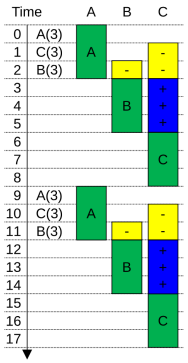




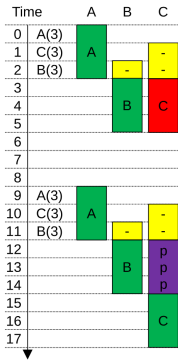
- Simulation trace for **BusyUntil** bus with global quantum value of zero

LT-CA BusyUntil Contention - Round Robin

- Not as simple as FCFS, LT-CA support round-robin (RR) scheduling
- To avoid complex data structure such as Payload Event Queue (PEQ), we tradeoff some accuracy for speed by approximating the bus contention
- Details of algorithm can be found in the [TECS'23] journal



(b) RR policy



(c) Approximate RR policy

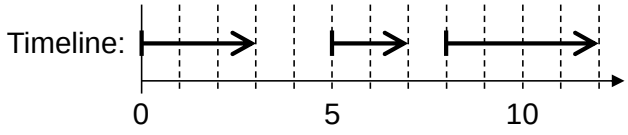
ALGORITHM 3: Approximating busy state in b_transport with Round-Robin (RR) arbitration policy

Input : Reference to transaction object
Reference to delay object
socket id where this request came from
Output: Reference to transaction object
Reference to the updated delay object
State : Circular array request[N] where each request is a tuple with start_time, memory_delay, and penalty (all initialized to 0)

```
Procedure interconnect_b_transport(transaction, delay, id) begin
    // perform the memory transaction
    d1 := delay;
    socket->b_transport(transaction, delay);
    d2 := delay;
    memory_delay := d2 - d1;
    // update the scheduler state
    request[id].start_time := current_time;
    request[id].memory_delay := memory_delay;
    if earliest_active_request != FindEarliestOtherActiveRequest(id) then
        RescheduleActiveRequests(earliest_active_request, id, memory_delay);
    end
    busy_delay := request[id].start_time - current_time;
    busy_delay += request[id].penalty;
    request[id].penalty := 0;
    // update the delay
    delay += interconnect_delay + busy_delay;
end
Function FindEarliestOtherActiveRequest(id) begin
    earliest_active_request := none;
    earliest := infinity;
    for r ∈ request[N] from id + 1 to id − 1 do
        if r.start_time + r.memory_delay >= current_time then
            if r.start_time < earliest then
                earliest := r.start_time;
                earliest_active_request := r;
            end
        end
    end
    return earliest_active_request;
end
Procedure RescheduleActiveRequests(earliest_active_request, id, penalty) begin
    busy_until := earliest_active_request.start_time + earliest_active_request.memory_delay;
    for r ∈ request[N] from earliest_active_request + 1 to earliest_active_request − 1 do
        if r.start_time + r.memory_delay <= current_time then
            // skip old requests
        else
            if r.start_time < busy_until then
                r.start_time := busy_until;
                if r != id then
                    r.penalty += penalty;
                end
            end
        end
        busy_until += r.memory_delay;
    end
end
```


LT-CA BusyMap Contention

- **BusyUntil** is simple and effective approach but requires improvements for temporal decoupling and multi-level interconnects
- We introduce a new data structure, **BusyMap**, to replace the state variable in **BusyUntil**
- **BusyMap** allows temporally decoupled initiator modules that use *out-of-order* transactions with different `delay` offsets from the simulator `global_time`
- Ordered map of key-value (k, v) of *sc_time*
  - key k specifies the start time when the resource becomes busy
  - value v specifies the duration of how long the resource is used



Busy Periods: [0, 3) [5, 7) [8, 12)

Map Elements: 0→3 5→2 8→4

Type: `std::map<sc_time,sc_time>`

LT-CA BusyMap Contention

- Bus contention model contains the ordered **busy_map** with its essential member variables and methods
- Details of algorithms can be found in the [DATE'24] paper

