Thermal and Energy Efficiency of Heterogeneous Mobile SoCs: Current and Upcoming Trends/Challenges

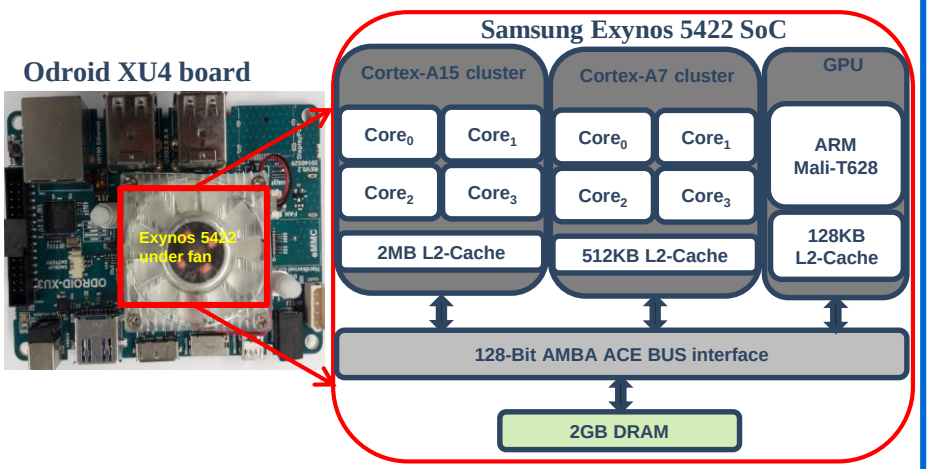
Amit Kumar Singh[1], Karunakar Reddy Basireddy[2], Geoff V. Merrett[3], Bashir M. Al-Hashimi[3]

[1]University of Essex, UK, [2]ARM, Bangalore, India, [3]University of Southampton, UK

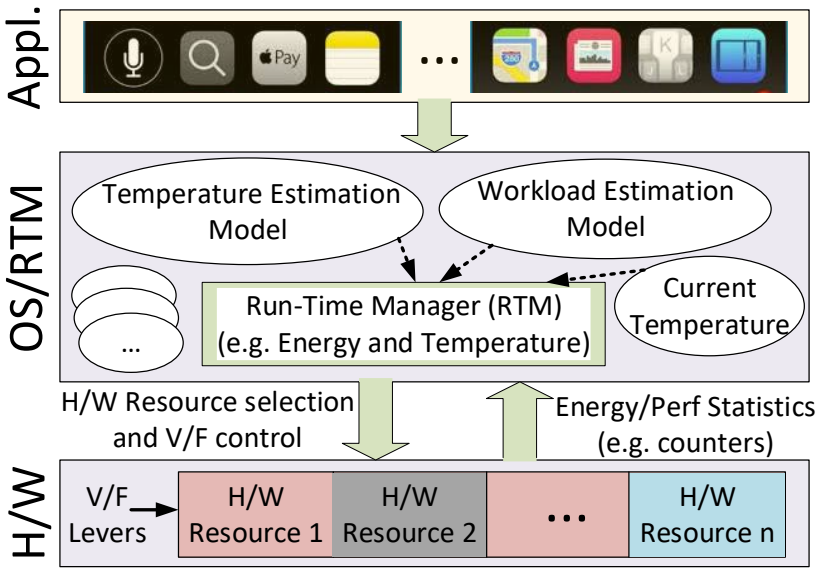
Introduction

- Modern mobile SoCs are containing greater number of heterogeneous cores to support highly diverse and varying workloads
- To meet performance, energy consumption and thermal efficiency requirements, the process of thread-to-core mapping and setting DVFS levels can be exploited
- The process becomes complex with increasing concurrent applications and heterogeneity

A Heterogeneous Mobile SoC



Mapping and DVFS Process



State-of-the-art: Shortcomings

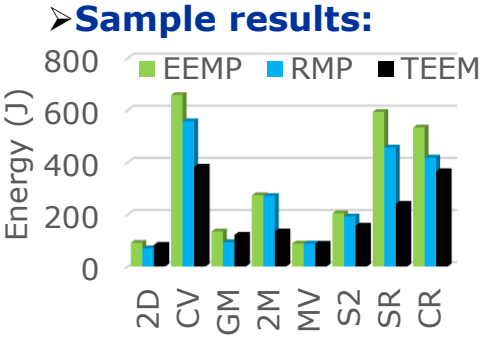
- Mostly use heavy application-dependent profile data -> not efficient in managing dynamic workloads with unknown applications
- Do not perform adaptations (changing the mappings and/or DVFS settings) at an application arrival/completion, and performance variations.

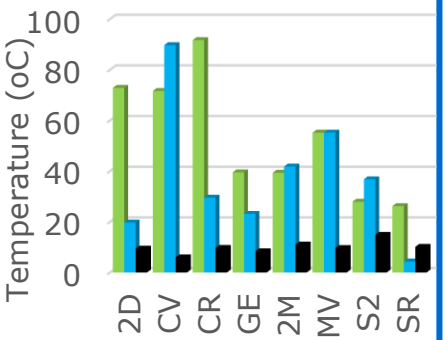
Adaptation for Thermal and Energy Efficiency

- **Adapting DVFS with reliance on profiled data**
  - Predictive Thermal Management for Energy-Efficient Execution of Concurrent Applications on Heterogeneous Multicores, *in IEEE TVLSI 2019*
  - EdgeCoolingMode: An Agent Based Thermal Management Mechanism for DVFS Enabled Heterogeneous MPSoCs, *in IEEE VLSID 2019*
  - Teem: Online thermal-and energy-efficiency management on cpu-gpu mpsocs, *in IEEE DATE 2019*
- **Adapting DVFS without reliance on profiled data**
  - User Interaction Aware Reinforcement Learning for Power and Thermal Efficiency of CPU-GPU Mobile MPSoCs, *in IEEE DATE 2020*
- **Adapting Mapping and DVFS with reliance on profiled data**
  - Collaborative adaptation for energy-efficient heterogeneous mobile SoCs, *in IEEE TC 2019*
    - Adaptation happens at application arrival and departure
    - It can be extended to consider a thermal threshold
- **Adapting Mapping and DVFS without reliance on profiled data**
  - AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multi-cores, *in IEEE TCAD 2019*
    - Adapts to runtime execution scenarios efficiently by monitoring the application status, and performance/workload variations.
    - It can be extended to consider a thermal threshold

Experiments

- On Odroid XU3/XU4, Galaxy Note 9, Huawei P20 Lite
- Currently considering Google Pixel 3
- **Sample results:**





Upcoming Trends & Conclusions

- Hierarchical Management for Multi-cluster SoCs
- Increasing Application Domains
- Multi-objective Optimizations
- Secure and Efficient Interaction with Cloud
- Adaptation for thermal and energy efficiency
- Dynamic Energy and Thermal Management of Multi-Core Mobile Platforms: A Survey, *in IEEE D & T, 2020.*

University of Essex · University of Southampton · arm

A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim

Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, Tushar Krishna

anandsamajdar@gatech.edu

Motivation

Number of applications using DNNs

Sizes of state-of-the art networks

Tolerance on latency

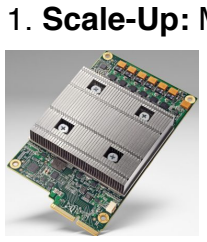
Energy efficiency

Benefits from device scaling

Need more powerful accelerators

Two approaches for scaling

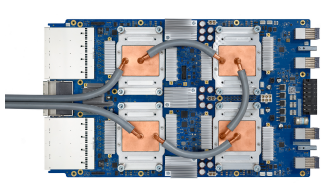
1. **Scale-Up:** Make big chips



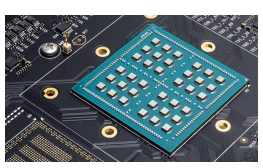
TPU v1

Cerebras WSP

2. **Scale-Out:** Make big collection of chips



TPU v3

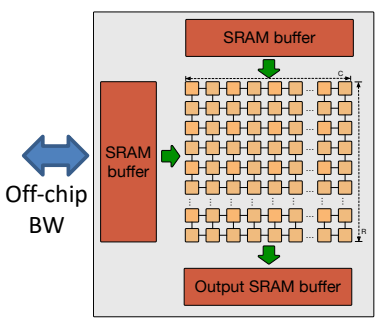


Nvidia Simba

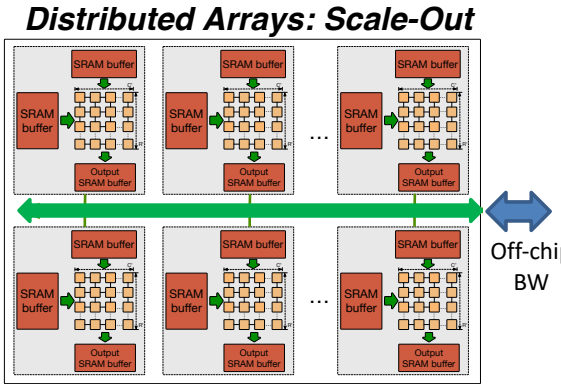
But which design approach is more beneficial?

Target Systems

Monolithic Array: Scale-Up



Distributed Arrays: Scale-Out



*Both configurations have **equal***

— Number of MAC units

— Total size of SRAM buffers

Which configuration has:

a. Better Runtime?

b. Higher Efficiency?

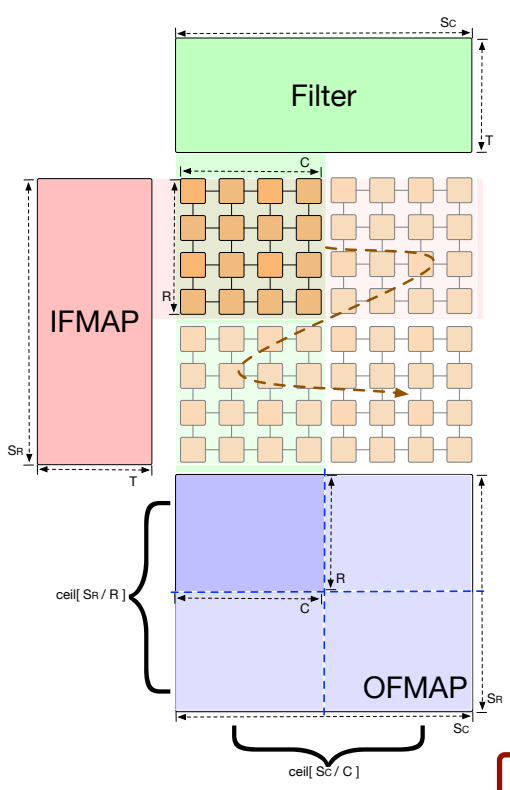
c. Lower off-chip bandwidth demand?

Systematic analysis using ***analytical model*** and ***SCALE-Sim****

* https://github.com/ARM-software/SCALE-Sim

Analytical model

Monolithic Array: Scale-Up



Mapping for OS dataflow

R: Rows, **C**: Cols

S_R : Spatial mapping dimension along rows

S_C : Spatial mapping dimension along cols

T : Temporal mapping dimension

	Spatial (S_R)	Spatial (S_C)	Temporal (T)
OS	OFMAP Px per channel	Num Filter	Elem per Conv Window
WS	Elem per Conv Window	Num Filter	OFMAP Px per channel
IS	Elem per Conv Window	OFMAP Px per channel	Num Filter

Fold: *Number of serial steps*

Runtime, τ = Runtime for one-fold **X** Number of folds

Number of folds = Folds in row dimension x Folds in col dimension

$= \lceil S_R / R \rceil \times \lceil S_C / C \rceil$

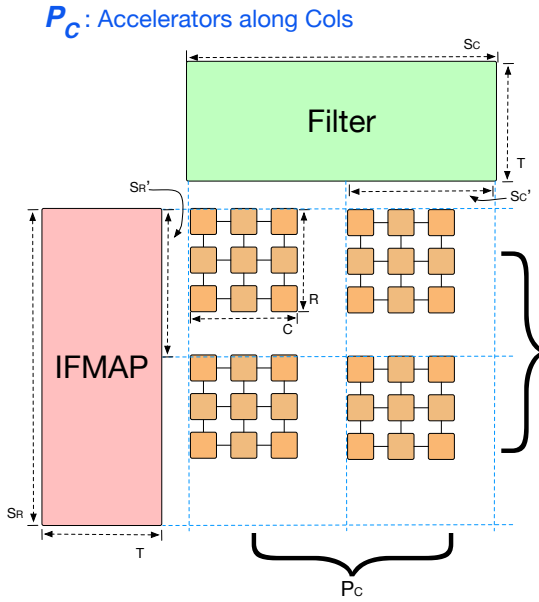
Runtime one-fold $= (2R + C + T - 2)$

$$\tau = (2R + C + T - 2) \times \lceil S_R / R \rceil \times \lceil S_C / C \rceil$$

Distributed Arrays: Scale-Out

P_R : Accelerators along Rows

P_C : Accelerators along Cols



Distributed mapping for OS dataflow

Accelerators arranged as a P_R x P_C spatial grid

Each unit works on a part of the problem

All units run in parallel

Number of folds per unit

$= \lceil S_R' / R \rceil \times \lceil S_C' / C \rceil$

Where,

$S_R' = \lceil S_R / P_R \rceil$

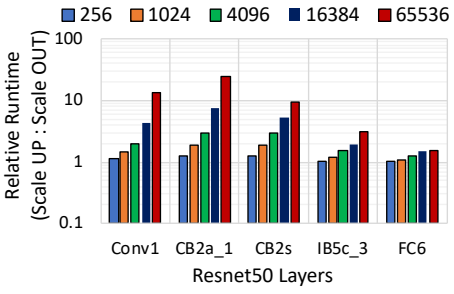
$S_C' = \lceil S_C / P_C \rceil$

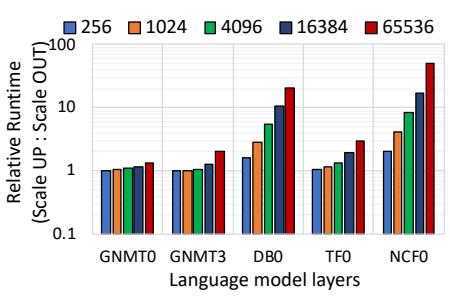
Total Runtime = Runtime of one unit

$$\tau = (2R + C + T - 2) \times \lceil S'_R / R \rceil \times \lceil S'_C / C \rceil$$

Results

Performance





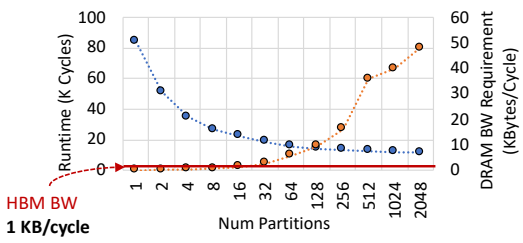
Performance improves with more partitions

Runtime per fold **Number of folds**

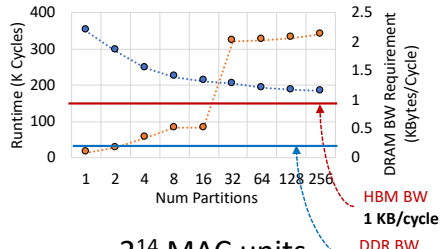
$\tau = (2R + C + T - 2) \times \lceil S_R' / R \rceil * \lceil S_C' / C \rceil$

Both terms lower for Scale-Out

Off - Chip Accesses



2^{18} MAC units

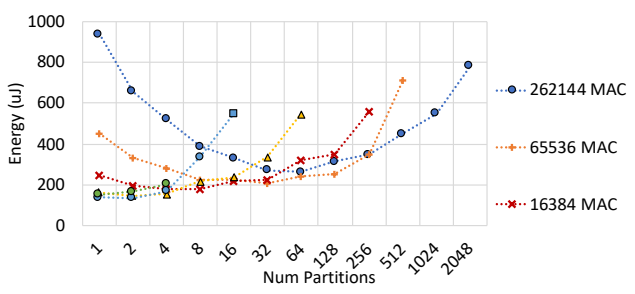


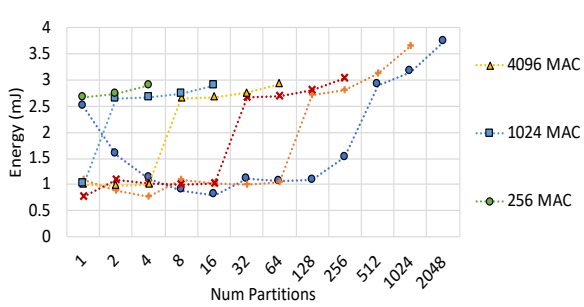
2^{14} MAC units

Monolithic configurations are best with least off chip accesses

Off-chip accesses increase due to loss of spatio-temporal reuse

Energy Efficiency





Definite sweet spots exist

Most efficient configuration dependent on the interplay of reuse and performance

Conclusions

Scaling DNN accelerators efficiently is non intuitive even for simple designs like a systolic array

We propose an analytical model and simulation infrastructure to help make scaling decisions

Our analysis depict that we can find out sweet spots for energy efficient and performant configurations

Link to paper: https://bit.ly/3fSD8O0

Hardware and Compiler Support for Programmable Non-Volatile Memory

Apostolos Kokolis, Thomas Shull, Jian Huang and Josep Torrellas

ILLINOIS

Programming Non-Volatile Memory 1

- NVM offers an attractive combination of capacity, persistence and performance
- Users have access to NVM at byte-granularity
- NVM success depends on easy to use frameworks
- Most frameworks put the burden on programmers

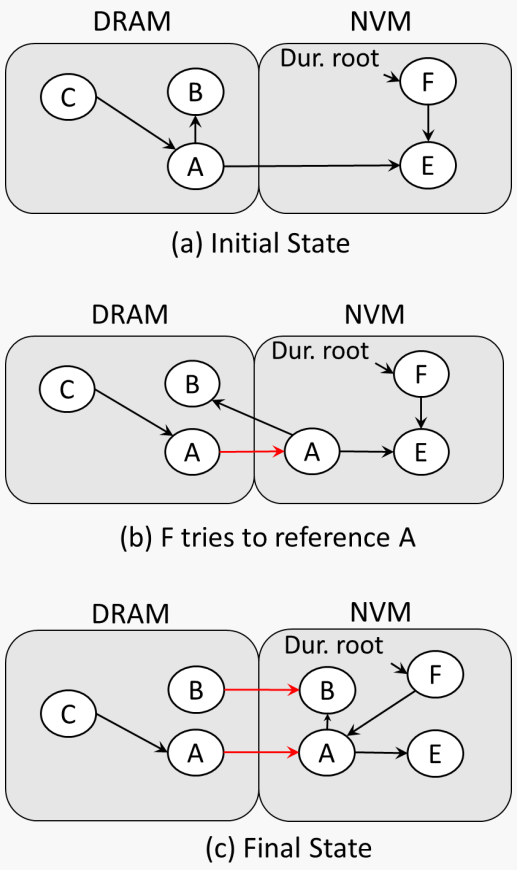
Problem: *Hard to program applications*

Persistency by Reachability: Programmable NVM with AutoPersist (PLDI'19 http://iacoma.cs.uiuc.edu/iacoma-papers/pldi19_1.pdf) 2

- Minimize user involvement
- Users only mark entry points (*durable roots*) to persistent data
- Rely on the compiler and runtime to dynamically persist objects and ensure crash consistency

Problem: *The runtime introduces overheads to check the object state during execution*

Operation of Persistency by Reachability 3



- Durable object (*F*) tries to reference a DRAM object
- Runtime identifies that *A* is not durable
- Moves *A* to NVM
- Moves the transitive closure of *A* to NVM
- DRAM objects *A* and *B* act as forwarding pointers to NVM objects *A* & *B*

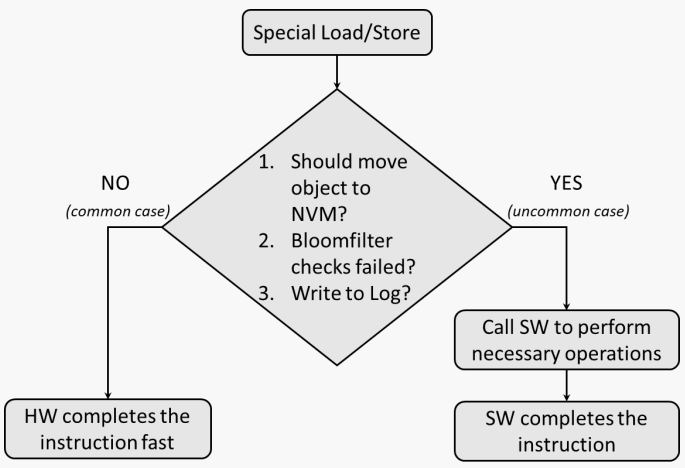
Hardware Support for NVM Programming with P-INSPECT (MICRO'20 http://iacoma.cs.uiuc.edu/iacoma-papers/micro20_2.pdf) 4

- Replace runtime checks of object state with HW
- On an access to an object, HW checks if:
  - Object is part of the NVM structures
  - Object is in the process of moving to NVM
  - Object is inside a transaction
- Added software managed Bloomfilters in HW to check for membership of objects