Algorithm 2: Modeling bus contention using BusyMap

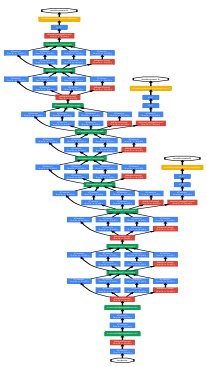
```
Module Bus_BusyMap begin
    target_socket S_in[NUM_TARGETS];
    initiator_socket S_OUT[NUM_INITIATORS];
    time bus_delay;
    time contention := 0;
    ordered_map<time,time> busy_map;
    Procedure ForwardRequest (trans, delay) begin
        // perform address translation
        socket := decode_and_translate(trans.address);
        // forward the transaction
        d1 := global_time + delay;
        socket→b_transport(transaction, delay);
        d2 := global_time + delay;
        memory_delay := d2 - d1;
        // maintain bus contention
        busy_span := bus_delay + memory_delay;
        AdvanceTime();
        available_slot := FindFreeSlot(d1, busy_span);
        busy_delay := availabe_slot - d1;
        SetBusy(available_slot, busy_span);
        contention += busy_delay;
        delay += bus_delay + busy_delay;
    end
    Procedure AdvanceTime begin
        if busy_map.empty() then
            return;
        end
        keep := busy_map.upper_bound(global_time);
        if keep = busy_map.begin() then
            return;
        end
        cut := prev(keep);
        if cut→start = global_time then
            busy_map.erase(busy_map.begin(), cut);
            return;
        end
        if (cut→start + cut→duration) > global_time then
            cut_duration := cut→start + cut→duration -
              global_time;
            busy_map.erase(busy_map.begin(), keep);
            busy_map[global_time] := cut_duration;
        else
            if keep = busy_map.end() then
                busy_map.clear();
            else
                busy_map.erase(busy_map.begin(), keep);
            end
        end
    end
end
```

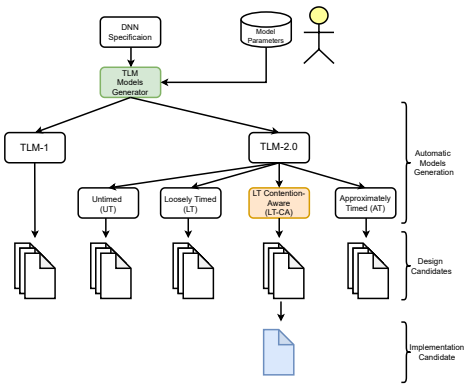
Algorithm 3: Modeling bus contention using BusyMap (continued)

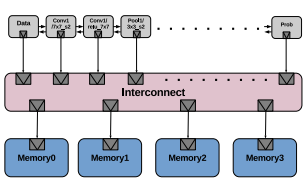
```
Function FindFreeSlot (earliest,span) begin
    if busy_map.empty() then
        return earliest;
    end
    iter := busy_map.upper_bound(earliest);
    if iter = busy_map.begin() then
        gap_at := 0;
    else
        gap_at := prev(iter)→start + prev(iter)→duration;
    end
    if gap_at<earliest then
        gap_at := earliest;
    end
    while iter ≠ busy_map.end() do
        gap_duration := iter→start - gap_at;
        if span ≤ gap_duration then
            return gap_at;
        end
        gap_at := iter→start + iter→duration;
        iter++;
    end
    return gap_at;
end
Procedure SetBusy (slot,span) begin
    if busy_map.empty() then
        busy_map[slot] := span;
        return;
    end
    r := busy_map.upper_bound(slot);
    if r = busy_map.begin() then
        if r→start = slot + span then
            busy_map[slot] := span + r→duration;
            busy_map.erase(r);
        else
            // no adjacency, insert a new element
            busy_map[slot] := span;
        end
        return;
    end
    l := prev(r);
    if l→start + l→duration = slot then
        if (r ≠ busy_map.end()) and (r→start = slot + span)
          then
            // merge in between adjacent elements
            l→duration += span + r→duration;
            busy_map.erase(r);
        else
            // merge with adjacent element on the left
            l→duration += span;
        end
    else
        if (r ≠ busy_map.end()) and (r→start = slot + span)
          then
            // merge with adjacent element on the right
            busy_map[slot] := span + r→duration;
            busy_map.erase(r);
        else
            // no adjacency, insert a new element
            busy_map[slot] := span;
        end
    end
end
```


TLM-2.0 LT-CA Deep Neural Network (DNN)

- DNNs are data-intensive software applications that demand early attention to performance metrics in the design flow
- SystemC enables rapid systematic evaluation of design candidates for lower-level implementation, e.g., RTL
- We implement SystemC TLM DNN modeling framework
  - Generic and self-contained layers, reusability and modularity





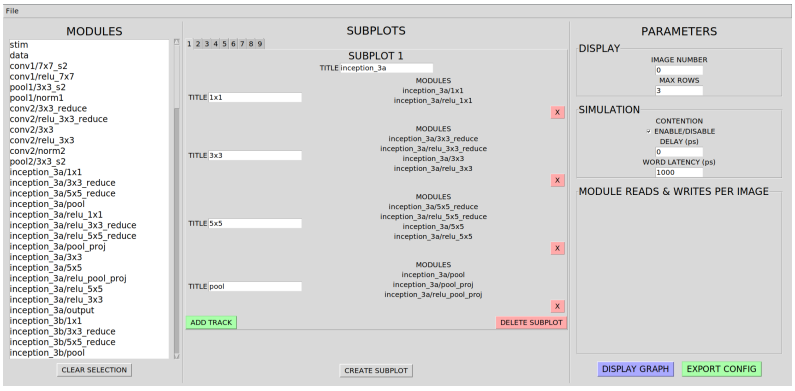


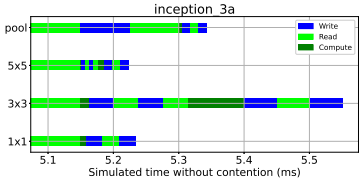
TLM-2.0 LT-CA DNN

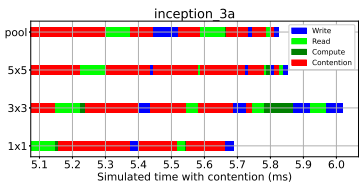
- TLM DNN framework, **netspec**, is configurable, customizable and extensible
  - TLM-1 and TLM-2.0
  - UT, LT, LT-CA and AT
  - Buffer architecture
  - Interconnect addressing
  - Memory and compute latency

Visualization of TLM-2.0 LT-CA DNN

- We introduce, **netmemvisual**, visualization tool to plot LT and LT-CA timing diagrams for rapid contention analysis
- Interactive and cross-platform supporting command-line and graphical user interfaces

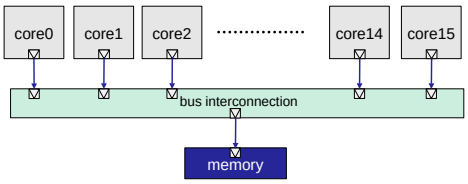






Experimental Measurements and Results

- Measure total simulated time of GoogLeNet for different computational capacities and memory latencies Using FCFS scheduling (in seconds)



	Loosely-timed				Loosely-timed contention-aware				Approximately-timed			
comp / mem	1ns	10ns	100ns	1000ns	1ns	10ns	100ns	1000ns	1n	10ns	100ns	1000ns
1000 GFLOPS	0.088	0.877	8.763	87.62	1.161	11.61	116.1	1161	1.161	11.61	116.1	1161
100 GFLOPS	0.403	0.888	8.773	87.63	1.164	11.61	116.1	1161	1.164	11.61	116.1	1161
10 GFLOPS	3.603	4.034	8.888	87.73	3.618	11.64	116.1	1161	3.623	11.64	116.1	1161
1 GFLOPS	35.60	36.03	40.34	88.88	35.61	36.18	116.4	1161	35.62	36.23	116.4	1161

- Generic LT does not take contention into account
- LT-CA considers the effect of contention and shows high accuracy in simulated time
- AT accurately models contention, hence showing a significant increase in simulated time

Experimental Measurements and Results - Accuracy

- Accuracy of LT and LT-CA BusyUntil (FCFS) Compared to Reference AT

	Loosely-timed				Loosely-timed contention-aware			
comp / memory	1ns	10ns	100ns	1000ns	1n	10ns	100ns	1000ns
1000 GFLOPS	7.6%	7.6%	7.5%	7.5%	100.0%	100.0%	100.0%	100.0%
100 GFLOPS	34.6%	7.6%	7.6%	7.5%	100.0%	100.0%	100.0%	100.0%
10 GFLOPS	99.4%	34.7%	7.7%	7.6%	99.9%	100.0%	100.0%	100.0%
1 GFLOPS	99.9%	99.4%	34.7%	7.7%	100.0%	99.9%	100.0%	100.0%