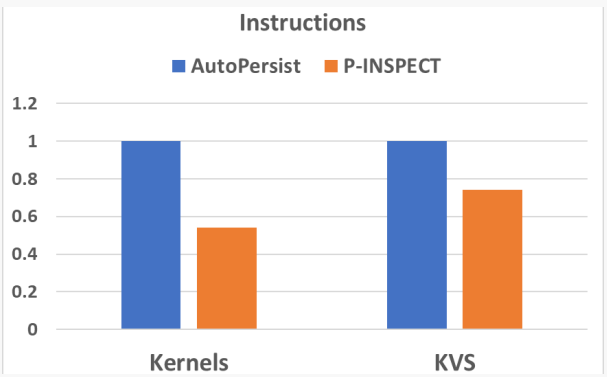
Hardware Operations 5

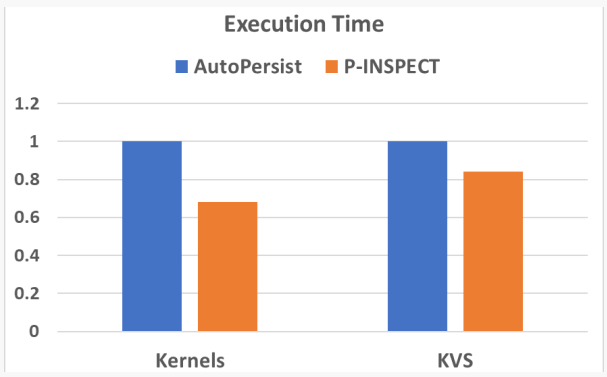
- Bloomfilters are accessed as part of reads & writes
- We need instructions to check and operate on the Bloomfilters

Special loads & stores	Bloomfilter Operations
Check object state	Insert elements and clear the BFs



Evaluation Highlights 6





- Ease NVM programming
- Simple HW
- Performance

TornadoVM: Transparent Hardware Acceleration of Managed Languages in the ARM Ecosystem

MANCHESTER 1824
The University of Manchester

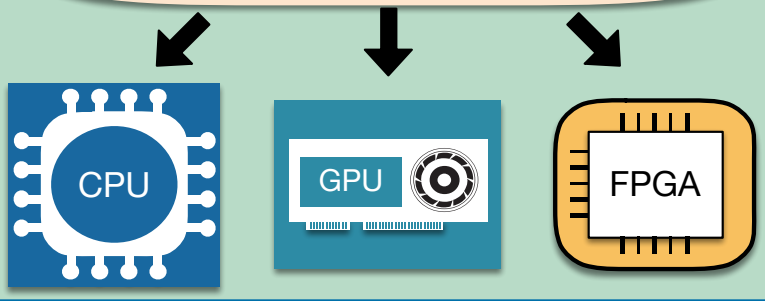
E2Data

Florin Blanaru, Juan Fumero, Thanos Stratikopoulos,
Michail Papadimitriou, Maria Xekalaki, Christos Kotselidis

Programming Accelerators

Compiled Languages

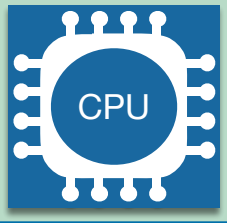
C/C++ | OpenCL | CUDA | HLS



Managed Languages

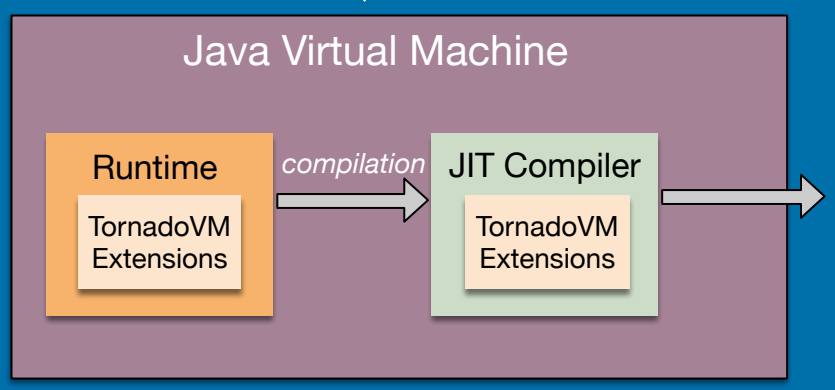
Java | JS | Ruby | R | Python

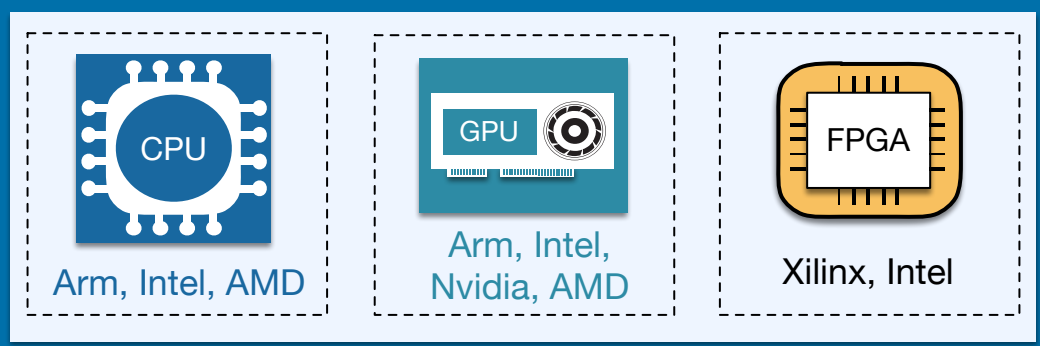
Virtual Machine



TornadoVM

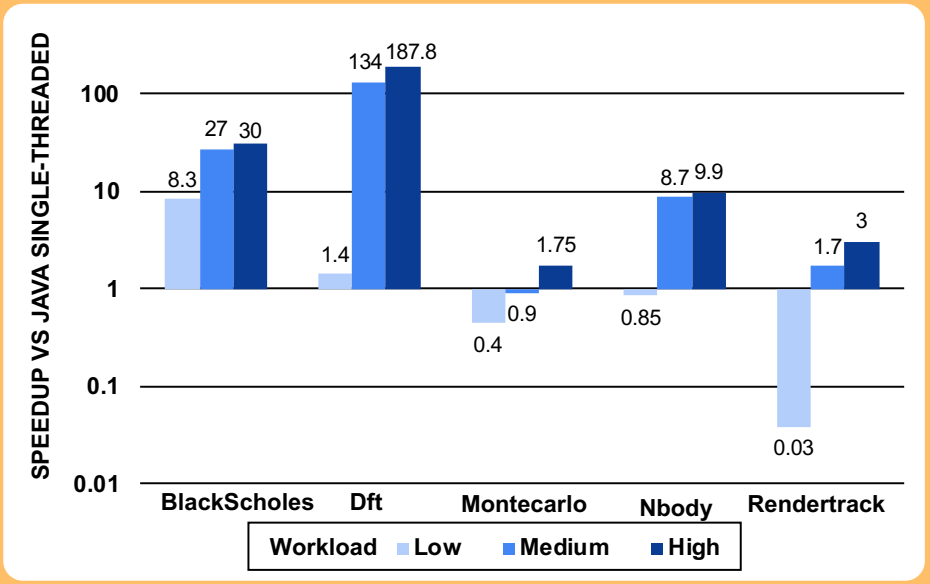
Java methods



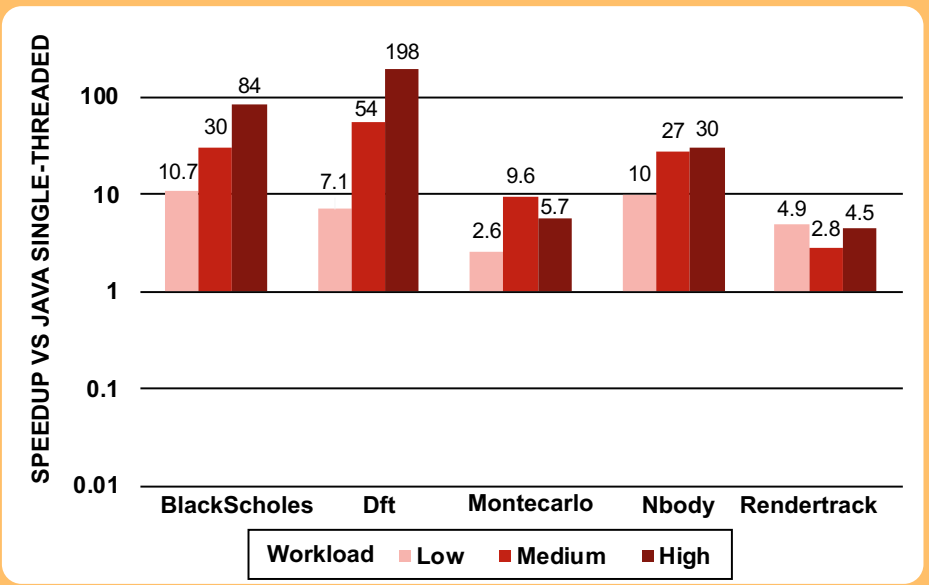


✓Modular Architecture ✓Vendor Independence ✓Transparent Acceleration

Performance



Mali G71 GPU vs Intel i7-8700K



Mali G71 GPU vs Arm A73/A53 big.LITTLE

Relation to the Arm Ecosystem

IoT SoC Solutions (e.g. Hikey960 SoC)

(e.g. AWS Graviton EC2)

LIMELIGHT+: GRAPH THEORY AND SEMANTIC LEARNING FOR DATACENTRE CALL GRAPHS

PENGFEI ZHENG,[1] XIAODONG WANG,[2] KIM HAZELWOOD,[2] DAVID BROOKS,[2] **BENJAMIN C. LEE**[3]

[1] University of Wisconsin, [2] Facebook Research, [3] University of Pennsylvania

1 INTRODUCTION

Datacentres

- Optimize hardware, software, energy efficiency.
- Identify workloads' optimization targets

Profiles

- Use stack traces to detail how software consumes processor time
- Use call graph to detail how software composes

Objectives

- Reveal structure of massive software systems
- Identify hotspots for optimization

2 CHALLENGES

Discovering Optimization Targets

- Look beyond fine-grained hotspots (*e.g.*, memcmp, write)
- Discover coarse-grained computation (*e.g.*, key-value seeks)
- Discover synonymous computation (*e.g.*, malloc, calloc, mallocx)

Interpreting Call Graphs

- Analyze DAG for functions, caller-callee relationships
- Scale analysis to O(10K) functions, O(100K) relationships

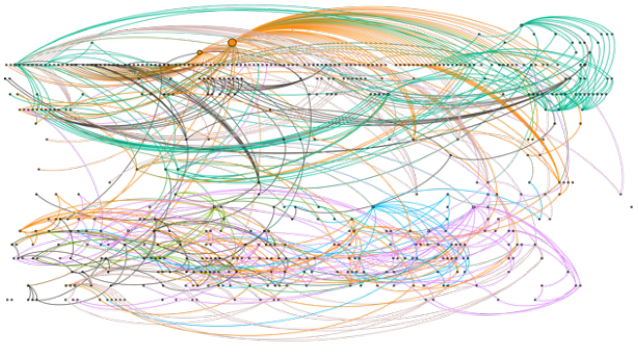
Accounting for Time

- Eliminate abstract wrappers (*e.g.*, StartThreadWrapper)
- Avoid double counting time for function and its sub-routines

3 LIMELIGHT+ OVERVIEW

Analysis pipeline for datacentre call graphs

- FDMAX: Partition graph into coarser layers
- STEAM: Cluster semantically similar functions
- HELP: Identify hotspots for optimization



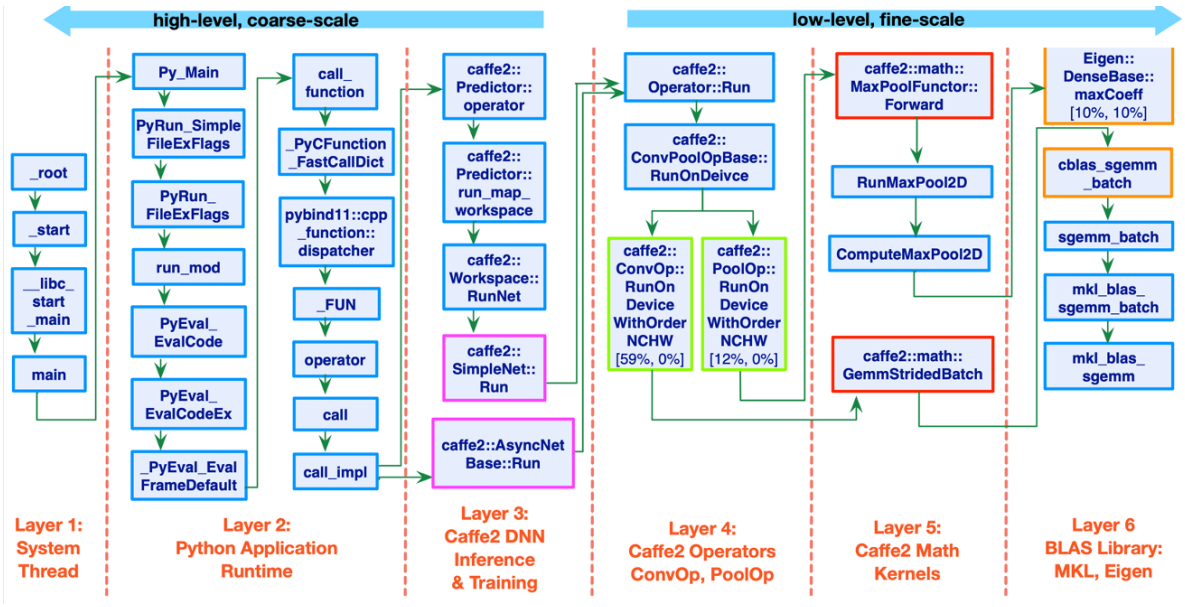
Case Studies

- **Testbed.** 400K stacks traces, 3.6K unique functions from PyTorch Caffe2, RocksDB, HHVM
- **Facebook.** 300M stack traces, 240K unique functions from production datacentre
- **DARPA.** Software assurance graphs from Automated Rapid Certification of Software (ARCOS)

4 PARTITIONING CALL GRAPHS

FDMAX layers the call graph by maximizing the foundational degree, which encodes desiderata.

- Functions in a layer compute at same scale
- Lower layers are foundations for higher layers
- Call graph is compacted to enhance interpretability



5 CLUSTERING SIMILAR FUNCTIONS

STEAM algorithm clusters semantically similar functions with machine learning.

- Use token co-occurence in function declarations to infer similarity
- Use token proximity in graph to infer similarity
- Discover synonymous functions to enhance interpretability

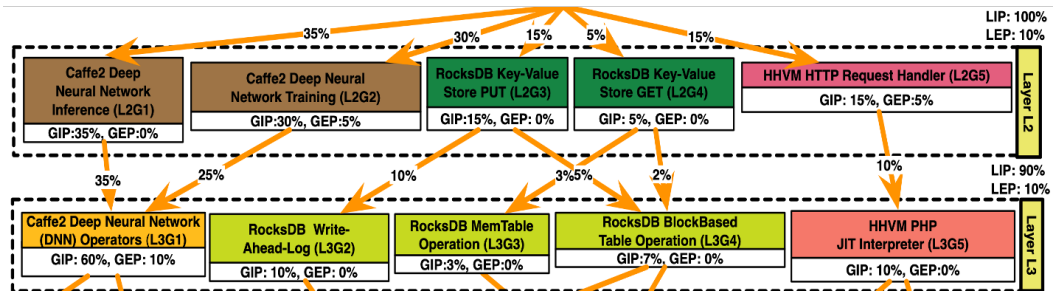
Hash/ Map	`std::_Hashtable::_M_assign, folly::hash::SpookyHashV2::Hash128, java.util.HashSet.add, MurmurHash64A`
(De) Compression	`qlz_decompress, LZ4_compress_fast, HUF_compress1X_usingCTable, snappy::InternalUncompress`

Tensor Operators	`caffe2::Tensor::Resize, caffe2::OperatorBase::Output, caffe2::Tensor::CopyFrom, caffe2::Tensor::dim32`
Math Functions	`__ieee754_exp_avx, __log2f_finite, __pow, __ceil_sse41, __cos_avx, __floor_sse41`

6 ATTRIBUTING COMPUTATION TIME

HELP algorithm maps information in original call graph into the transformed call graph with layers, clusters

- Exclusive Cycles: Cycles consumed by function when treated as leaf function.
- Inclusive Cycles: Cycles consumed by function and its sub-routines.
- Calculate group-, layer-level measures of cycles.



7 DATACENTRE INSIGHTS

Macro-scale Analysis

- Hottest 38 functions from 43 services account for only 60% of cycles
- Business critical services (ads, feeds, search); Microservices (machine learning, data stores, datacentre ops)

Micro-scale Analysis

- Hottest 37 clusters from O(10K) libraries account for only 50% of cycles
- No single dominant hotspot in software infrastructure

Exploiting Arm's Assembly data burst in Brain Machine Interface Applications

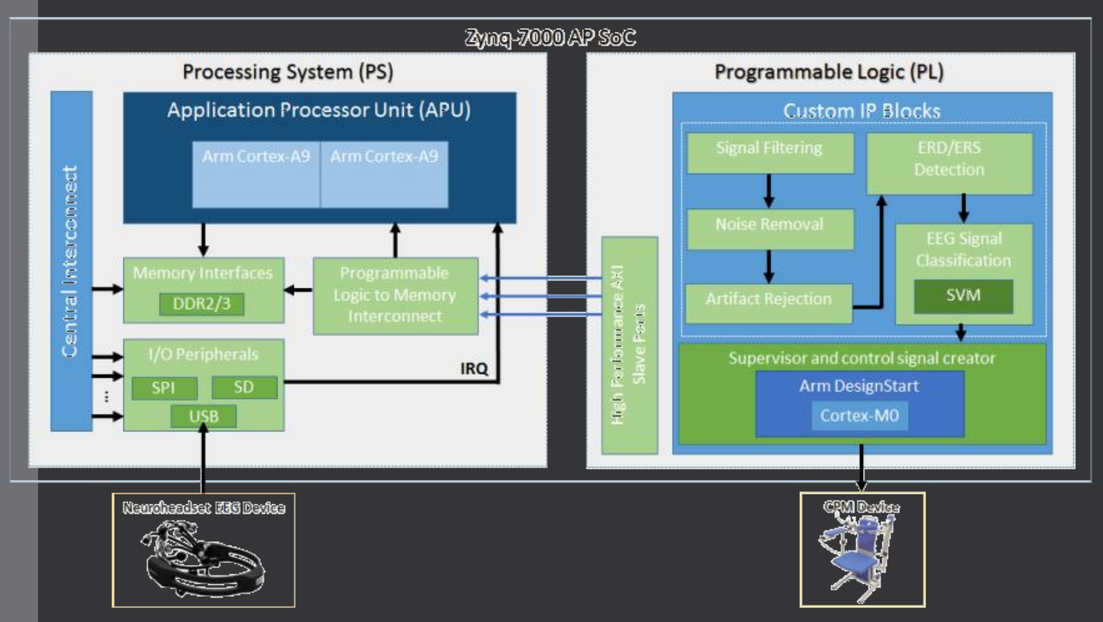
Gregory Kalogiannis (1); George Hassapis (1)
Organization(s): 1: Aristotle University of Thessaloniki, Department of Electrical and Computer Engineering, Greece
1: gkalogiannis@ece.auth.gr, 2: chasapis@eng.auth.gr

Introduction

Designing and prototyping a Brain-Computer Interface (BCI) using embedded systems is mainly an **integration process** where designers connect a set of **custom** Intellectual Properties (IP's) cores together using standard buses in order to build their custom systems [1]. **Such IPs are commonly modelling central aspects of a BCI system such as Electroencephalography (EEG) signal capturing, preprocessing, filtering, de-mixing, feature extraction, classification and mapping**

BCI System-On-Chips (SoC's), hosts most of the necessary computing components into a single chip. In previous works we have presented **an Arm-based multi-core heterogeneous SoC for Brain Machine Interfaces focused around the Zynq 7000 SoC.**

We have also presented several custom IPs hosted in the Programmable Logic (PL) that facilitated the previous BMI aspects. However, writing applications for such SoC's must be dictated by the BCI application needs while fast computation must be served in both hardware and software level.

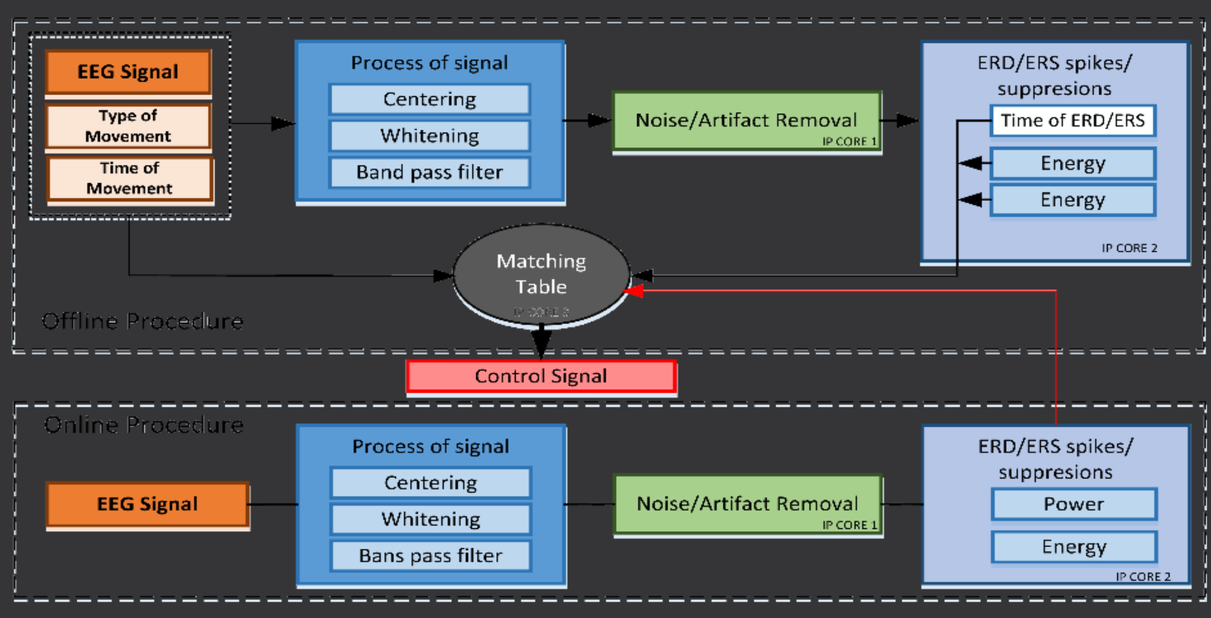


Our Study

The purpose of this study is to exploit on how to improve the performance of BCI SoC applications in software level by using Arm Assembly features such as the **data transfer burst**.

Therefore, we have chosen and accelerate the demanding algorithm of **Second Order Blind Identification (SOBI)** commonly used for signal separation in BCI systems that are dedicated in recognizing human motor imagery movements [2].

In our BCI system, imagery motor movements are recognized as mu and beta event-related desynchronization (ERD) and event-related synchronization (ERS) patterns inside EEG recordings by measuring power and energy signal features [3]. In order to do so, mixed signals arrived from EEG recordings must be separated into their original sources. **SOBI algorithm is responsible for such separation while it contains several matrix multiplications that can be speeded up using assembly burst-block data transfer commands.**



Burst matrix multiplication

The multiplication operations between the tables in the SOBI algorithm are the most time consuming piece. So in an effort to improve time execution, a function that performs multiplication of two tables with the help of the following burst data transfer commands of Arm assembly has been created.

LDM :Load Multiple registers
MLA : Multiply and accumulate

Such matrix multiplication that uses burst technique can be applied in order to transfer large volumes of data in several other cases during EEG process such as **any reversing array procedure, eigenvalues calculation or even inside the Centering / Whitening procedure** using, the singular value decomposition (SVD) algorithm. The latter is a factorization of a real or complex matrix that generalizes the Eigen decomposition of a square normal matrix to any m × n matrix via an extension of the polar decomposition.

Results

SOBI algorithm was executed and time measured on **STM32 Nucleo CortexM0** and **CortexM4** developing boards (STM32F042 and STM32F401 respectively) using **Arm Keil MDK**. The results revealed **a speed up of 10% to 15%** in all matrix sizes for both CortexM0 and CortexM4 when using the burst commands.

Another interesting aspect based on the results is that burst has better results when **increasing the matrix size**. Therefore, SOBI algorithm using burst commands may be used on large matrix sizes exploiting simultaneous transfer of large volumes of data which is typically on BCI applications.

	STM32 Nucleo CortexM0		STM32 Nucleo CortexM4	
Matrix size (nxn)	No Burst (sec)	With Burst (sec)	No Burst (sec)	With Burst (sec)
256	0.247	0.215	0.269	0.218
542	2.147	1.886	2.256	1.004
1024	31.032	27.122	35.034	26.256

References

[1] Cortex-M-based SoC Design and Prototyping using Arm DesignStart. Ashkan Tousi, Xabier Iturbe, Mirko Gagliardi, Grigorios Kalogiannis https://developer.arm.com/solutions/research/research-enablement-kits, 1-26
[2] Adel Belouchrani, Member, IEEE, Karim Abed-Meraim, Jean-Fran ¸ Cardoso, Member, IEEE and Eric Moulines, Member ,IEEE, "A Blind Source Separation Technique Using Second-Order Statistics", IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 45, NO. 2, FEBRUARY 1997
[3] J. Yongwoong, N. Chang S., K. Young-Joo and W. Cheol, "Event-related (De) synchronization (ERD/ERS) during motor imagery tasks: Implications for brain–computer interfaces," International Journal of Industrial Ergonomics, September 2011.

Contact Author(s)

GitHub

Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques

Jeremie Kim[1,2], Minesh Patel[1], A. Giray Yağlıkçı[1],
Hasan Hassan[1], Roknoddin Azizi[1], Lois Orosa[1], and Onur Mutlu[1,2]

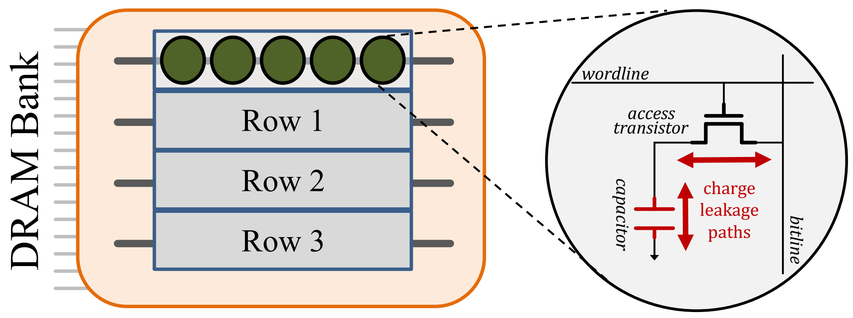
[1] ETH zürich [2] Carnegie Mellon

SAFARI

Full Paper
https://people.inf.ethz.ch/omutlu/pub/Revisiting-RowHammer_isca20.pdf

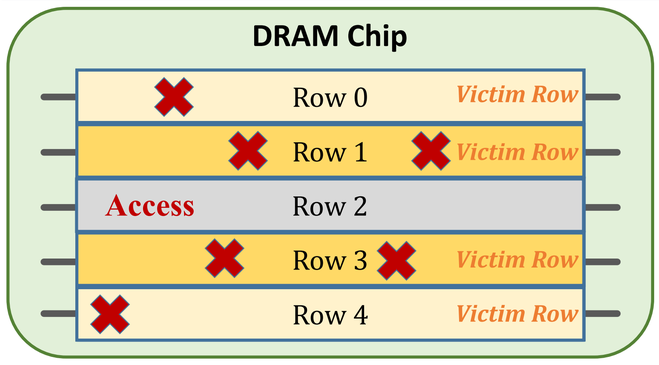
Full Talk Video
https://www.youtube.com/watch?v=Lqxc4_ToMUw

1: DRAM Background

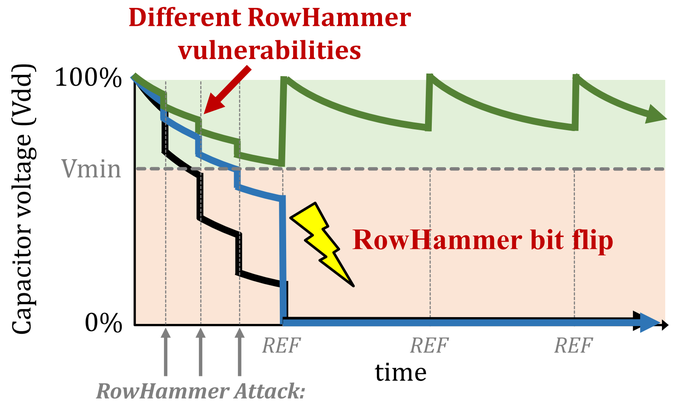


Stored data is **corrupted** if too much charge leaks (i.e., the capacitor voltage degrades too much)
DRAM cells are refreshed periodically to maintain data correctness

2: The RowHammer Phenomenon



Repeatedly **opening (ACT)** and **closing (PRE)** a DRAM row causes failures in nearby rows



If a nearby row is activated enough times within a refresh window, the charge leakage rate can be accelerated to the point of **failure.** Some cells require more hammers to fail.

3: Motivation

- Denser DRAM chips are **more vulnerable** to RowHammer
- **No comprehensive experimental study** demonstrating **how vulnerability scales** across DRAM types and tech node sizes
- **Unclear whether current mitigation mechanisms will remain viable** for future DRAM chips that are likely to be more vulnerable to RowHammer

4: Our Goal

1. **Experimentally demonstrate** how vulnerable modern DRAM chips are to RowHammer and **predict how this vulnerability will scale** going forward
2. Examine the viability of current mitigation mechanisms on **more vulnerable chips**

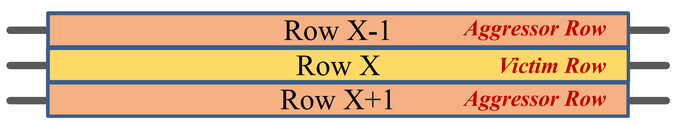
5: Experimental Methodology

1580 total chips tested from **300** modules

DRAM type-node	Number of Chips (Modules) Tested *Mfr. A*	*Mfr. B*	*Mfr. C*	*Total*
DDR3-old	56 (10)	88 (11)	28 (7)	**172 (28)**
DDR3-new	80 (10)	52 (9)	104 (13)	**236 (32)**
DDR4-old	112 (16)	24 (3)	128 (18)	**264 (37)**
DDR4-new	264 (43)	16 (2)	108 (28)	**388 (73)**
LPDDR4-1x	12 (3)	180 (45)	N/A	**192 (48)**
LPDDR4-1y	184 (46)	N/A	144 (36)	**328 (82)**

Experimental Testing Infrastructures
1. **DDR3:** SoftMC [Hassan+, HPCA'17] (Xilinx ML605)
2. **DDR4:** SoftMC [Hassan+, HPCA'17] (Xilinx Virtex UltraScale 95)
3. **LPDDR4:** In-house testing hardware

1. **Prevent sources of interference during core test loop**
We disable:
DRAM refresh: to avoid refreshing victim row
DRAM calibration events: to minimize variation in test timing
RowHammer mitigation mechanisms: to observe circuit-level effects
Test for **less than refresh window (32ms)** to avoid retention failures
2. **Worst-case access sequence**
- We use **worst-case** access sequence based on prior works' observations
- For each row, **repeatedly access the two directly physically-adjacent rows as fast as possible**

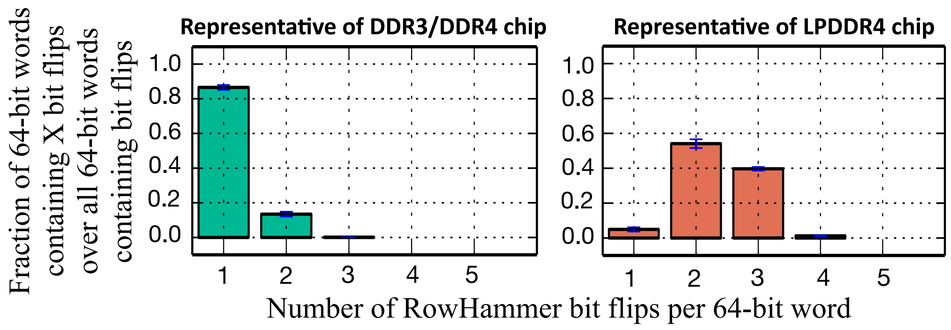


6: Characterization Results

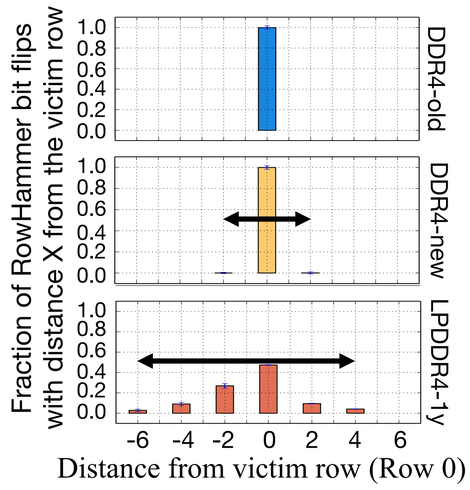
a) RowHammer Vulnerability
- **Newer DRAM chips are more vulnerable to RowHammer**

b) Data Pattern Dependence
- Worst-case data pattern is **same for chips** of same mfg. and type-node config.

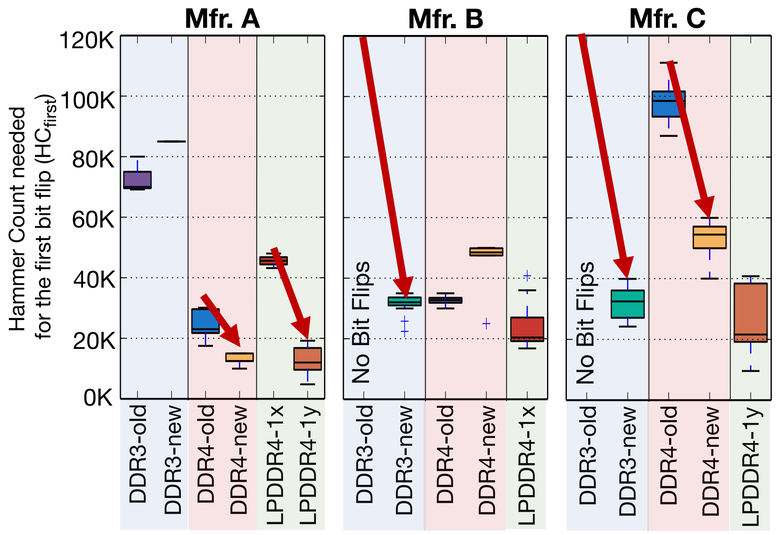
c) Hammer Count Effects
- **RowHammer bit flip rates increase with technology node generation**



d) Hammer Count Effects
- The distribution of RowHammer bit flip density per word **changes in LPDDR4 chips** from other DRAM types likely due to **on-die ECC**
- At a bit flip rate of 10^{-6}, a 64-bit word can contain up to **4 bit flips**. Even at this very low bit flip rate, a **very strong ECC** is required to prevent failures



e) Spatial Effects
- The number of RowHammer bit flips that occur in a given row decreases as the distance from the **victim row (row 0)** increases
- Chips of newer DRAM technology nodes can exhibit RowHammer bit flips 1) in **more rows** and 2) **farther away** from the victim row



f) First RowHammer Failure per Chip
- In a DRAM type, **HC_{first} reduces significantly** from old to new chips, i.e., **DDR3:** 69.2k to 22.4k, **DDR4:** 17.5k to 10k, **LPDDR4:** 16.8k to 4.8k
- In LPDDR4-1y chips from manufacturer A, there are chips whose weakest cells fail after only **4800 hammers**

7: Evaluation of Mitigation Mechanisms

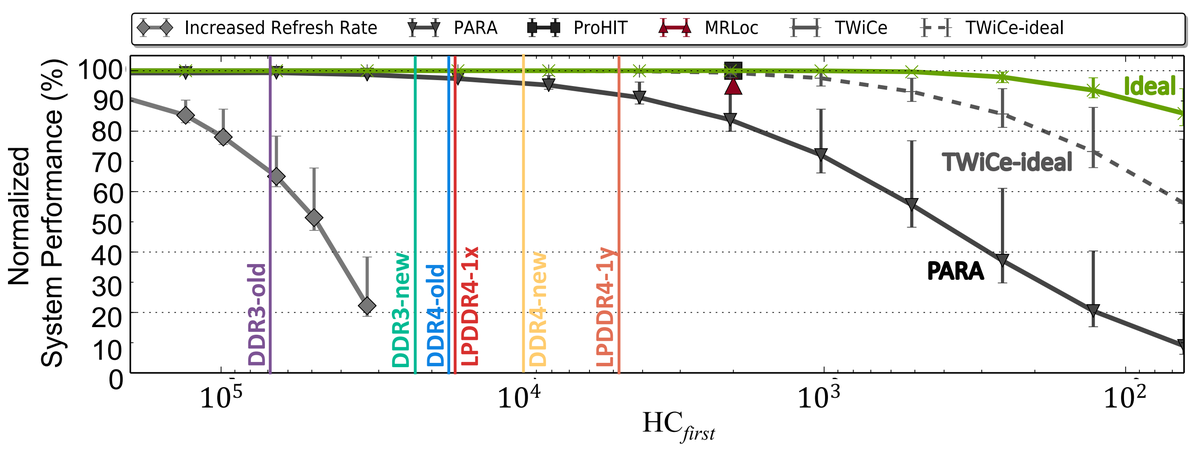
Increased Refresh Rate: Substantial overhead for high HC_{first} values. **Prohibitively high refresh rates** required for **HC_{first} < 32k.**

PARA: **Scales to low HC_{first} values**, but significantly high performance overheads **(e.g., 80% performance loss when HC_{first} = 128).**

ProHIT **MRLoc:** Models for scaling ProHIT and MRLoc for **HC_{first} < 2k** are not provided and how to do so is not intuitive.

TWiCe: **Does not support HC_{first} < 32k**, but we evaluate an ideal version ignoring two critical issues. Ideal performs better than PARA.

Ideal: ideal refresh-based mechanism provides reasonably high normalized system performance across all tested HC_{first} values.



Key Takeaways

1. **PARA, ProHIT, and MRLoc** mitigate RowHammer bit flips in **worst chips** with reasonable performance (92%, 100%, 100%)

2. **Only PARA scales to low HC_{first}** but has **low normalized system performance**

3. **Ideal** mechanism is **significantly better** than existing mechanisms for **HC_{first} < 1024**

4. **Significant opportunity** for developing a scalable and low overhead solution

8: Future Work and Conclusion

Future Research Directions

1. **DRAM-system cooperation**
- A DRAM-based or system-level mechanism alone ignores potential benefits of a holistic solution

2. **Profile-guided**
- Accurately profiling RowHammer-susceptible cells in DRAM provides a powerful substrate for building targeted RowHammer solutions e.g.:
  - **Increasing refresh rate:** increase refresh rate for rows with RowHammer-susceptible cell
  - **Access counters:** only count accesses to rows containing RowHammer-susceptible cells
- A **fast and accurate** profiling mechanism is a key research challenge for developing low-overhead and scalable RowHammer solutions

Conclusion: It is critical to research more effective solutions to RowHammer for future DRAM chips that will likely be even more vulnerable to RowHammer

FlexPool: A Distributed Model-Free Deep Reinforcement Learning Algorithm for Shared Passengers and Goods Delivery

Kaushik B. Manchella, Abhishek K. Umrawal, and Vaneet Aggarwal
Purdue University

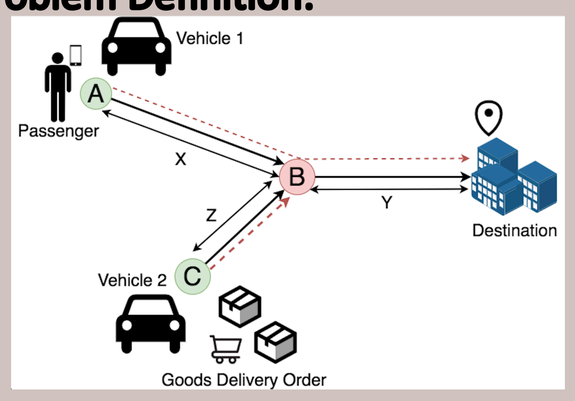
Cloud Computing, Machine Learning, And Networking Research (CLAN)

Arxiv Link: https://arxiv.org/pdf/2007.13699.pdf

Motivation:

- Stemming from the growth of E-Commerce, crowd-sourced delivery is also on the upsurge in adoption and demand. This includes last-mile delivery services (e.g. Amazon Flex), and urban delivery services (e.g. Postmates, DoorDash, Instacart, etc.)
- This points toward an increased load on the road transportation system which would have ramifications from both environmental sustainability and operational profitability standpoints.
- A joint transportation system for goods and passengers could address the aforemetioned problem, however all the existing methods proposed in research literature rely on an accurate pre-defined model of the urban environment.

Problem Definition:



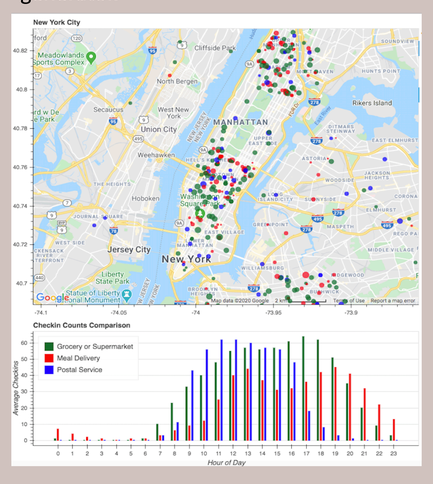
An illustration of multi-hop transportation scenario with hybrid delivery workload. Vehicle 1 and 2 are in zones A and C respectively. Zone B represents a "hop-zone" where the packages part of the goods order packages transfer over to another vehicle

- Our transportation system considers the combination of passenger ridesharing with Multi-hop goods delivery, where an existing ridesharing service is leveraged to "hitchhike" packages for their delivery.
- We develop a Model-Free algorithm that dispatches vehicles to locations of anticipated demand for passengers and goods.
- Given an objective as a reward function, the model-free algorithm learns the system dynamics through its interaction with the environment.
- Our proposed algorithm optimizes for the following objective:
  - Minimize the supply-demand gap for all requests
  - Minimize dispatch time for fleet vehicles
  - Minimize extra travel time incurred by requests due to vehicle sharing
  - Minimize number of hop transfers for a package/good
  - Minimize the number of fleet vehicles on the road

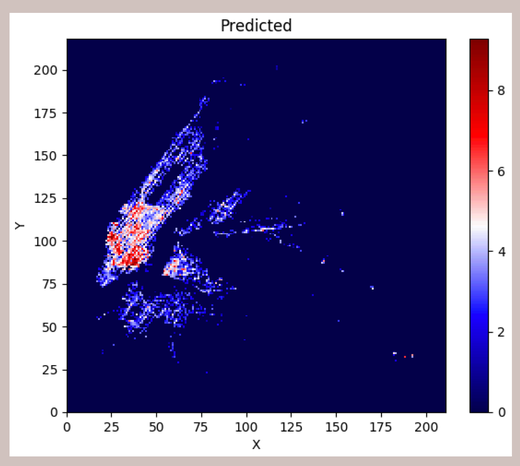
Methodology:

(1) Dataset

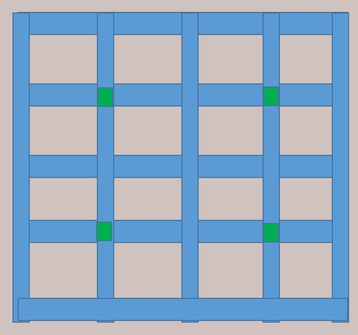
- To simulate passenger request, we use a real taxi trip dataset (NYC Taxi-Limousine commission-trip record data 2018). We specifically simulate the request time, location, and destination.
- For goods requests, we extract customer check-in traffic from Google Maps (see below) for all postal service, meal delivery and supermarket locations and build a request generator.