- LT models shows very low accuracy
- LT-CA models show almost complete accuracy

Experimental Measurements and Results - Accuracy

- Accuracy of LT and LT-CA BusyUntil (RR) Compared to Reference AT

	Loosely-timed				Loosely-timed contention-aware			
comp / mem	1ns	10ns	100ns	1000ns	1n	10ns	100ns	1000ns
1000 GFLOPS	7.6%	7.6%	7.5%	7.5%	93.1%	94.2%	93.8%	93.7%
100 GFLOPS	34.6%	7.6%	7.6%	7.5%	94.2%	93.1%	94.2%	93.8%
10 GFLOPS	99.4%	34.7%	7.7%	7.6%	99.8%	94.2%	93.1%	94.2%
1 GFLOPS	99.9%	99.4%	34.6%	7.7%	100.0%	99.8%	94.2%	93.1%

- Same pattern applies for LT and LT-CA models in RR scheduling
- LT-CA for RR shows a minor decrease in accuracy, it is still a very accurate model compared to LT

Experimental Measurements and Results - Speed

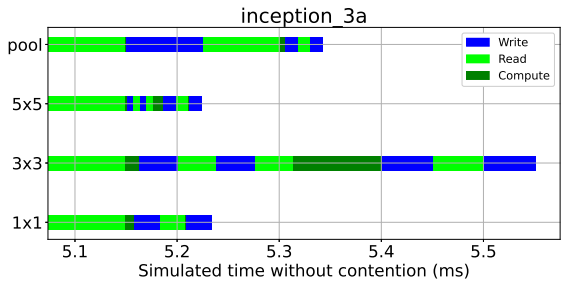
- Measure total simulator run-time of GoogLeNet for different computational capacities and memory latencies using FCFS scheduling on a 32-core host (in seconds)

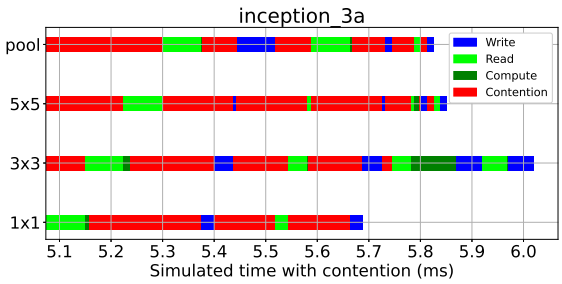
	Loosely-timed				Loosely-timed contention-aware				Approximately-timed			
comp / mem	1ns	10ns	100ns	1000ns	1ns	10ns	100ns	1000ns	1n	10ns	100ns	1000ns
1000 GFLOPS	124.4	121.6	120.4	119.8	141.3	140.0	139.0	137.9	6496	6594	6520	6473
100 GFLOPS	106.9	123.6	123.0	124.0	145.8	145.0	144.9	141.4	6476	6504	6569	6434
10 GFLOPS	105.0	108.5	123.4	131.8	126.5	146.2	142.9	142.7	6310	6669	6544	6529
1 GFLOPS	98.9	104.3	108.9	127.6	124.9	124.6	143.8	141.7	6493	6360	6621	6473

- LT models simulate faster than their LT-CA and AT counterparts
- LT-CA models simulate slightly slower than LT models (1.2x) but simulate order of magnitude faster than AT
- AT models simulate 46x slower than LT and LT-CA