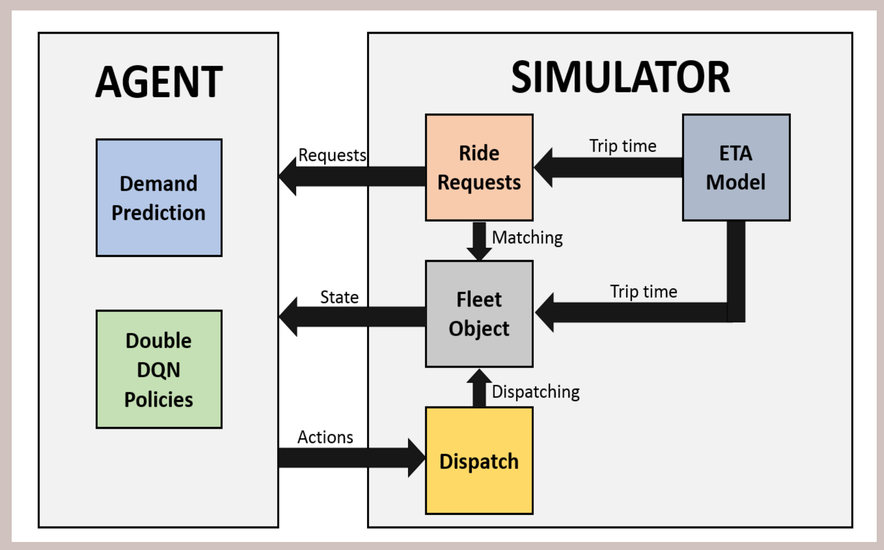
(2) Simulator Setup



New York city is divided into 212 x 219 grids. The passenger+goods pickup demand is anticipated over the next 30 minutes as seen above.



Schematic of the grids – package transfer can take place only at hop zones (in green)

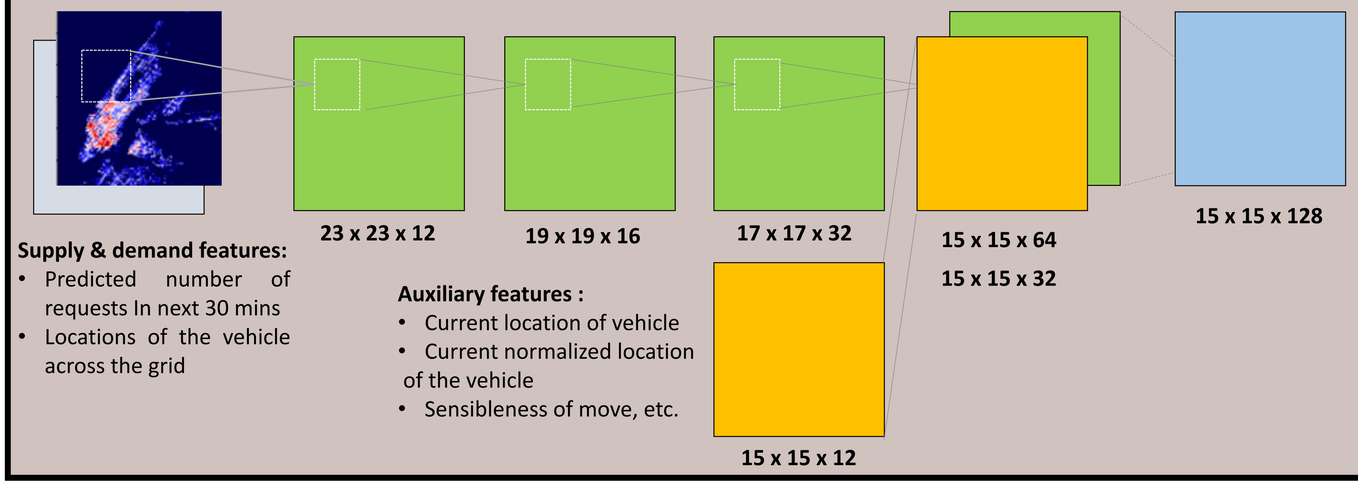


The simulator operates in this grid like environment – gives state to agent and gets actions as output

(3) FlexPool Algorithm

Inputs: Environment State Vector Ω_t, Pick-up Requests, Map-based locations.
Outputs: Decisions for Matching and Dispatching.
Create state vector $\Omega_{t,n} = (\mathbf{X}_t, \mathbf{V}_{t:t+T}, \mathbf{D}_{t:t+T})$
Initialize vehicle states X_0 as location of first N requests.
for $t \in T$ **do**
  Fetch all passenger ride requests at time t
  for each passenger ride request **do**
    Get passenger-vehicle assignments using greedy matching
    Update Vehicle seating capacity $C_{p,n}$
    Update the state vector $\Omega_{t,n}$
  Fetch all goods delivery requests at time t
  Assign hop-zones and extract trip segments using the hopzone assignment heuristic (see full paper).
  Update request backlog with hop-trips
  for each goods delivery request **do**
    Get package-vehicle assignments using greedy matching
    Update Vehicle trunk capacity $C_{k,n}$
    Update the state vector $\Omega_{t,n}$
  for each idle vehicle n **do**
    Input $\Omega_{t,n}$ to DDQN
    Get the optimal dispatch action a_t for each agent from the trained DDQN
  for all available vehicles $n \in a_t$ **do**
    Update $H_{t,n}$ if needed, and update vehicle location
    Find shortest path to each dispatch location for every vehicle n
    Estimate the travel time using the ETA model
    Update $\delta_{t,n}$ if needed, and generate vehicle trajectory n
  Update the state vector $\Omega_{t,n}$

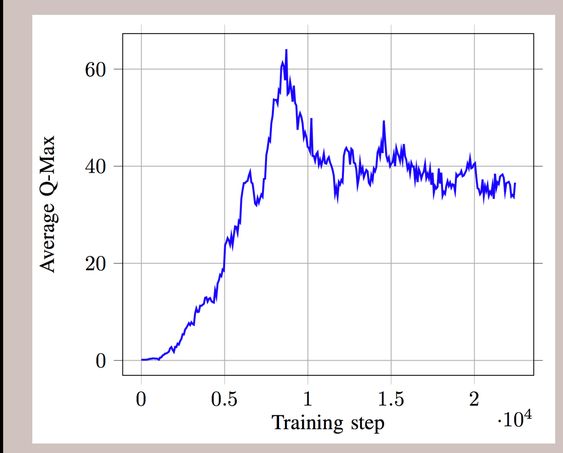
(4) Q-Network Architecture



Supply & demand features:
- Predicted number of requests In next 30 mins
- Locations of the vehicle across the grid

Auxiliary features :
- Current location of vehicle
- Current normalized location of the vehicle
- Sensibleness of move, etc.

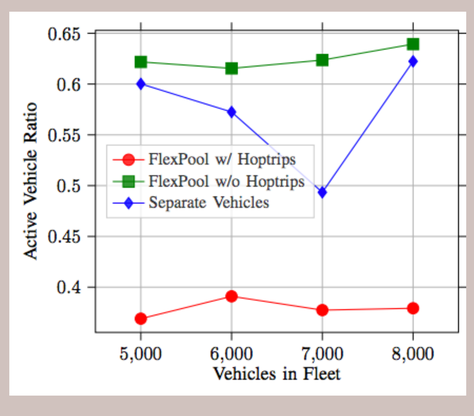
Simulation Results

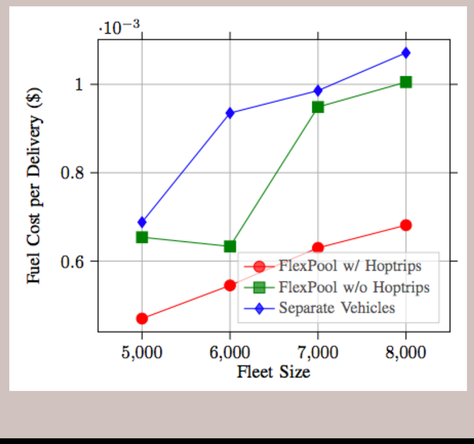


This plot shows the convergence of Q-values. During training. We see convergence in approximately 15,000 training steps. Q-values are representative of the reward function (the objective of our system). In optimizing for the Q-values, we optimize for the metrics mentioned in the problem definition.

Baselines:
- **FlexPool w/ Hoptrips:** This is our proposed algorithm where each vehicle is capable of serving both passengers and goods. The goods are routed using hop transfers.
- **FlexPool w/o Hoptrips:** In this baseline, we present the results when hop transfers are disabled from our proposed algorithm.
- **Separate Vehicles:** This baseline represents how majority of crowd-sourced transportation happens nowadays. Ridesharing and crowd-sourced package delivery is performed by completely separate fleet.

Evaluation Results:
- **Active Vehicles Ratio:** This metric indicates how many vehicles are currently on the road to serve the demand of requests. Our simulations show that FlexPool w/ Hoptrips performs approx. 30% better than the baselines. With fewer vehicles on the road, environmental sustainability is improved through minimizing congestion, emissions etc.
- **Fuel Cost per Delivery:** This metric computes the amount of fuel each vehicle expended normalized to the number of requests fulfilled. Our simulations show that FlexPool w/ Hoptrips performs approx. 35% better than the baselines. This points to an improved Operational profitability of the proposed method.





Conclusion

- We have proposed an efficient model-free algorithm where reinforcement learning techniques are used to learn the optimal decisions for each vehicle individually.
- We have observed improved vehicle utilization, vehicle fuel efficiency in the context of serving the combined demands of passengers and goods.
- As a future research direction, we recommend exploring the feasibility of hop transfers for goods with practical incentives.

References

1. A. Alabbasi, A. Ghosh, and V. Aggarwal, "DeepPool: Distributed model free algorithm for ride-sharing using deep reinforcement learning," IEEE Trans. Intelligent Transportation Systems (to appear), 2019.
2. T. Oda and C. Joe-Wong, "Movi: A model-free approach to dynamic fleet management," in IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018, pp. 2708–2716.

Testing Concurrency in Compilers with Téléchat

Luke Geeson and Jade Alglave

luke.geeson, jade.alglave,(@arm.com)

Abstract

Open source compilers are broken, and that means your programs are broken too.

Compiler Correctness must account for the memory model in the source and target program.

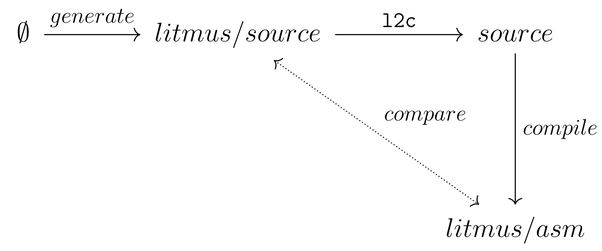
This isn't simple however as we must account for interpretations of language standards, the effects of optimizations, and requirements of the architecture.

We should not leave this work to interpretation, yet there is no tool that can automatically check correctness of C++ compilers with respect to memory models [2, 3, 4].

We propose the Téléchat tool to check whether compiler translation preserves concurrent program semantics over the source and target memory models.

Main Ideas

Téléchat is a translation validation tool that uses memory models to validate concurrent programs.



Takes small concurrent programs called litmus tests, compiles them, and checks whether the expected behaviour of the compiled program is a refinement of the source program behaviour with respect to the C/C++11 memory model and relaxed Arm Memory model.

Rely on the Herd [1] toolsuite for this, which forms the basis of the official Arm memory model [2, 3] and has models of the latest rc11 memory model for C/C++11 [4].

Can run assembly tests on hardware if we wish.

We can check whether:

- Compilers have introduced forbidden behaviour.
- Re-ordering or compiler optimization has led to invalid behaviour.
- Language semantics are violated, security properties.

Message Passing MP - C/C++11

initially `x=0` and `y=0`

Processor 0 (`P0`)	Processor 1 (`P1`)
(a): `x ← 1`	(c): `r0 ← y`
(b): `y ← 1`	(d): `r1 ← x`

Can `r0=1` and `r1=0` occur?

For C++ this is undefined behaviour if it happens.
For AArch64 without synchronization this is allowed!
How do we check that compilers have got it right?

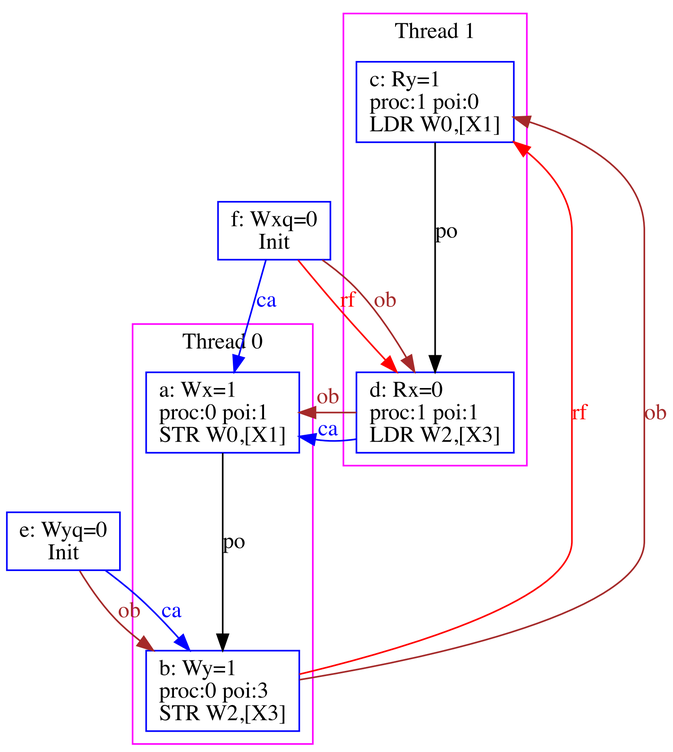
compiled to ⇓ or possibly ⟹

MP - AArch64

initially `P0:X1=x` and `P0:X3=y` and `P1:X1=y` and `P1:X3=x`

P0	P1
`MOV W0,#1`	`LDR W0,[X1]`
`STR W0,[X1]`	`LDR W2,[X3]`
`MOV W2,#1`	
`STR W2,[X3]`	

Can `P1:X0=1` and `P1:X2=0` occur? **Yes!**



Test MP, Generic(ARMv8 AArch64)

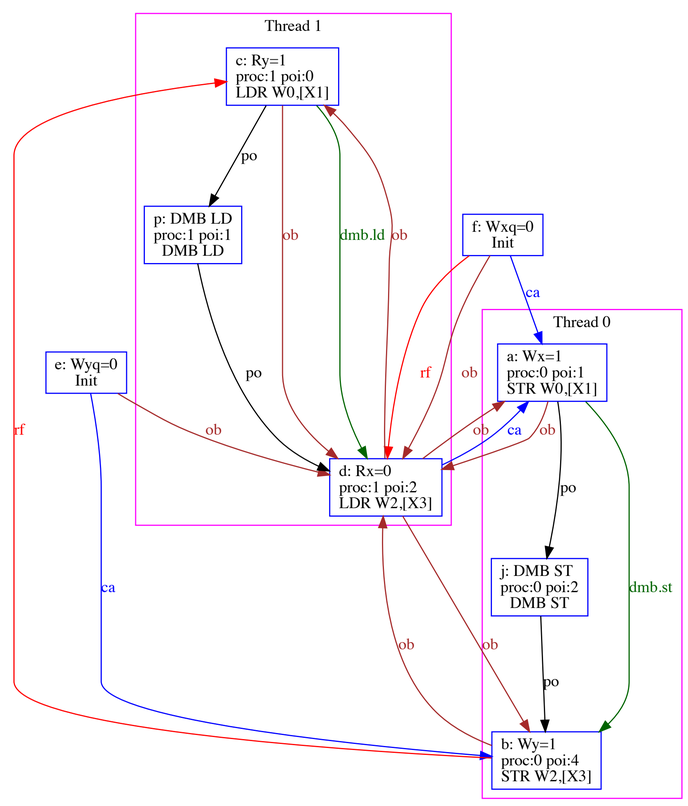
Allowed by the Arm architecture [2]
Runnable at:
`http://diy.inria.fr/www/?record=aarch64&litmus=MP`

MP+dmbst+dmbld - AArch64

initially `P0:X1=x` and `P0:X3=y` and `P1:X1=y` and `P1:X3=x`

P0	P1
`MOV W0,#1`	`LDR W0,[X1]`
`STR W0,[X1]`	`DMB LD`
`DMB ST`	`LDR W2,[X3]`
`MOV W2,#1`	
`STR W2,[X3]`	

Can `P1:X0=1` and `P1:X2=0` occur? **No!**



Test MP+DMB.ST+DMB.LD, Generic(ARMv8 AArch64)

Forbidden by the Arm architecture [2]
Runnable at:
`http://diy.inria.fr/www/?record=aarch64&litmus=MP+dmb.st+dmb.ld`

Acknowledgements

In no particular order:
Lee Smith, Alastair Reid, Luc Maranget, Gustavo Petri, Christof Douma, Pablo Barrio, Oliver Stannard, Peter Smith, The Arm Compiler & GCC Teams, Arm Architecture Technology Group, Arm Research Security Group.

References

[1] J. Alglave, L. Maranget, and M. Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.*, 36(2), July 2014.

[2] Arm. Arm architecture reference manual armv8, for armv8-a architecture profile, section b2.3 for the memory model. `https://developer.arm.com/documentation/ddi0487/latest`. accessed: 2020-08-06.

[3] Arm. Arm memory model tool. `https://developer.arm.com/architectures/cpu-architecture/a-profile/memory-model-tool`. accessed: 2020-08-06.

[4] ISO. Iso c++ standards. `https://isocpp.org/std/the-standard`. accessed: 2020-08-06.

arm

Mitigating Harmful Prefetches with Reinforcement Learning

Majid Jalili, and Mattan Erez

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

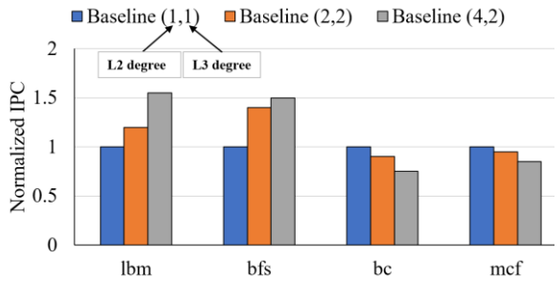
Single approach for providing fine-grain tunable prefetch throttling mechanisms based on the reinforcement-learning (RL)

Challenges

Prefetchers knobs

- Degree: number of prefetch requests to issue per trigger
- Prefetch distance: how far ahead of the demand access stream are the prefetch requests issued

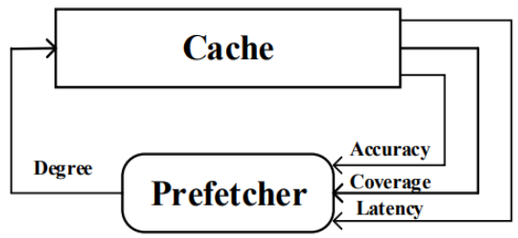
Impact of degree on IPC



We need many prefetchers with many knobs to configure

Solution

- We propose to design prefetcher throttler as an RL agent to learn automatically an optimal throttling policy via interaction with the rest of the system
- Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances
- Each state determines with a tuple <Accuracy, Coverage, Latency> with SARSA update



Degrees

States	0	1	2	3	4	5
000						
001						
010			Q			
011						
100						
101						Q
110						
111						

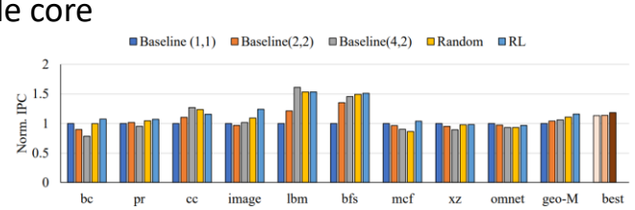
Methodology

Processor	Single-core 4.0GHz, ROB=192, fetch-width=4, LQ Entries = 32, SQEntries = 32 , fetchWidth= 8 Ubuntu 16.04 OS
L1 Cache	64KB, 8-way, MSHR 12 entries
L2 Cache	256KB, 4-way, MSHR 16 entries, tagged next line prefetcher, Queue size=32
L3 Cache	2MB, 16-way, MSHR 32 entries, Best Offset Prefetcher [17], Queue size=32
Main memory	A single DDR4-2400 x64 channel (one command and address bus), with timings based on a DDR4-2400 8 Gbit datasheet (Micron MT40A1G8) in an 8x8 configuration. Total channel capacity is 16GB

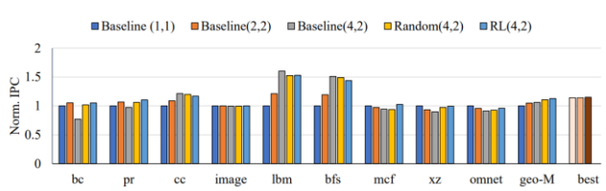
SPEC CPU 2017 and GAP benchmark suite , SimPoint [115] methodology, 500M warmup and 500M simulation