Experimental Measurements and Results - Visualization

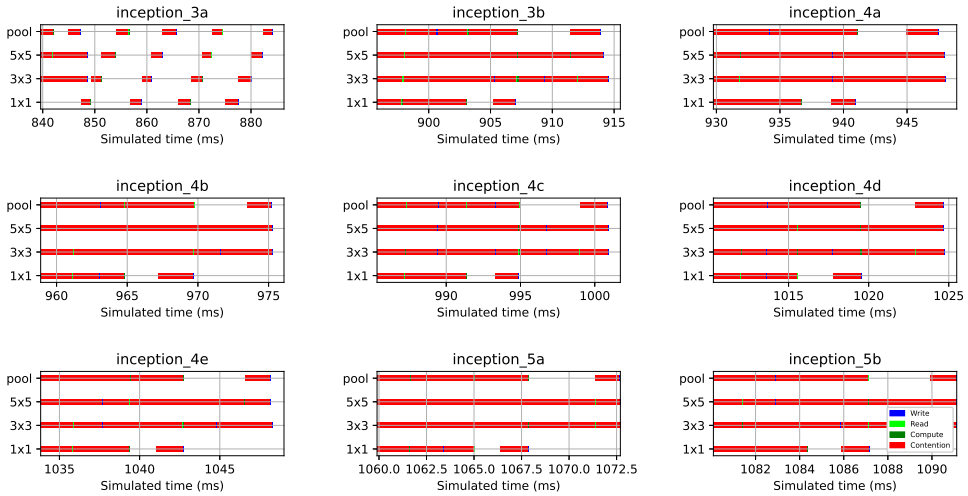
- TLM timing diagrams for the first inception module in GoogLeNet with pass of 1 image
- LT does not model contention so layers in parallel tracks access memory without blocking each other
- In LT-CA model, layers are blocked and wait until access becomes available red areas





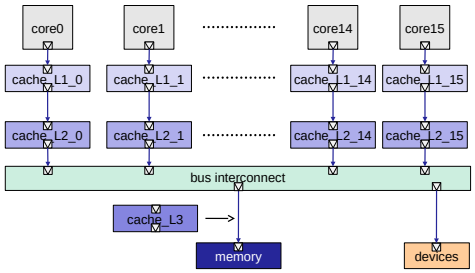
Experimental Measurements and Results - Visualization

- Contention significantly impacts performance when the DNN pipeline is fully loaded with images (image #75)
- As a result, most layers experience blocking due to contention



Experimental Measurements and Results - BusyMap

- Parallel JPEG simulation results running on RISC-V SMP VP

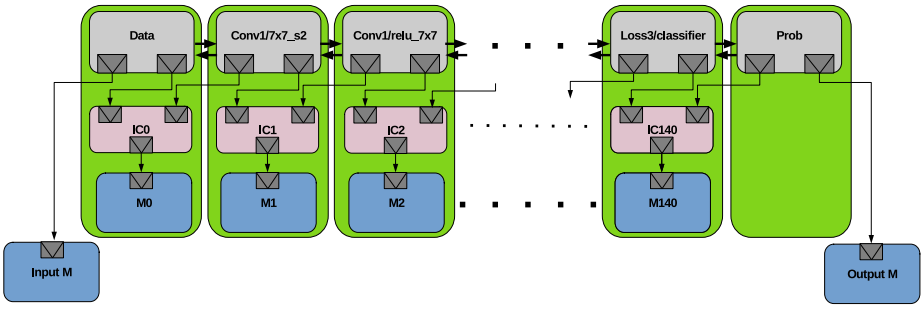


	Simulated time		Bus Contention		Simulator run-time		# wait statements (cores)		# wait statements (caches)	
Global quantum	BusyUntil	BusyMap	BusyUntil	BusyMap	BusyUntil	BusyMap	BusyUntil	BusyMap	BusyUntil	BusyMap
0ns	2.33s	2.33s	3.94s	4.00s	30m49s	18m50s	63574400	63628015	21359780	0
10ns	N/A	2.33s	N/A	4.00s	N/A	18m48s	N/A	63628015	N/A	0
100ns	N/A	2.33s	N/A	3.90s	N/A	13m33s	N/A	22157149	N/A	0
1000ns	N/A	2.44s	N/A	4.71s	N/A	11m38s	N/A	6478006	N/A	0
10000ns	N/A	2.65s	N/A	5.95s	N/A	10m20s	N/A	896267	N/A	0

- BusyMap significantly improves simulator run-time (3x)
- BusyMap supports temporal decoupling
- BusyMap has high accuracy in simulated time and contention

Research Outlook

- **Fast** and **accurate** LT-CA modeling enables the **efficient** exploration of alternative memory organizations
- **Early** detection of memory contentions suggests that local memories close to computing cores can eliminate memory contention in such data-intensive applications