Results

Single core



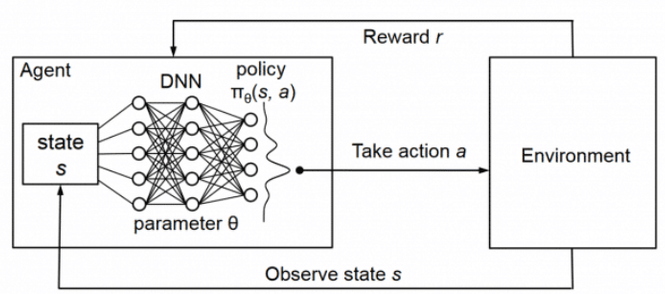
Multi-core



Future Work

Many prefetchers setting

Deep Q-learning with Experience Replay



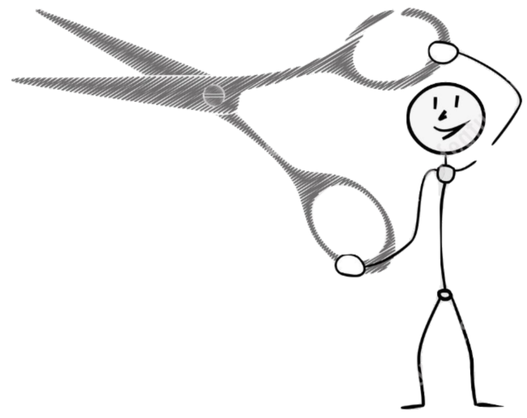
A Lightweight Reliability Enhancement Scheme for MLC STT-RAM based CNN Accelerators

Masoomeh Jasemi and Nader Bagherzadeh
University of California, Irvine

UCI Samueli School of Engineering
University of California, Irvine

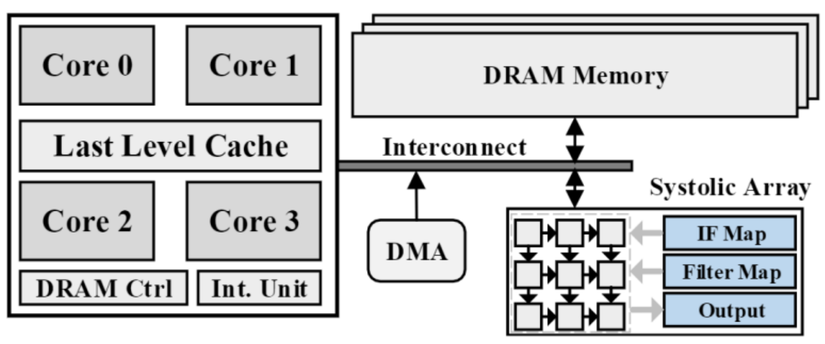
Reliability through data manipulation

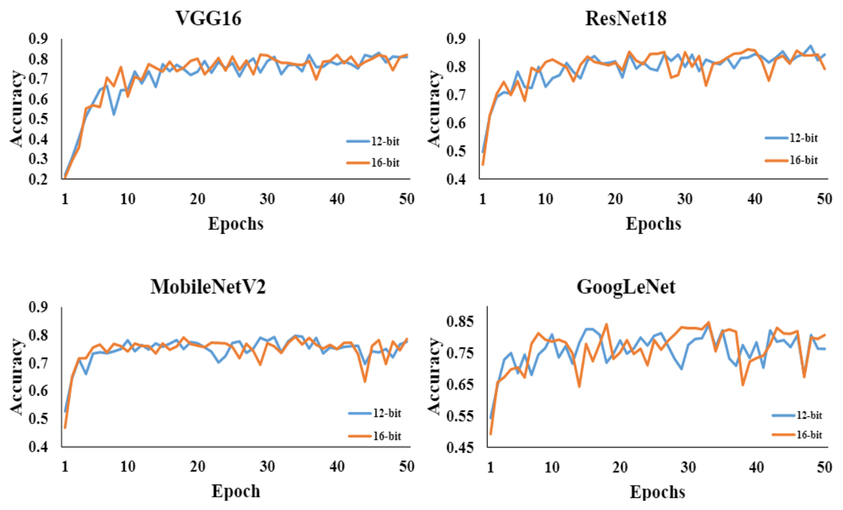
How Drop-and-Rearrange strategy can help to improve MLC STT-RAM reliability?

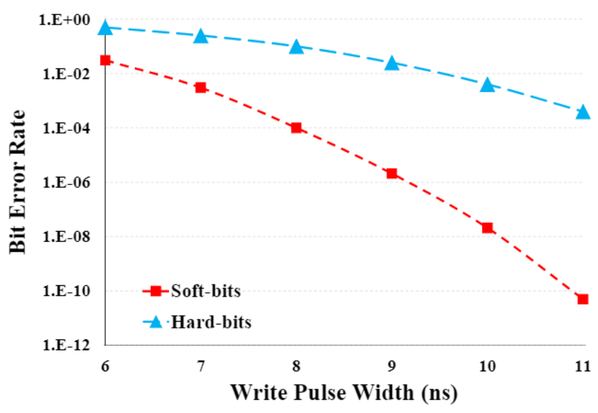


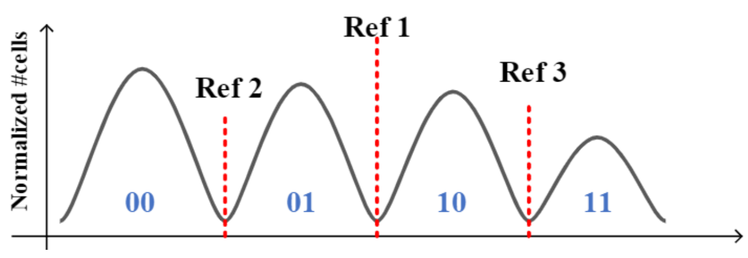
Problem and Motivations

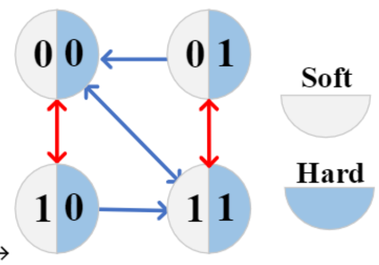
- MLC STT is large, but unreliable
- CNN is robust
- Asymmetric reliability of MLC STT-RAM











Solutions

1) Drop:

Drop=3

0 SLC	1 SLC	2 SLC	3	4	5	6	7	8	9	10	11	12

Drop=4

0 SLC	1 SLC	2 SLC	3 SLC	4	5	6	7	8	9	10	11

2) ReArrange:

DARA=3

0 SLC	1 SLC	2 SLC	3 Soft	8 Hard	4 Soft	9 Hard	5 Soft	10 Hard	6 Soft	11 Hard	7 Soft	12 Hard

DARA=4

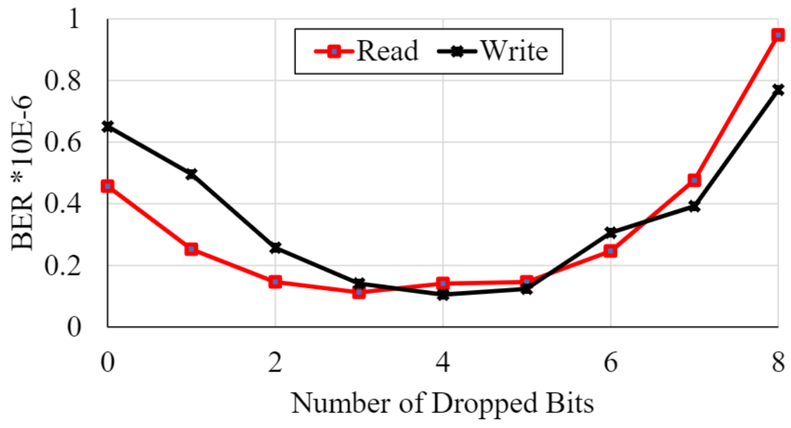
0 SLC	1 SLC	2 SLC	3 SLC	4 Soft	8 Hard	5 Soft	9 Hard	6 Soft	10 Hard	7 Soft	11 Hard

Analytical Model

$$Read_{Error} = \sum_{n=0}^{15} P \times penalty(i)$$

$$Write_{Error} = \sum_{n=0}^{15} HardSoft(i) \times penalty(i)$$

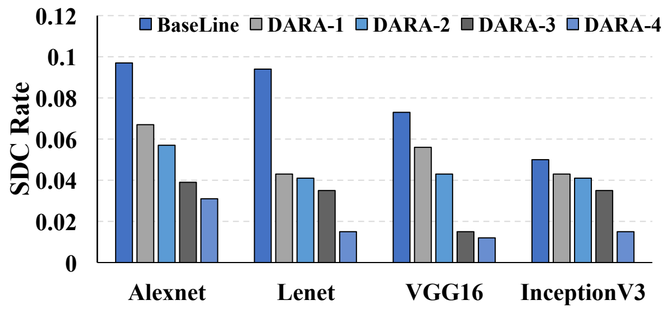
$$Read_{Error} = \sum_{n=0}^{D-1} penalty(i) + \sum_{n=D}^{15-D} P \times penalty(i)$$



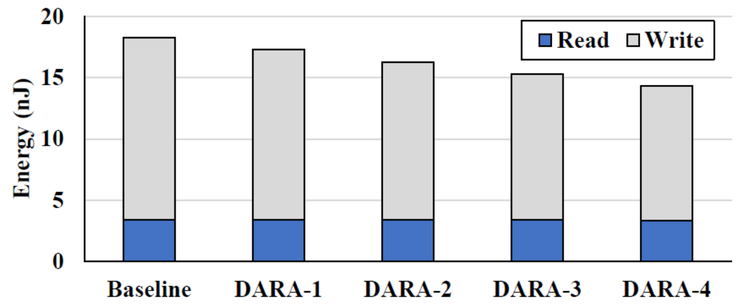
Methodology

- TensorFlow
- Graph
- TensorFI
- NVsim
- SCALE-Sim

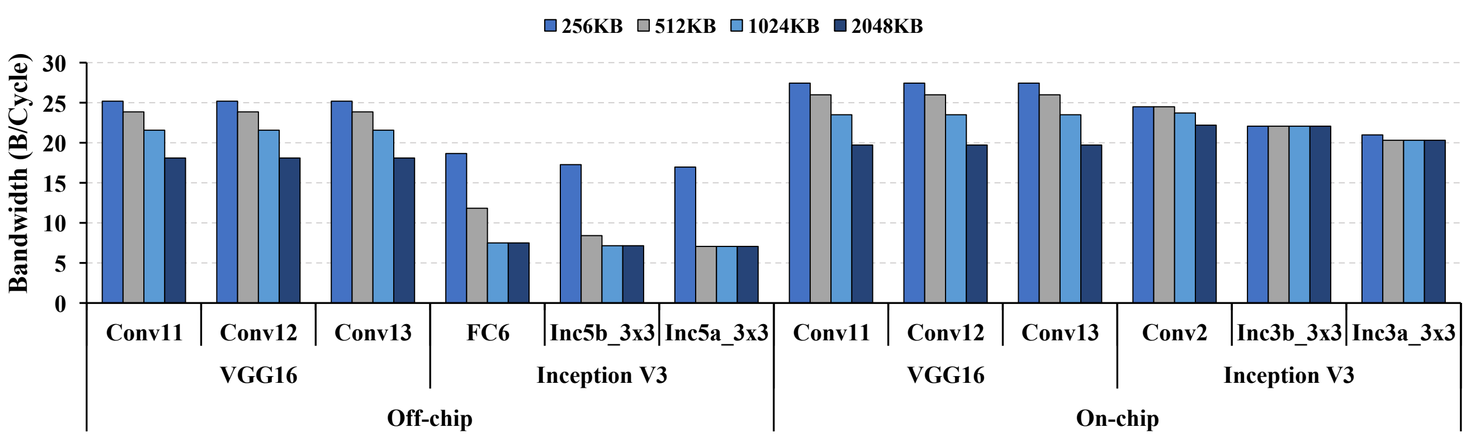
SDC



Energy



Bandwidth



Future Work

- Other memory technologies (ReRAM)
- Integer representation
- RNN and recommendation systems
- Compression and quantization

The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

SAFARI

Nastaran Hajinazar[1,2], Pratyush Patel[3], Minesh Patel[1], Konstantinos Kanellopoulos[1], Saugata Ghose[4], Rachata Ausavarungnirun[5], Geraldo F. Oliveira[1], Jonathan Appavoo[6], Vivek Seshadri[7], Onur Mutlu[1,4]

1 ETH zürich 2 SFU SIMON FRASER UNIVERSITY 3 UNIVERSITY of WASHINGTON 4 Carnegie Mellon 5 KMUTNB 6 BOSTON UNIVERSITY 7 Microsoft

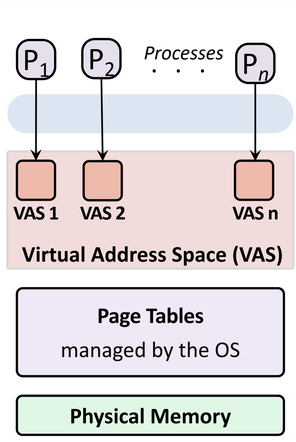
Full Paper
https://people.inf.ethz.ch/omutlu/pub/VBI-virtual-block-interface_isca20.pdf

Full Talk Video
https://www.youtube.com/watch?v=7c6LgVrCwPo

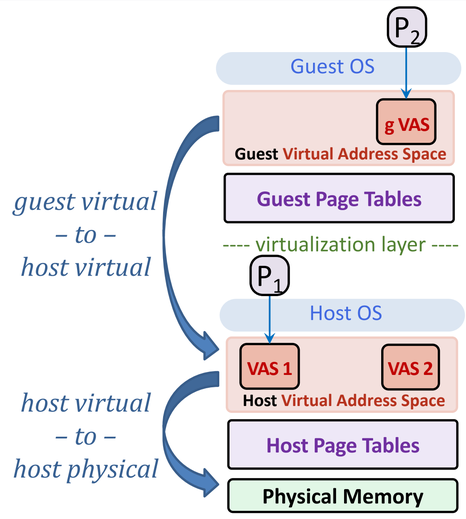
1: Motivation

- **Modern computing systems continue to diversify** with respect to system architecture, memory technologies, and applications' memory needs
- **Continually adapting the conventional virtual memory** framework to each possible system configuration is **challenging**

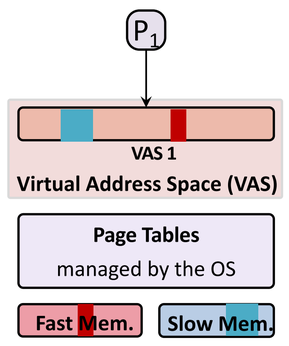
2: Example Challenges of the Conventional Virtual Memory Framework



1. **Page tables need to be shared between, and understood by both the hardware and the OS, resulting in rigid page table structures**
   - Challenging to implement the page table flexibility that applications can benefit from



2. **In virtual machines, both the guest and host OS perform address translation, resulting in an extra level of indirection**
   - Challenging to perform computation in virtualized environments, efficiently



3. **OS that defines and manages the address mapping, has low visibility into fine-grained runtime memory behavior information**
   - Challenging in heterogenous memories to make timely migration decisions based on quickly changing memory access patterns or other dynamic behavior

3: Our Goal

Design an alternative virtual memory framework that

- **Efficiently and flexibly supports increasingly diverse system configurations**
- **Provides the key features of conventional virtual memory framework while eliminating its key inefficiencies**

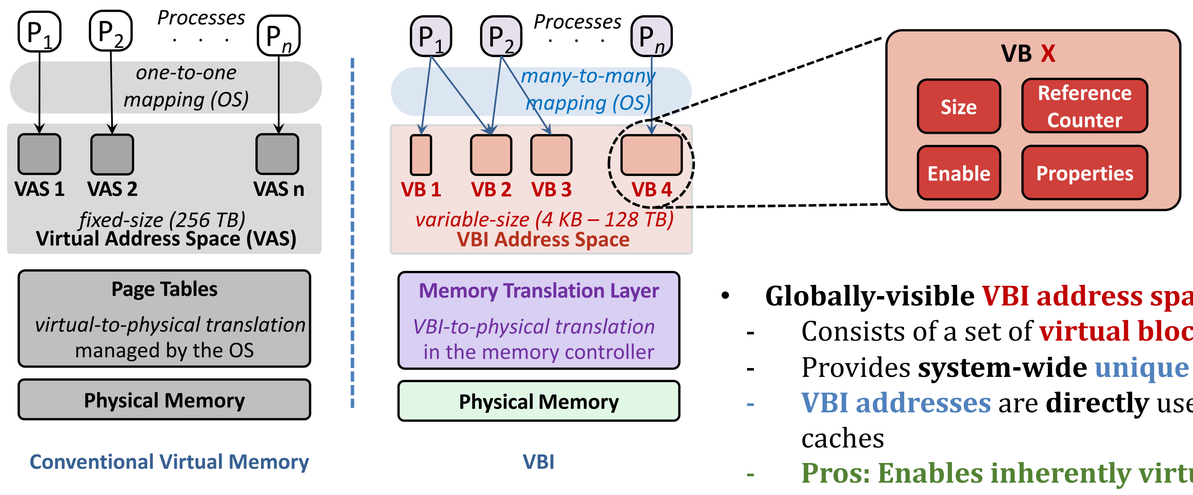
4: Virtual Block Interface (VBI)

Key idea:
Delegate physical memory management to dedicated hardware in the memory controller

Guiding Principles:

1. **Size virtual address spaces appropriately for processes**
   - Mitigates translation overheads of unnecessarily large address spaces
2. **Decouple address translation from access protection**
   - Defers address translation until necessary to access memory
   - Enables the flexibility of managing translation and protection using separate structures
3. **Communicate data semantics to the hardware**
   - Enables intelligent resource management

5: VBI Design Overview



Achieving the guiding principles:

1. **A process' VBs define its address space**, i.e., determined by the **actual needs of the process**
2. **Address mapping** is dedicated to the **MTL**, while **OS** retains full control over **managing the access permissions**
3. **Each VB is associated** with a set of information:
   - A System-wide unique ID
   - Size of the VB
   - Enable bit
   - Reference counter: number of processes attached to the VB
   - Properties bit vector: semantic information about VB contents, such as access pattern, latency sensitive vs. bandwidth sensitive

- **Globally-visible VBI address space**
  - Consists of a set of **virtual blocks** (VBs) of different sizes
  - Provides **system-wide unique VBI addresses**
  - **VBI addresses** are **directly** used to access on-chip caches
  - **Pros: Enables inherently virtual caches**
- **All VBs are visible to all processes**
  - **OS** controls which processes access which VBs
  - Each process has **its own permissions** (read/write/execute) when **attaching** to a VB
  - OS maintains a list of **VBs attached to each process** used to perform permission checks
- **Processes map each semantically meaningful unit of information to a separate VB**
  - e.g., a data structure, a shared library
- **Memory management is delegated to the Memory Translation Layer (MTL) in the memory controller**
  - Address translation and Physical memory allocation
  - Translation structures are **not shared** with the OS
  - **Per-VB translation structure** tuned to the VB's characteristics
  - **Pros: many benefits, including**
    - **Address translation overhead** for the processes running inside a **virtual machine**, is **no different** than the processes running **natively on system**
    - Enabling **flexible translation structures**

6: Optimizations Enabled by VBI

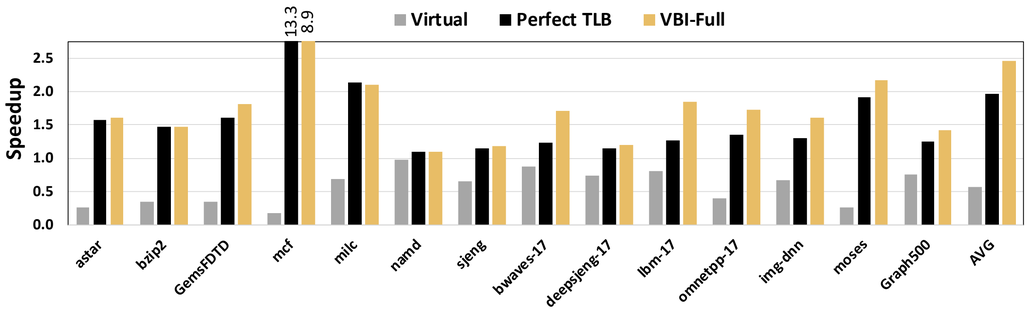
Naturally enabled by VBI and not easily attainable before:

Inherent to VBI design
- Appropriately sized process address space
- Flexible address translation structures
- Communicating data semantics to the hardware
- Inherently virtual caches
- Eliminating 2D page walks in virtual machines

Covered in the paper
- Delayed physical memory allocation
- Early memory reservation mechanism

7: Example Use Case: Address Translation

- **Native:** applications run natively on an x86-64 system
- **Virtual:** applications run inside a virtual machine (accelerated using 2D page walk cache [Bhargava+, ASPLOS'08])
- **Perfect TLB:** an unrealistic version of Native with no translation overhead
- **VBI-Full:** VBI with all the optimizations that it enables



Key Takeaways

1. **VBI-full** improves the performance, by 2.4X on average compared to **Native**

2. **VBI-full** enables **significant performance improvement** in virtualized environments (4.3X on average compared to **Virtual**)

3. **VBI-full** outperforms **Perfect TLB** (by 49% on average) as the **optimizations that VBI enables are not limited to only reducing the address translation overhead**

8: Conclusion

VBI is a promising new virtual memory framework

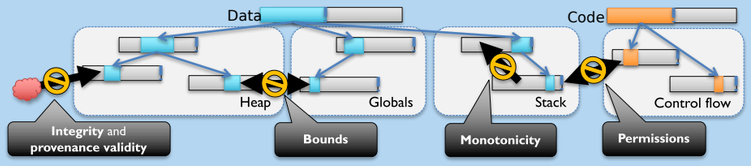
- **Can enable several important optimizations**
- **Increases design flexibility for virtual memory**
- **A new direction for future work in novel virtual memory frameworks**

Temporal Safety for CHERI Heaps

Nathaniel Wesley Filardo[†], Brett F. Gutstein[*], Jonathan Woodruff[*], Sam Ainsworth[*], Lucian Paul-Trifu[*], Brooks Davis[‡], Hongyan Xia[*], Edward Tomasz Napierala[*], Alexander Richardson[*], John Baldwin[§], David Chisnall[†], Jessica Clarke[*], Khilan Gudka[*], Alexandre Joannou[*], A. Theodore Markettos[*], Alfredo Mazzinghi[*], Robert M. Norton[*], Michael Roe[*], Peter Sewell[*], Stacey Son[*], Timothy M. Jones[*], Simon W. Moore[*], Peter G. Neumann[‡], Robert N. M. Watson[*]

[†]Microsoft Research; [*]University of Cambridge; [‡]SRI International; [§]Ararat River Consulting

Spatial Safety with CHERI and CheriABI



CHERI introduces architectural *memory capabilities*: unforgeable bearer tokens pairing a traditional pointer with bounds and permissions. Attempting to use a capability to access out of bounds memory or beyond its permissions instead raises an architectural trap. In "pure capability" C/C++ atop **CheriABI**, all language and runtime pointers are lowered to capabilities and capability bounds are applied by the compiled code, runtime library, and kernel. Notably, `malloc()` bounds each returned pointer to grant access only to one (padded) allocation.