Conclusion

- This work improves high-level modeling and simulation of interconnect contention by navigating trade-offs between simulation speed and timing accuracy
- Specifically, we proposed BusyUntil and BusyMap for modeling bus contention in SystemC TLM-2.0 LT-CA models
  - Supports FCFS and RR arbitration policies
  - Supports temporal decoupled with *out-of-order* transactions
  - Can effectively model multi-level interconnects and caches
- Using BusyUntil, we achieve a speedup of up to 46x on a 32-core host with only 1% accuracy loss in simulated time
- For temporal decoupled with multi-level caches models (BusyMap), we achieve a speedup of up to 3x on a 16-core host with only 10% accuracy loss in simulated time and bus contention compared to BusyUntil

References (1)

- [DATE'24] E. Arasteh, V. Govindasamy, R. Dömer: *"BusyMap, an Efficient Data Structure to Observe Interconnect Contention in SystemC TLM-2.0"*, Proceedings of Design, Automation and Test in Europe Conference, Valencia, Spain, March 2024.
- [DVCon'23b] C. Raccomandato, E. Arasteh, R. Dömer: *"Grid-based Mapping and Analysis of a GoogLeNet CNN using MapGL Editor"*, Proceedings (engineering track) of the Design and Verification Conference in Europe, Munich, Germany, November 2023.
- [DVCon'23a] C. Raccomandato, E. Arasteh, R. Dömer: *"MapGL: Interactive Application Mapping and Profiling on a Grid of Processing Cells"*, Proceedings (research track) of the Design and Verification Conference in Europe, Munich, Germany, November 2023.
- [TECS'23] E. Arasteh, R. Dömer: *"Fast Loosely-Timed System Models with Accurate Memory Contention"*, Journal of ACM Transactions on Embedded Computing Systems, July 2023.
- [FSE'23] N. Farzan, E. Arasteh, *"Visualizing Transaction-Level Modeling Simulations of Deep Neural Networks"*, Fowler School of Engineering, Technical Report 23-01, August 2023.
- [IESS'22] V. Govindasamy, E. Arasteh, R. Dömer: *"Minimizing Memory Contention in an APNG Encoder using a Grid of Processing Cells"*, Proceedings of the International Embedded Systems Symposium, "Designing Modern Embedded Systems: Software, Hardware, and Applications" Springer, Lippstadt, Germany, November 2022.
- [FDL'21] E. Arasteh, R. Dömer: *"Improving Parallelism in System Level Models by Assessing PDES Performance"*, Proceedings of Forum on Specification and Design Languages, Antibes, France, Sep. 2021.

References (2)

- [CECS'21] E. Arasteh, R. Dömer: *"Systematic Evaluation of Six Models of GoogLeNet using PDES"*, Center for Embedded and Cyber-Physical Systems, Technical Report 21-03, 2021, Sep. 2021.
- [Springer'20] R. Dömer, Z. Cheng, D. Mendoza, E. Arasteh: *"Pushing the Limits of Parallel Discrete Event Simulation for SystemC"*, in "A Journey of Embedded and Cyber-Physical Systems" by Jian-Jia Chen, Springer Nature, Switzerland, August 2020.
- [DATE'20] D. Mendoza, Z. Cheng, E. Arasteh, R. Dömer: *"Lazy Event Prediction using Defining Trees and Schedule Bypass for Out-of-Order PDES"*, Design, Automation and Test in Europe Conference, Grenoble, France, March 2020.
- [ASPDAC'20] Z. Cheng, E. Arasteh, R. Dömer: *"Event Delivery using Prediction for Faster Parallel SystemC Simulation"*, Asia and South Pacific Design Automation Conference, Beijing, China, Jan. 2020.
- [IESS'19] E. Arasteh, R. Dömer: *"An Untimed SystemC Model of GoogLeNet"*, Proceedings of the International Embedded Systems Symposium, Friedrichshafen, Germany, Sep. 2019.

Acknowledgments

- I would like to give a special thanks to my collaborators:
  - The team at UC Irvine
    - Rainer Dömer
    - Vivek Govindasamy
    - Zhongqi Cheng, Daniel Mendoza, Claudio Raccomandato
  - My team at Chapman
    - Nataniel Farzan
    - Ewan Shen
    - Maha Bhatti