Use After Free and Use After Reallocation

"Use After Free" flaws occur when software continues to use a reference to an allocation whose lifetime has ended. Such uses risk *aliasing* with new objects occupying *reused* memory ("Use After Reallocation"): new allocations proper or allocator internal metadata. These aliased accesses leak information and/or corrupt program state and are widely used components of exploit chains.

These make up **24%** of Microsoft CVEs 2006–2018! (Matt Miller, BlueHat IL, 2019)

Temporal Safety Through Sweeping Revocation

We aim to deterministically mitigate use-after-reallocation vulnerabilities involving heap pointers, causing *architectural traps* on the use of stale references that have come to refer to regions of memory holding new objects. We will accomplish this by **revocation**, replacing these stale references with invalid ones, eliciting a trap on any attempted subsequent use, before any reuse of `free()`-d memory.

Capabilities to heap allocations may be stored in heap, stack, or global memory, register files, or even the kernel. Revocation must scan all of these locations, precisely identify capabilities, and ensure that the application cannot retain a stale reference across the end of a scan.

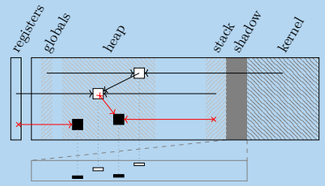
Quarantine

Revoking on every `free()` would be prohibitively expensive, but revocation must be performed to ensure that no surviving references exist to `free()`-d memory. Instead, we accumulate `free()` memory, without any reuse, in a **quarantine**. Only once quarantine is sufficiently large do we trigger revocation, reclaiming many `free()` objects at once.

Pages in quarantine can have their physical memory released back to the system prior to the *virtual* address being revoked and made reusable. These released pages have modest associated costs, and so the quarantine need only grow with fragmentation, not total `free()` throughput.

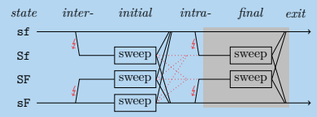
Cornucopia: An End-to-End Implementation in CheriBSD

We have implemented revocation-based heap temporal safety in CheriBSD for CheriABI programs. Changes were confined to the kernel and runtime libraries; application code already ported to CheriABI needed no further work.



Cornucopia adds an in-kernel revocation service, which finds and removes dangling pointers (×) when requested by the application. Userspace indicates quarantine through a **shadow bitmap** which covers all of user memory. The revoker is a shared resource, and multiple allocators can safely share revocation work without direct communication.

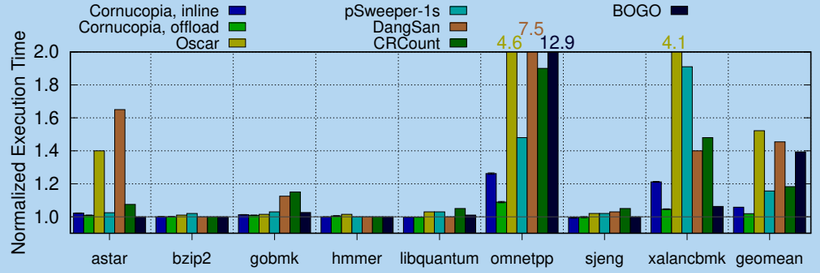
Revocation is *mostly concurrent* with the application. A first, background, pass sweeps all pages holding capabilities; a second pass, with application threads paused, re-sweeps pages written to since the start of the first pass (as well as capabilities held in registers and the kernel) before resuming the application.



Morello has all the architectural mechanisms required for Cornucopia.

Cornucopia Runtime Overheads

Cornucopia compares favorably against prior temporal safety systems:

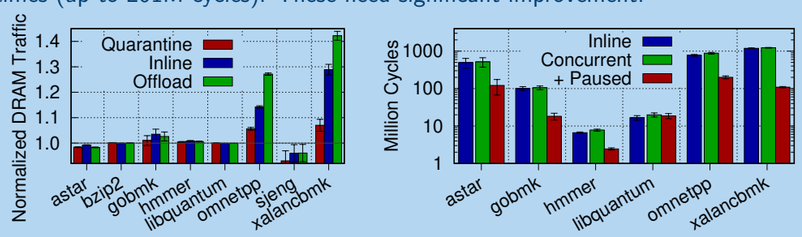


	Geo. Mean	Worst Seen
Inline	5.8%	26.2%
Offload	**1.9%**	8.9%

Average cycle overhead is under 2% on CHERI-compatible SPEC2006 benchmarks when equipped with a second core for background scanning (and excluding the cost of zeroing `free()` memory before reuse, ≈ 3% overhead).

Cornucopia DRAM Access and Pause Times

Cornucopia induces substantial DRAM traffic overheads (up to 43%) and pause times (up to 201M cycles). These need significant improvement.



Concurrency reduces pause times but at the cost of increased DRAM traffic, which directly translates to increased power use.

Looking Ahead: Reducing Costs of Revocation

Load Barriers and Page Generations Stopping application threads during revocation is unfortunate, but is required in Cornucopia to ensure that the revoker catches up to the application. Instead, we can *revoke read access* from pages at the beginning of revocation and allow the application to read only pages that we have cleaned. Page faults guide us to pages needed urgently, which can be cleaned as needed. Background scans ensure that eventually all pages are cleaned again. **Morello** features a fast mechanism for revoking access to all capability-bearing pages without needing to update all page table entries. Excitingly, unlike Cornucopia, this mechanism *scans each page once per revocation* and should have commensurately lower DRAM traffic.

CHERI+MTE ARMv8.5-MemTag (MTE) looks to be able to, when suitably combined with CHERI, deterministically enhance temporal safety. `free()` can immediately reuse a granule of memory by incrementing its MTE tag, until all have been used, at which point, the memory must be quarantined. MTE tags obviate Cornucopia's shadow bitmap: the revoker prunes all capabilities with mismatched tags. The potential benefits include significant reduction in quarantine growth, making revocations multiplicatively less frequent, and prompt reuse of memory, which simplifies system software, improves cache utilization, and decreases heap fragmentation.

Conclusion

Always-on temporal memory safety for heaps in CheriABI programs looks to be nearing practicality. CHERI, despite having no overt support for temporal safety or capability revocation, nevertheless provides an excellent architectural substrate for software-managed revocation.
New architectural mechanisms proposed by the CHERI team at Cambridge and soon to be available in **Morello** and other prototypes, in combination with ongoing engineering effort, should let us lower the runtime overheads, DRAM traffic costs, and pause times of revocation significantly.

Further Reading

- CHERI Project Homepage: `http://cheri-cpu.org`
- *CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment* (ASPLOS 2019): `http://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201904-asplos-cheriabi.pdf`
- *Cornucopia: Temporal Safety for CHERI Heaps* (IEEE S&P 2020): `http://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/2020oakland-cornucopia.pdf`

Acknowledgements

This work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ("CTSRD") and HR0011-18-C-0016 ("ECATS"). The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. We also acknowledge the EPSRC REMS Programme Grant (EP/K008528/1), the ABP Grant (EP/P020011/1), the ERC ELVER Advanced Grant (789108), the Gates Cambridge Trust, Arm Limited, HP Enterprise, and Google, Inc. Approved for Public Release, Distribution Unlimited.

Microsoft · SRI International · University of Cambridge

Powering the Internet of Things: Alternative form factors for Energy Storage

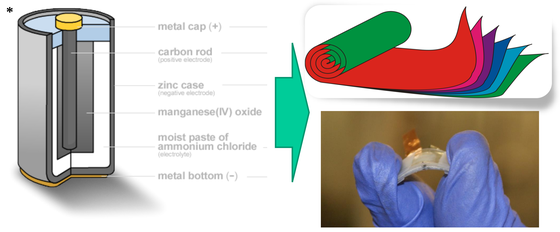
Pritesh Hiralal, Dilek Ozgit Butler, Karolina Spalek, Gehan Amaratunga

[a] Zinergy UK Ltd., Unit 2, Trinity Hall Industrial Estate, CB4 1TG, Cambridge, UK

*pritesh@zinergy-power.com

The Phone..... Its Energy

1 Flexible Printed Batteries



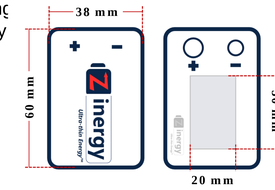
The Initial step was to turn a classic Alkaline battery into a flexible device

Hiralal et al. ACS Nano, 4. 2730-2734, 2010

This was achieved by tailoring the material in each layer of the battery into a flexible form factor. E.g. by using carbon fibre and solid electrolytes.

The printed battery

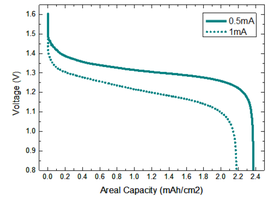
Zinergy's MK1Series is the first of a new generation of ultra-thin flexible battery products. Made by printing thin layers of functional materials onto a custom designed flexible packaging. These zincs based primary cells are suitable for cost effective low power electronic applications where safety and environmental credentials are fundamental.



Ozgit et al. ACS Appl. Mater & Inter. 2014

Specifications	
Battery Type	Zinc Flexible
Chemical System	Zinc – Manganese Dioxide (Zn/MnO2)
Continuous Drain Current	0.5 mA
Max. Continuous Drain Current	2 mA
Maximum Peak Current *	10 mA
Operating Temperature	-10°C to 50°C
Exterior Casing	Polymer Laminate
Environmentally Friendly	RoHS compliant
Nominal Voltage	1.5 V
Capacity @ 0.5 mA CC	70 Ω
Height (mm)	58
Length (mm)	38
Thickness (mm)	0.5
Maximum weight	1.2 g

* Pulsing conditions: 100 ms pulse, 2s rest, 1V cut-off



Printed battery discharge curve

Commercial Product

The Shirt..... Its Energy

2 Direct Printing on device

Why Direct Print?

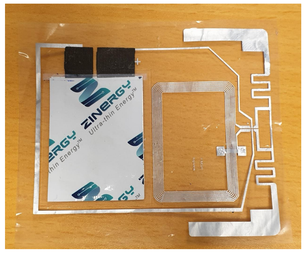
The ability to print directly onto a device, for instance a smart label or tag has several advantages including:

1/reduced production cost

2/improved contact reliability and

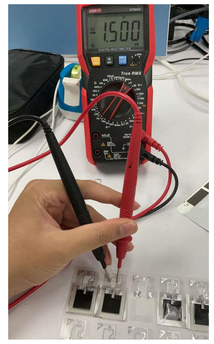
3/ increased production speeds.

This capability combined with a roll to roll print press opens the avenue for huge volumes



Battery printed directly onto RFID tag, allowing for significant cost reduction in bonding costs.

A typical roll on which Smart tags are produced, which can be used for direct printing



Sheet of batteries printed onto Smart tags.

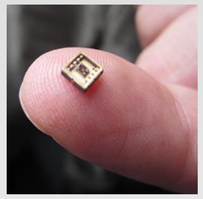
Large-Area Electronics

Large-Area Electronics (LAE), including printed, plastic, organic and flexible electronics, is a new way of making electronics that:

- is enabled by new materials that can be processed at low-temperatures;
- enables the use of new manufacturing processes for electronics such as printing and digital fabrication;
- enables products having new form factors, new cost structures and the potential for customisation

Pre-production

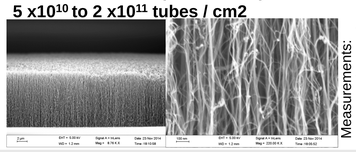
The sensor..... Its Energy



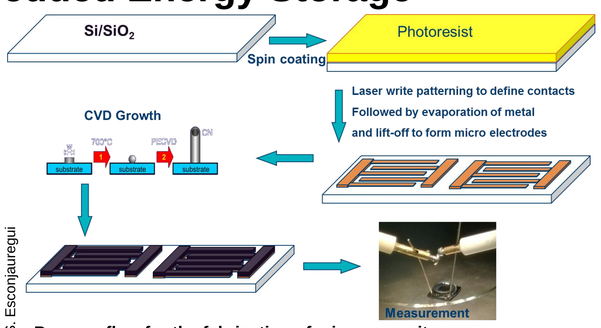
3 Microchip Embedded Energy Storage

Our approach – To use vertically oriented carbon nanotubes which can be lithographically patterned and grown directly onto Si/SiO_2 chips by chemical vapour deposition. Interdigitated electrodes form an ideal, low internal resistance structure for surface micro-capacitors.

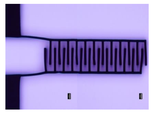
CNT Forest Density – Currently 5 x10^{10} to 2 x10^{11} tubes / cm2



Measurements: S. Esconjauregui

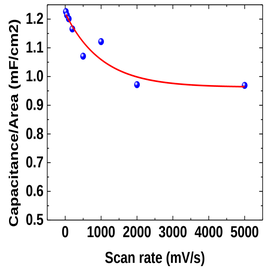


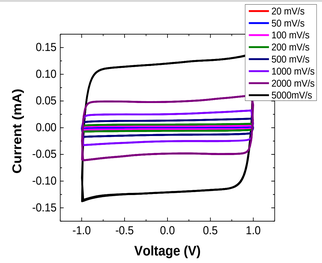
Process flow for the fabrication of micro-capacitors



Lithographically patterned high aspect ratio electrodes mean high power densities can be achieved

Electrode witdh ~50µm
Electrode height ~50µm





Current electrodes achieve an areal capacitance of ~1mF/cm^2 at scan rates up to 100 V/s. Capacitance can be correlated to the density of carbon nanotube forests, meaning that two orders of magnitude higher should be achievable by fine-tuning growth processes.

Experimental

ELECTRONICS, POWER AND ENERGY CONVERSION GROUP

B V RAJU INSTITUTE OF TECHNOLOGY, Narsapur, Telangana, India
CroPillar - ARM Research Summit- 2020

ABSTRACT

Food is a constant need for any living being, THIS is an established truth. With the increasing population today, the demand for food is at an all-time high. Usage of land is also increasing in order to satisfy the growing needs of the growing population. For this purpose, the land previously used for agricultural activities is now being used for other activities.

●In agricultural sector, the amount of water wastage is very high in the irrigation system and this is one of the prime reasons for weed formation and plant diseases. So to eradicate this problem a system of hydroponics was introduced which is a no-soil farming concept, but the nutrients might not be given at full potential in that system and usage of water is also high, and only giving a constant water supply would lead to the plant disease.

●The usage of water resources Is very high in agriculture sector , in which a lot of water used is left stagnant, resulting the formation of weeds in the field.

●Here comes the requirement for a farming solution, which gives a better yield, uses minimal water supply, utilizes lesser space and more importantly a much healthier option.

INNOVATION

Our design model is a octagonal shaped , each of the face consists of sampling s for planting . So in our design the roots are planted in the area and then the water ,required minerals are given directly to the roots of the plant.

The circular spaces in device is where the plant samplings are placed ,But unlike that, our proposed model shall be used to solve the flaws in hydroponics. Also our design is compatible in both indoor and outdoor farming .

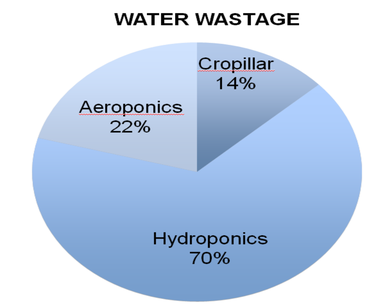
Also in a sudden floods scenario the entire yield can be stored at safer place without effecting the crop .

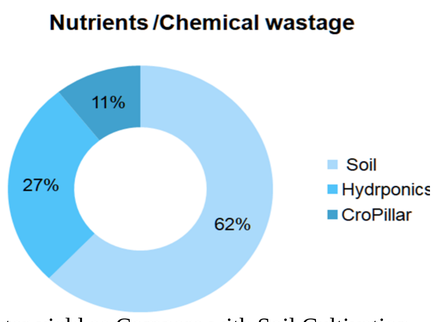
Our model shall provide the required water, nutrients directly to the roots of the plant. This proposed model would minimize the water intake by plants in agriculture. Also, the major effecting factor in agriculture sector is weed formation and diseases.

So our solution will be giving the required nutrients and water supply directly to the roots of plants.

The entire process is monitored using certain sensors; the data is updated through the cloud on regular basis.

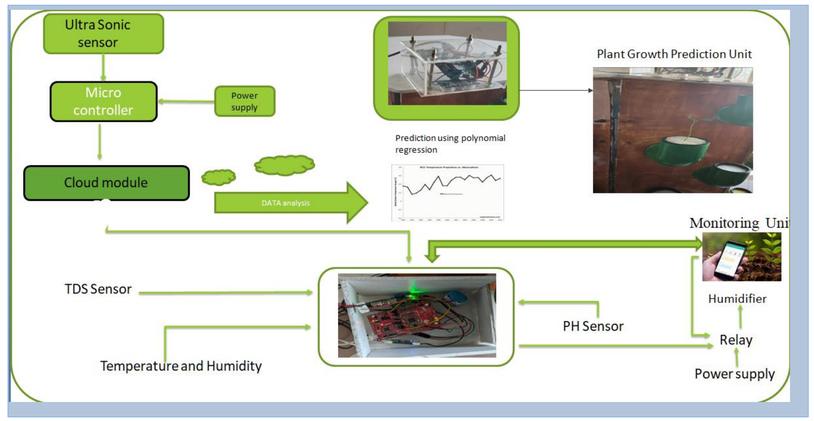
IMPACT !





Our product can give 10 times greater and 2 times faster yield as Compare with Soil Cultivation

BLOCK DIAGRAM

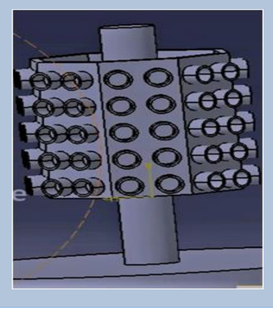


DESIGN STRUCTURE:

3-D Modeling of our Design

This modular and concept model has a very vast yield production. But the mode of communication is totally based on IOT.

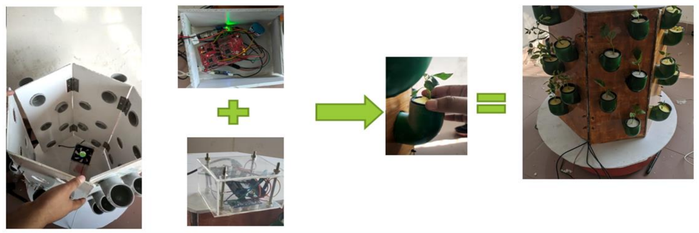
The design includes a disk like feature at the bottom which allows the upper channel to align along sun light.



The designed AutoCAD model is the merely an accurate representation of the prototype model where the humidifier is placed at the center of the Cultivation Pod.

As shown the plant spacing at each side of the pod where the sampling is placed also Styrofoam is placed for proper stem support .

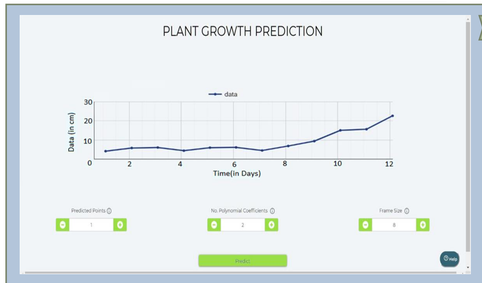
IMPLEMENTATION



NOVELTY OF OUR SOLUTION

- One of the major novelty of our solution is conservation of water.
- Maximum yield (about 6 times more) can be obtained by providing a minimum amount of water.
- Since there is no soil involved, weeds are no longer a problem.
- The nutrients required are directly fed to the roots.
- Complete monitoring of the system.
- There are many regions in India prone to calamities like droughts, floods, cyclones, etc. Since our design is portable, it can be transported (which would not affect the crop) to a safer place, which might not be possible with regular farming.

RESULTS



A prediction unit would determine plant growth and data is updated using IoT.

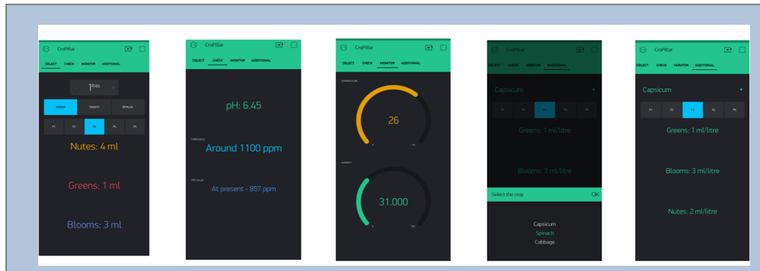
Now the parameter taken into consideration is the height of the crop.

So using certain polynomial regression technique where

X-axis = Time interval

Y-axis = Plant height(in cm)

- Yellow representation is the Predicted Scale
- Red representation is the reference growth Curve
- Blue representation is the Growth curve of crop recording every day



The mobile application is the key element in terms of informing the user with certain important elements regarding the crop growth.

The application has 4 Pages interfaced.

1) The nutrients which are to be added and at what quantity with crop time taken into factor.
2) The sensors units would give a brief values of the data recorded on real time.
3) Followed by the temperature and humidity
4) The additional crop data is also provided as well.

Cumulatively the UI of the application is so calibrated in such a way that the user only has to select the crop and the entire details are calculated according to the acquired sensor values

The Automation unit where user could turn of and on the system through this mobile application

AUTHORS

JAYANTI SAIKIRAN, UG Student, ECE Department, B.V. Raju Institute of Technology, Narsapur, Telangana, India

ITHARAJU NAVEEN, UG Student, ECE Department, B.V. Raju Institute of Technology, Narsapur, Telangana, India

RANIRUDH REDDY , Assistant Professor, ECE Department, B.V. Raju Institute of Technology, Narsapur, Telangana, India

A Vector-Length Agnostic Compiler for the Memory Limited Connex-S Accelerator

Alex E. Şuşu prof. Gheorghe M. Ştefan
DCAE department, Politehnica University of Bucharest, Romania

Key Idea

Connex-S compiler generates efficient code from sequential C programs for the Connex-S vector accelerator, in a **portable** way for arbitrary vector widths (and the CPU).
We achieve portability mainly by employing a simple JIT assembly technique, with very small overhead.

State-of-the-Art

Similar work is done by Bocchino et al. [3] for the Motorola *Reconfigurable Streaming Vector Processor (RSVP)*, also of customizable width, but their input programs are vector LLVM IR code.
Recently, ARM's compiler for the *SVE (Scalar Vector Extension) unit* is able also to generate portable binary programs, which are independent of the vector width [9].
Nuzman et al. generate portable SIMD code by using GCC to output vector .Net code, then use Mono VM to run the code on x86 with SSE or PowerPC with an AltiVec SIMD unit [7].

The Connex-S Accelerator

- Easily customizable vector processor, with 32-4096 lanes (or *execution units, EUs* or cells) [8], designed to accelerate *Basic Linear Algebra Subroutines (BLAS), deep learning and Computer Vision*.
- Use 16-bit integer EUs since narrow datapaths have good power consumption and performance [4].
- Currently, it is implemented in Xilinx Zynq-7020 FPGA SoC (28 nm technology), coupled with the integrated ARM Cortex A9 CPU. It is low power, suitable to accelerate embedded applications - for 128 lanes it consumes at most 1.5 Watts [10, 2], but as a 28 nm IC it draws more than an order of magnitude less power [5].

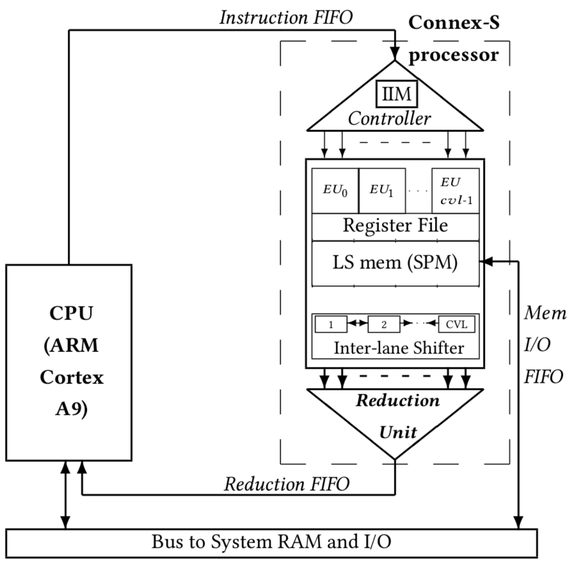


Figure 1: The architectural organization of a system with Connex-S accelerator

- The processor has: i) a linear-array interconnection network [1] called *inter-cell shift vector unit* exposed to the programmer, ii) a separate banked vector scratchpad memory (SPM) of 256 KB normally, called *Local Storage (LS)* and iii) a hardware sum-reduction tree.
  - Connex-S has a predictable performance: it has SPM instead of caches, it fetches an instruction per cycle, very simple branching checking for a scalar counter, and very few data hazards.
- The vector processor implements a predication mechanism with conditional instruction blocks of arbitrary size, similar to ARM's Thumb-2 *it* instruction.
  - The processor performs clock gating for the lanes that are not selected during predicated execution. This can yield at maximum 33% power savings.

The OPINCAA Assembler

- Our compiler generates code for OPINCAA, a runtime assembler and coordination C++ library for Connex-S.
  - OPINCAA can assemble at runtime scalar immediate operands from symbolic C/C++ expressions representing the width of the vector processor, the sizes of program arrays, access indices of arrays, loop trip counts or scalar elements of arrays, in effect allowing to run **portable** OPINCAA programs on Connex-S processors of different widths.
  - Connex-S ASM limitations addressed: no loop nesting, no scalar memory/registers, repeat loops of only constant trip counts, no kernel arguments, no function call mechanism.
- OPINCAA has a coordination API for Connex-S kernel dispatch, synchronization between CPU and Connex-S and memory transfer to/from LS memory.

The Connex-S Compiler

- The compilation flow diagram is presented in Figure 2.

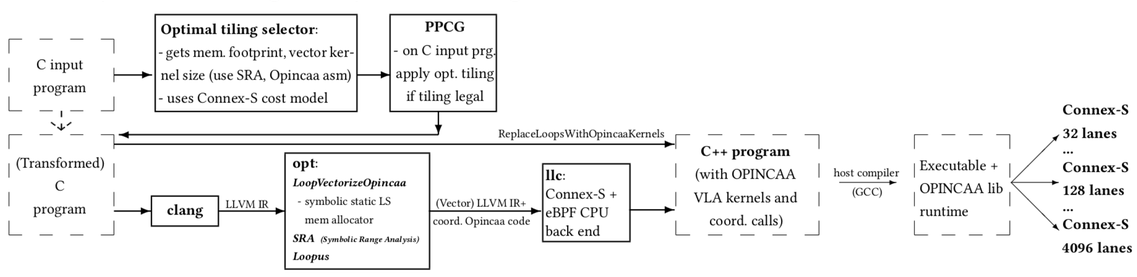


Figure 2: The stages of the Connex-S compiler

- We write a back end for the wide Connex-S vector processor with special support for:
  - symbolic scalar immediate operands, which are handled by the OPINCAA runtime assembler, effectively passing (scalar) source program variables to the kernel;
  - efficient emulation for *32-bit integer (i32)* and *16-bit floating point (f16)* types by inlining and optimizing (CSE, LICM) the code of the operations discussed in Table 1.

Type / Operation	i16	i32	f16
add	1*	15	279
sub	1*	15	280
addc	1*	21	-
subc	1*	21	-
mult	1* (+1)	27	249
srl, sra	1*	33	-
div	471		232
red (+)	1*,▷	14▷	128▷

Table 1: The number of Connex instructions implementing the emulated operations for the various types. The operations of type i16, marked with *, are directly implemented in hardware. By ▷ we mean we need to emulate also on the CPU.

- We also extend LLVM's loop vectorizer pass, which finds and handles individual loops that are profitable to vectorize, in order to:
  - generate automatically coordination calls;
  - perform static analysis with *Symbolic Range Analysis (SRA)* [6] and ScalarEvolution to retrieve the number of array elements accessed and the trip counts of loops. We recover them as symbolic expressions from LLVM IR to C/C++, which provides also input to our symbolic static LS memory allocator;
  - encode in the compiled OPINCAA programs the width of the processor as a parameter, thus being able to run most programs on Connex-S machines of any width;
  - implement efficiently loop nests by orchestrating Connex-S hardware loop instructions and unrolling

Compilation Examples

```
// C program: Multiplication of matrices with i16
//   elements, in row-major order, matrix B transposed.
//#define N 128
#define N 256
short A[N][N], B[N][N], C[N][N];
void MatMul_BTransposed() {
  int i, j, k;
  for (i = 0; i < N; ++i)
    for (j = 0; j < N; ++j) {
      C[i][j] = 0;
      for (k = 0; k < N; ++k)
        C[i][j] += A[i][k] * B[j][k];
    }
}
```

```
// Compiled OPINCAA program - works for any N and CVL
void MatMul_BTransposed() {
  // Due to lack of space we handle only this case:
  assert(N % CVL == 0 && 2 * N*N * sizeof(TYPE) <= LS_MEM_SIZE);
  // Assume: N % CVL == 0 (otherwise: pad with 0, etc)

  writeDataToConnex(A, N * N / CVL, N * N, 0);
  writeDataToConnex(B, N * N / CVL, N * N,
                    /*offset*/ N * N / CVL);

  BEGIN_KERNEL("MatMul_BTransposed.i16");
    EXECUTE_IN_ALL(
     for (i = 0; i < N; ++i) {
      R(0) = 0;
      R(1) = 1;
      R(2) = R(0);

      R(3) = N * N / CVL; // start offset for B

      REPEAT(N);
        R(2) = R(3);
        R(4) = i * N / CVL; // vload with index for A[i]
        R(5) = 0; // accumulator

        // Vectorized innermost loop k: strip-mine dot product
        for (idxLLVM = 0; idxLLVM < N; idxLLVM += CVL) {
          R(9) = LS[R(4)]; // read A[i] vector
          R(6) = LS[R(2)]; // read B[j] vector
          R(4) = R(4) + R(1);
          R(2) = R(2) + R(1);
          R(6) * R(9); R(10) = MULT_LOW();
          R(5) = R(5) + R(10); // accumulate (for dot product)
        }

        REDUCE R(5); // compute C[i][j]
        R(3) = R(2);
      END_REPEAT;
     } // End for i loop
    );
  END_KERNEL("MatMul_BTransposed.i16");

  executeKernel("MatMul_BTransposed.i16");
  readCorrectReductionResults(C, N * N, sizeof(short));
}
```

```
// Another C program for matrix multiplication: interchange
//   loops j and k to avoid transposition of B. Also, result
//   is put in LS memory, which helps if result is being
//   reused e.g. in Power of matrix kernel. % procedure.
```

```
// C program: Floyd-Warshall for i16 weights.
#define N 128
short A[N][N];
for (k = 0; k < N; ++k)
  for (i = 0; i < N; ++i) {
    // LICM correct if graph wo self-loops
    short Aik = A[i][k];
    for (j = 0; j < N; ++j)
      // Path relaxation step
      A[i][j] = min(A[i][j], Aik + A[k][j]);
  }
```

```
// Compiled OPINCAA program
// Assume: N % CVL == 0
writeDataToConnex(A, N * N / CVL, N * N, 0);

BEGIN_KERNEL("FloydWarshall_i16");
  EXECUTE_IN_ALL(
   for (k = 0; k < N; ++k) {
    R(0) = 0;
    R(1) = 1;

    R(4) = 0;
    R(5) = 0; // LS line for row of A[k]

    REPEAT(N);
      R(2) = i * N / CVL; // LS line for row of A[i]

      R(3) = A[i][k]; // INCORRECT: need read from LS

      for (idxLLVM = 0; idxLLVM < N; idxLLVM += CVL) {
        R(6) = LS[R(5)];
        R(7) = LS[R(2)];

        R(5) = R(5) + R(1);
        R(2) = R(2) + R(1);

        // Path relaxation step
        R(7) = R(7) + R(3);
        R(8) = R(6) < R(7);
        NOP;
        );
        EXECUTE_WHERE_LT(
         R(7) = R(6) | R(6);
        );
        EXECUTE_IN_ALL(
         LS[R(4)] = R(7);
         R(4) = R(4) + R(1);
        } // End for index loop
    END_REPEAT;

   } // End for k loop
   REDUCE R(0);
  );
END_KERNEL("FloydWarshall_i16");

executeKernel("FloydWarshall_i16");
readReduction(); // for sync

readDataFromConnex(A, N * N);
```

Experiments

- Figure 3 shows performance speedups when running on Connex-S synthesized at 100 MHz on Xilinx Zynq-7020 FPGA, with 128 lanes and 256 KB LS memory [2], w.r.t. one ARM core of Cortex A9 at 667 MHz.
  - GCC 7.2 is unable to vectorize for ARM NEON the following kernels using i16 elements: *DotProd* (dot product), *MatMul* (square matrix multiplication, the second matrix being already transposed), *Sum of Squared Differences (SSD)* and *Sum of Absolute Differences (SAD)*. The slowdown of *DotProd* is due to the small arithmetic intensity, while for i32 elements it gets even smaller since the CPU code gets vectorized and we emulate i32 on Connex-S.
  - The subunitary speedups can be avoided by increasing the Connex-S vector width to e.g. 1024.
  - Kernel runtime assembly takes on average 20.78 ms, but we cache the binary instruction stream for the same kernel input sizes.
- We can run the same generated C++ OPINCAA program on Connex-S machines of different widths - see Figure 4. The log-log plot for the speedup of a given benchmark in this figure is not linear due to the non-negligible communication, vector loop prologue and epilogue overheads and scalar code.

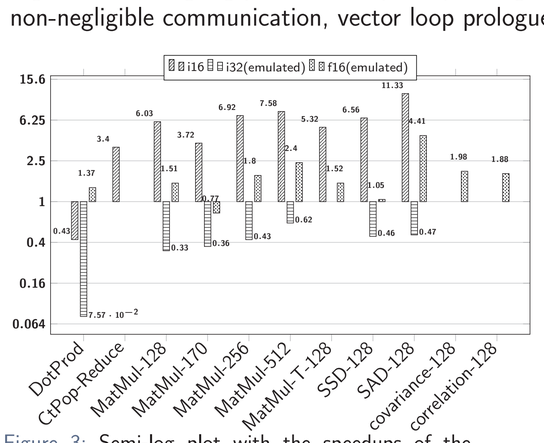


Figure 3: Semi-log plot with the speedups of the benchmarks on Connex-S with 128 lanes, at 100 MHz w.r.t. the dual-core ARM Cortex A9 at 667 MHz with a total of two 128-bit NEON SIMD units

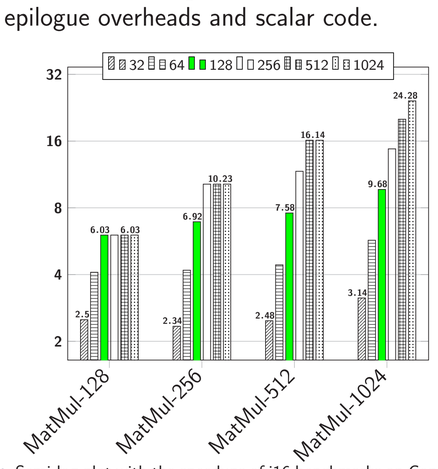


Figure 4: Semi-log plot with the speedups of i16 benchmarks on Connex-S with a number of lanes between 32 and 1024, clocked at 100 MHz w.r.t. the dual-core ARM Cortex A9 at 667 MHz with a total of two 128-bit NEON SIMD units

- For *MatMul-128* we achieve on Connex-S 1.12 GOPS/Watt for type i16, 0.172 GOPS/Watt for type i32 and 0.031 GFLOPS/Watt for type f16. For the *MatMul-128.i16* kernel we achieve 1.22x better energy efficiency w.r.t. the dual-core ARM Cortex A9 integrated in the Zynq SoC, which is low power, and for most other i16 kernels we achieve even higher ratios.

Future Work

- Compare throughput and *Energy Delay Product (EDP)* of f16 kernels when running on very wide (4096 lanes) Connex-S with 16-bit narrow EUs w.r.t. mainstream GPUs and ARM SVE.

References

[1] S. G. Akl.
The Design and Analysis of Parallel Algorithms.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[2] C. Bîră, L. Gugu, R. Hobincu, L. Petrică, V. Codreanu, and S. Coţofană.
An Energy Effective SIMD Accelerator for Visual Pattern Matching.
In *Proceedings of the 4th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2013.

[3] R. L. Bocchino, Jr. and V. S. Adve.
Vector LLVA: A Virtual Vector Instruction Set for Media Processing.
In *Proceedings of the 2nd International Conference on Virtual Execution Environments*, VEE '06, pages 46–56, New York, NY, USA, 2006. ACM.

[4] D. Brooks and M. Martonosi.
Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance.
In *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, HPCA '99, pages 13–, Washington, DC, USA, 1999. IEEE Computer Society.

[5] M. Maliţa and G. M. Ştefan.
Map-Scan Node Accelerator for Big-Data.
In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3524–3529, Dec 2017.

[6] G. Mendonça, B. Guimarães, P. Alves, M. Pereira, G. Araújo, and F. M. Q. a. Pereira.
DawnCC: Automatic Annotation for Data Parallelism and Offloading.
ACM Trans. Archit. Code Optim., 14(2):13:1–13:25, May 2017.

[7] D. Nuzman, S. Dyshel, E. Rohou, I. Rosen, K. Williams, D. Yuste, A. Cohen, and A. Zaks.
Vapor SIMD: Auto-vectorize Once, Run Everywhere.
In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '11, pages 151–160, Washington, DC, USA, 2011. IEEE Computer Society.

[8] G. M. Ştefan and M. Maliţa.
Can One-Chip Parallel Computing Be Liberated From Ad Hoc Solutions? A Computation Model Based Approach and Its Implementation.
In *Proceedings of the 18th Int. Conf. on Computers (CSCC'14)*, Recent Advances in Computer Engineering Series 23, pages 582–597, 2014.

[9] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker.
The ARM Scalable Vector Extension.
IEEE Micro, 37(2):26–39, Mar. 2017.

[10] A. E. Şuşu.
Compiling for the Wide Connex Vector Processor.
Parallel Architectures and Compilation Techniques (PACT), Cyprus, 2018.

Visit https://gitlab.dcae.pub.ro/research/ConnexRelated/ - Opincaa C++ library has also cycle-accurate simulator for Connex of custom width.

alex.susu@gmail.com

Mignon: ML using Tsetlin Machine

A Hardware Demonstration

Adrian Wheeldon and Jie Lei
{adrian.wheeldon, jie.lei}@ncl.ac.uk

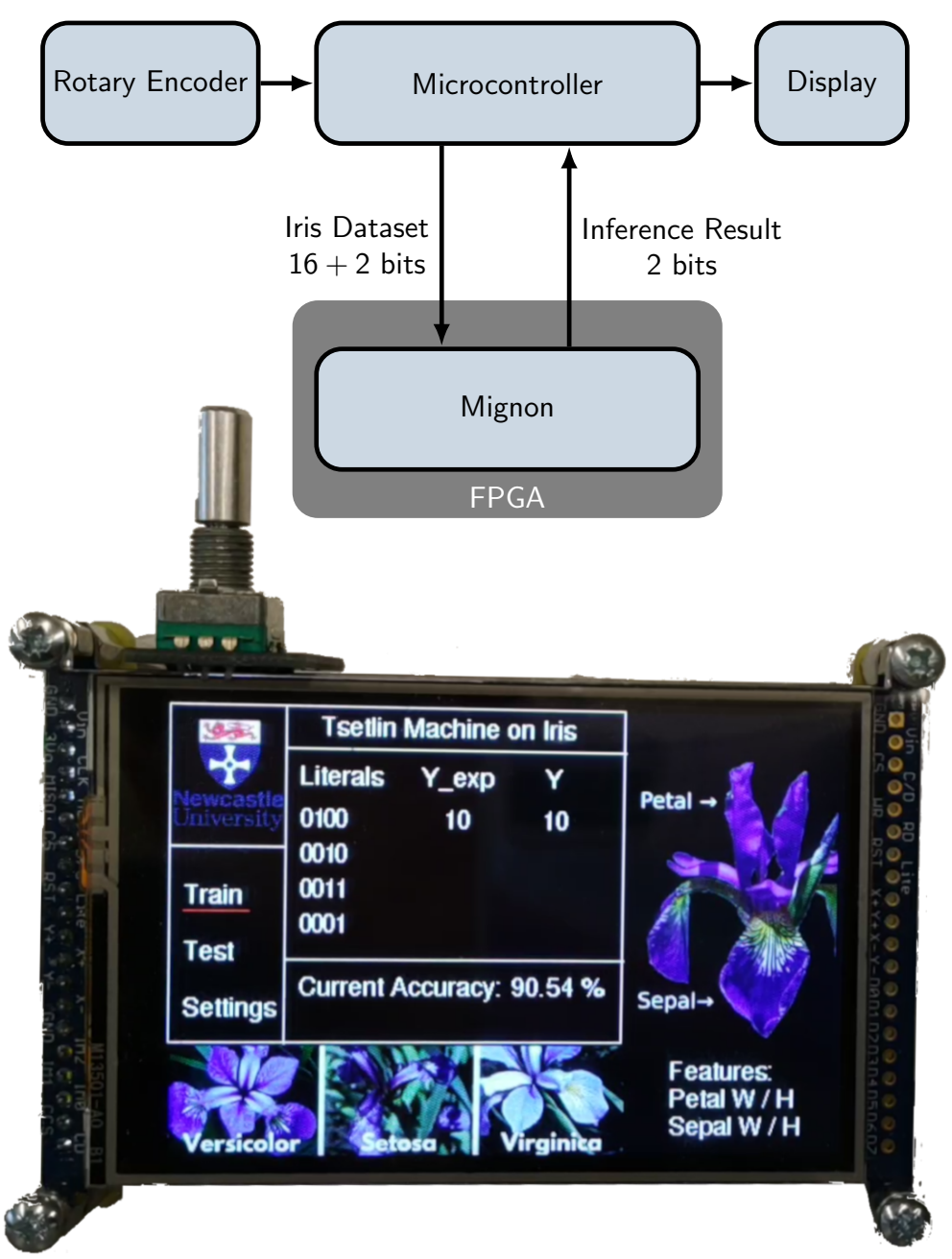
Newcastle University

Motivation

- Neural networks are presently king in machine learning.
  - Based around floating-point calculations.
  - Learning mechanism makes explainability difficult.
- Mignon is inherently CMOS-compatible.
  - Hardware-centric algorithm – **Tsetlin Machine**.
  - FSM-based learning ∴ trivial on-chip implementation.
  - **No floating-point** used in learning or classification.
  - Easily-explainable classification.
  - Inherently dependable.

Demo System Design

- **World's first** ML hardware based on Tsetlin Machine.
- Proof of concept classifies Iris flowers.

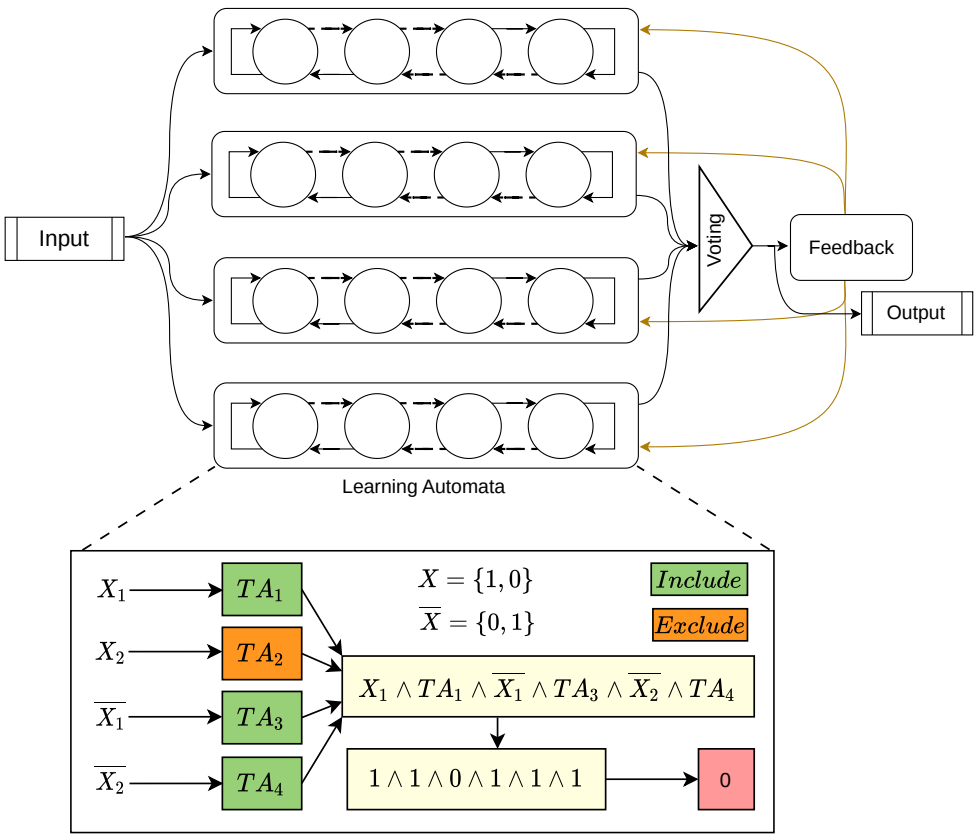


Iris Flower Classification

- Length and width of petal and sepal identifies the species.
  - Measurements are converted to 4-bit binary.
- 16-bit feature inputs to Mignon.
- 2-bit class encoding for 3 flower species.
  - Input expected class, and output predicted class.

Tsetlin Machine Algorithm

- Reinforcement of actions using **learning automata**.
  - FSMs with stochastic component for diverse learning.
- Inference using **propositional logic** and **majority voting**.
  - Simple logic leads to easy explainability.
  - Majority voting ensures high dependability.



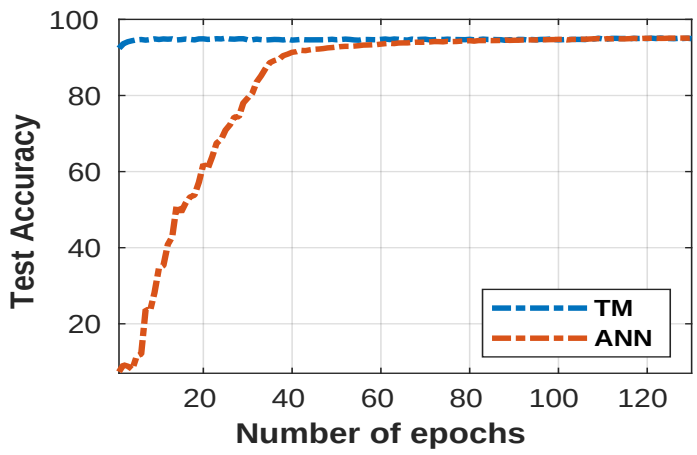
Results

- Iris Accuracy; **Mignon: 97.0 %**, XGBoost: 96.7 %.
- Inference energy vs competing hardware in 65 nm silicon:

BNN[a]	**Mignon***	Neuromorphic[b]	CBNN[c]
88.5 Top J^{-1}	**62.7 Top J^{-1}**	48.2 Top J^{-1}	25.2 Top J^{-1}

*From high-confidence ASIC simulation.

- **Faster learning convergence** vs ANN:



Future Work

- ASIC currently in manufacture.
- Algorithm improvements for decreased area and power.
- Develop low power/energy applications.

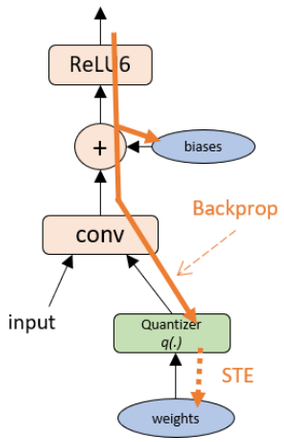
In collaboration with CAIR, Full Mignon Demo, Papers and Resources.

[a]W. Choi, DAC 2018. [b]D. Miyashita, JSSC 2017. [c]H Yonekawa, IPDPSW 2017.

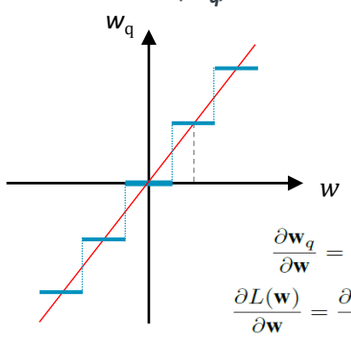
Alpha-Blending - An Optimization Technique to Quantize Low-precision Neural Networks

Zhi-Gang Liu and Matthew Mattina @arm research

❖ The straight-through estimator, introduced by Hinton, is a gradient estimator that allows to use binary threshold units in neural networks trained by backpropagation. It uses the threshold unit normally during the forward pass and replacing it with the identity function during the backward pass. However, STE is an empirical approach, lacks theoretical justification.



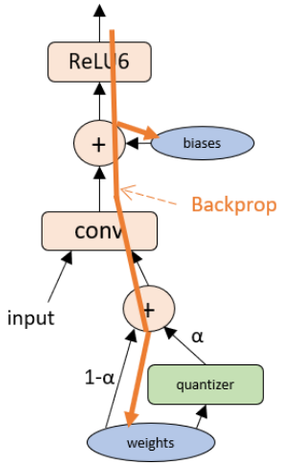
▪ Quantize parameter $\boldsymbol{w}$ to $\boldsymbol{w_q}$ to minimize loss function $\boldsymbol{L(w_q)}$ of a neural network using ***STE***



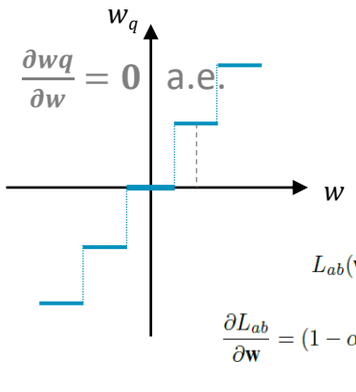
$$\frac{\partial \mathbf{w}_q}{\partial \mathbf{w}} = \frac{\partial q(\mathbf{w})}{\partial \mathbf{w}} \underset{STE}{\approx} \frac{\partial I(\mathbf{w})}{\partial \mathbf{w}} = 1 \quad (2)$$

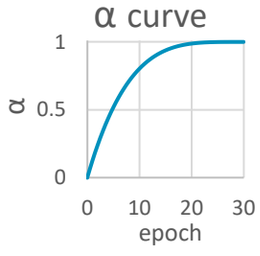
$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial L(\mathbf{w}_q)}{\partial \mathbf{w}_q} \cdot \frac{\partial \mathbf{w}_q}{\partial \mathbf{w}} \underset{STE}{\approx} \frac{\partial L(\mathbf{w}_q)}{\partial \mathbf{w}_q} \quad (3)$$

❖ Alternative to the well-known Straight-Through Estimator (STE), Our approach – Alpha Blending (**AB**) uses the affine combination (with coefficient α) of the trainable parameter w and the corresponding optimization w_q in the loss function for optimization training. BP doesn't go through the threshold op, therefore mitigated it's vanished gradient issue. Ramp up the coefficient α from 0 to 1 gradually.



▪ Quantize parameter $\boldsymbol{w}$ to $\boldsymbol{w_q}$ to minimize loss function $\boldsymbol{L(w_q)}$ of a neural network using **AB**

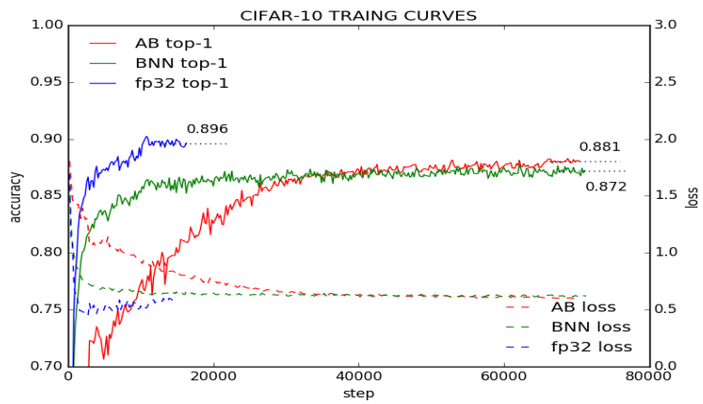




$$L_{ab}(\mathbf{w}, \alpha) = L((1-\alpha)\mathbf{w} + \alpha \mathbf{w}_q) \quad (4)$$

$$\frac{\partial L_{ab}}{\partial \mathbf{w}} = (1 - \alpha + \alpha \cancel{\frac{\partial \mathbf{w}_q}{\partial \mathbf{w}}}^{=0 \text{ a.e.}}) \frac{\partial L(\mathbf{w}')}{\partial \mathbf{w}'}\Big|_{\mathbf{w}'=(1-\alpha)\mathbf{w}+\alpha \mathbf{w}_q} \quad (5)$$

❖ Apply **AB** to BinaryNet on CIFAR10

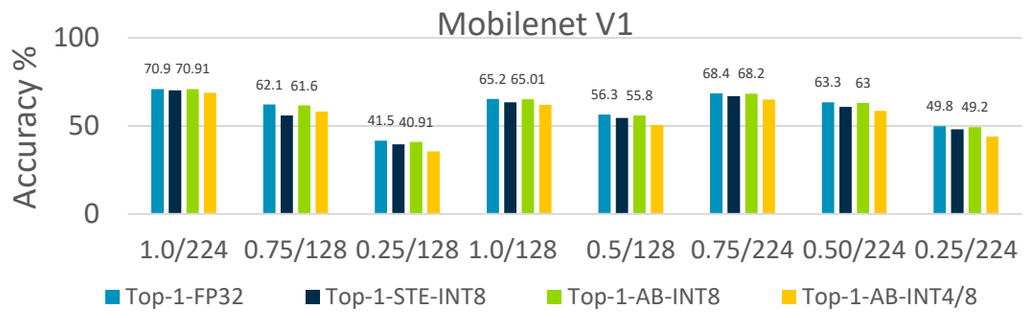


▪ Binarlization

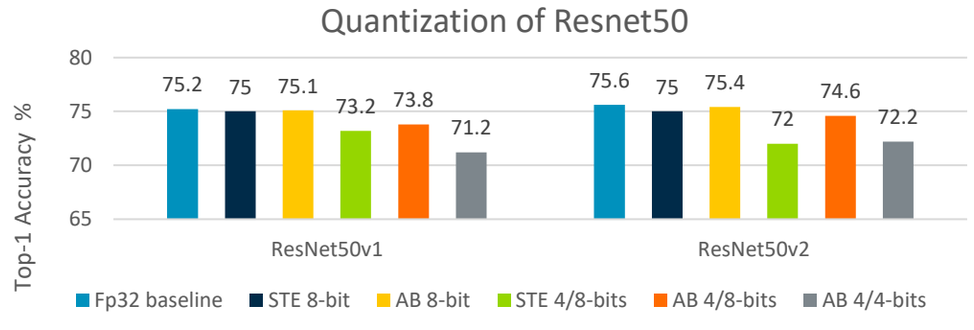
$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

[Hubara et al., 2016a] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 4107–4115. Curran Associates, Inc., 2016. https://github.com/itayhubara/BinaryNet.tf

❖ Apply **AB** to quantize MobileNet v1 on ImageNet



❖ Apply **AB** to quantize ResNet50 on ImageNet



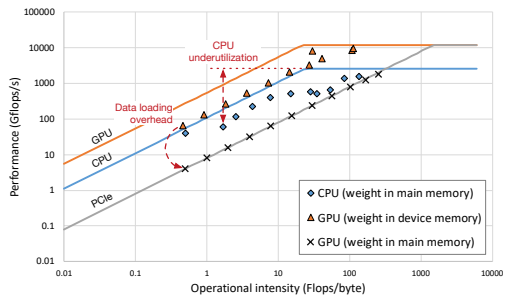
arm

- **Bandwidth-bound DL inference**
  - **MLP-dominant models**
    - Main target: GEMM in FC layers
  - **Small batch size (short latency target)**
    - Tall-skinny & fat-short activations
  - **Large memory-resident weights**
    - Likely in multi-tenant servers

Common DL-inference GEMM dimensions

	Description	Weights	Input	Batch
Language Model (GPT-2/3 & BERT)	MLP	768×3072	3072	8
	MLP	3072×768	768	8
	Projection	768×768	768	8
Recommender Model (DLRM RM3)	Bottom MLP	512×2560	2560	1-256
	Bottom MLP	32×512	512	1-256
	Top MLP	128×512	512	1-256
	Top MLP	1×128	128	1-256

- **Roofline analysis**
  - **CPU execution**
    - BW bound for batch ≤32
  - **GPU execution**
    - PCIe BW bound **OR**
    - Low BW utilization



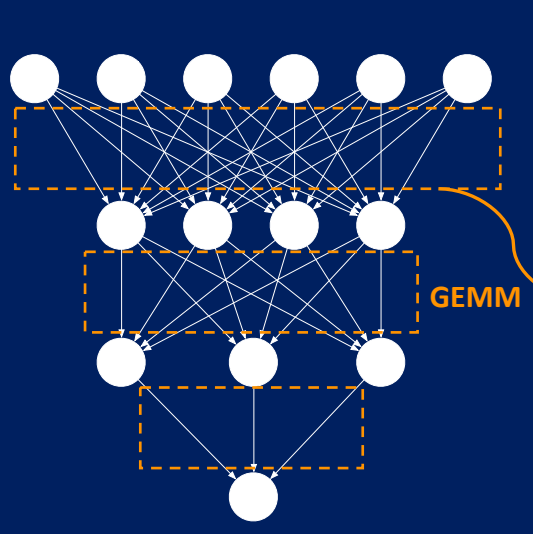
- **Opportunity and challenges**
  - **Processing in main memory (PIM)**
    - High memory BW
    - Already sharing data with the CPU
    - Freeing up the CPU resources
  - **Challenges**
    - Complex CPU address mappings
    - Exploiting GEMM locality
    - Avoid command BW bottleneck

Main-Memory Accelerators for Bandwidth-Bound DL Inference

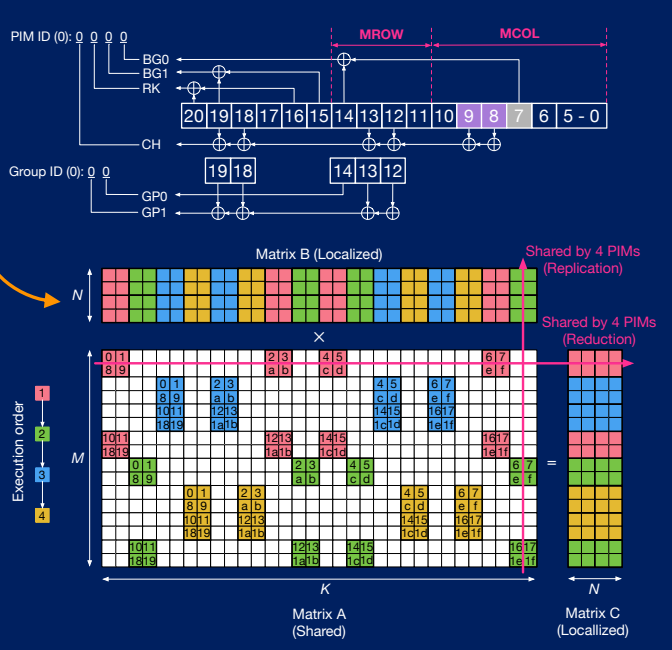
Benjamin Cho and Mattan Erez

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

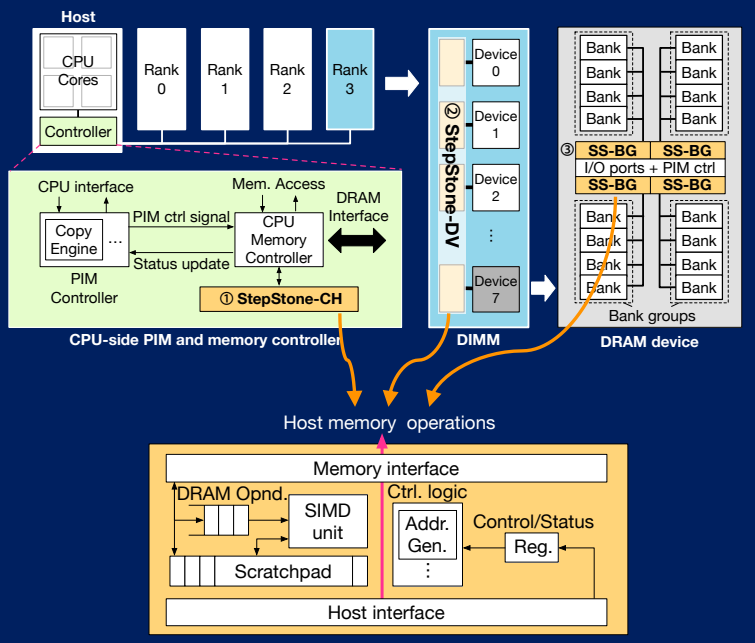
We accelerate bandwidth-bound DL inferences with processing in *main* memory, overcoming limited command BW and locality-breaking XOR-based address mappings.



Multi-Layer Perceptron (MLP)



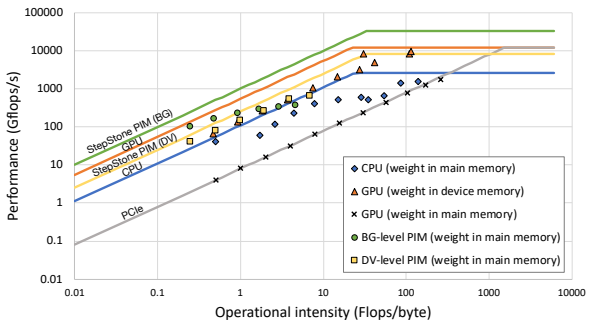
Blocked GEMM for PIM0



PIM architecture

- **Proposed mechanisms**
  - **Address mapping aware block grouping**
    - Grouping the weight matrix blocks based on temporal locality → effective data reuse in scratchpad
  - **StepStone address generation**
    - Generate physical addresses in each PIM → CPU-independent execution
    - Sequence addresses for each block group → exploit temporal locality

- **Key performance results**
  - **Comparison to CPU**
    - Batch-1: 45x speedup
    - Batch-32: 4x speedup



- **Power and energy results**
  - **Power:** dominated by DRAM power
  - **Energy**
    - Lower DRAM energy when accessing from closer to the memory cells
    - Impact of overheads ↑ when N ↑