Characterizing Optimizations To Memory Access Patterns Using Architecture Independent Program Features

PRESENTER: ADITYA CHILUKURI[1]

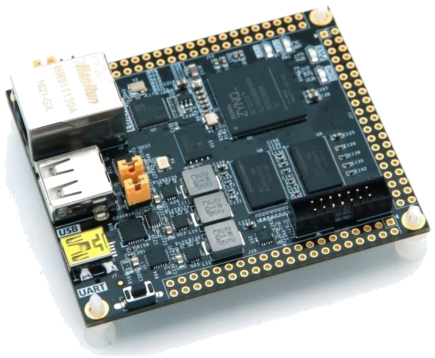
CO-AUTHORS: JOSH MILTHORPE[1], BEAU JOHNSTON[2,1]

[1] AUSTRALIAN NATIONAL UNIVERSITY

[2] OAK RIDGE NATIONAL LABORATORY

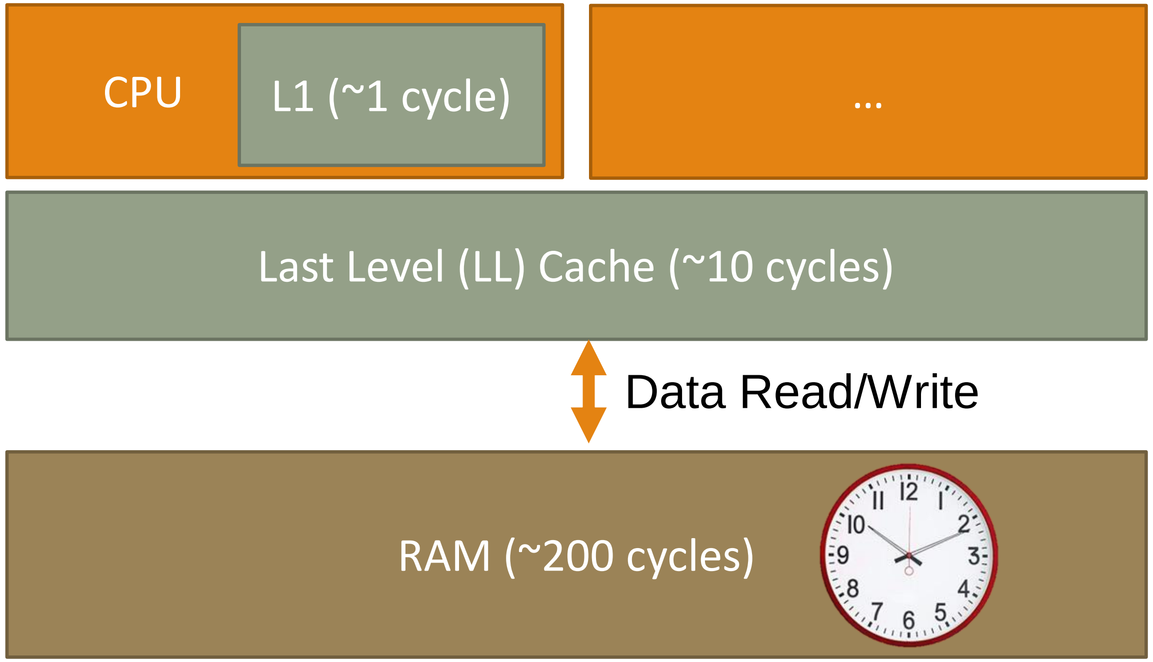
Introducing: Heterogenous Computing

- Shift towards incorporating diverse range of computer architectures: CPUs, GPUs, FPGAs, ASICs.
- OpenCL language designed for code to be executed on diverse hardware “targets”.



Why Memory Access Behaviour Matters

- Memory accesses are a major cause of bottlenecks on modern computer architectures.
- *Spatial Locality* for Caches: Programs that frequently access nearby memory addresses tend to have better performance.



What patterns in the memory accesses performed by a program are good for performance on varying hardware targets?

Problem Statement

“Develop a method to help HPC developers understand how their *code* interacts with *memory* – independent of the target hardware platform.”

Introducing: AIWC ('air-wik)

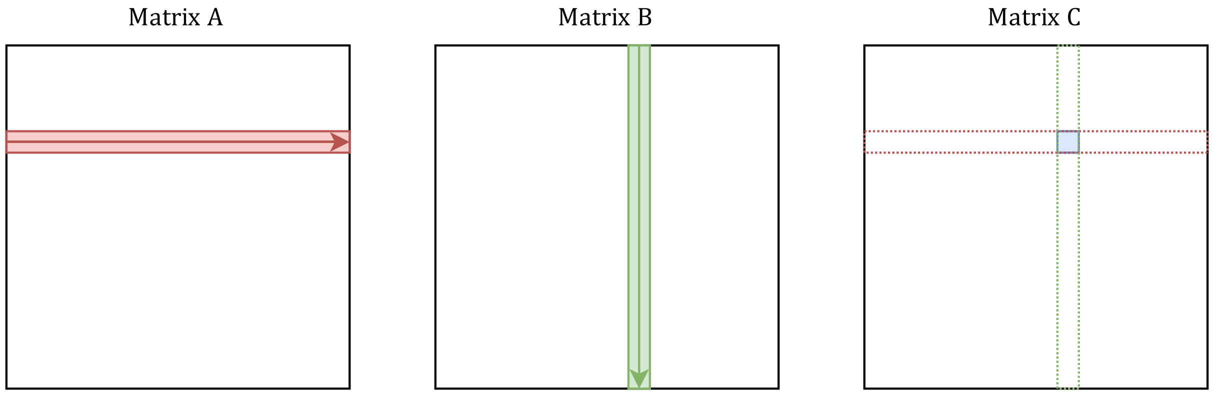
- Architecture Independent Workload Characterisation (AIWC) tool for OpenCL – Developed by Beau Johnston and Josh Milthorpe.
- Plugin for the Oclgrind simulator for OpenCL.
  - Executes OpenCL kernels on abstract virtual OpenCL devices
  - Follows OpenCL memory and execution model
- Architecture-Independent Oclgrind simulation allows for architecture-independent analysis of OpenCL code.

Introducing: AIWC ('air-wik)

- Collect metrics that characterise parallel programs.
- Metrics collected are independent of the hardware target of an OpenCL kernel.
- Memory based metrics:
  - Total Memory Footprint: How much memory access occurs. (*Lower is better*)
  - Memory Address Entropy: Measure of spread of memory regions accessed. (*Lower is better*)

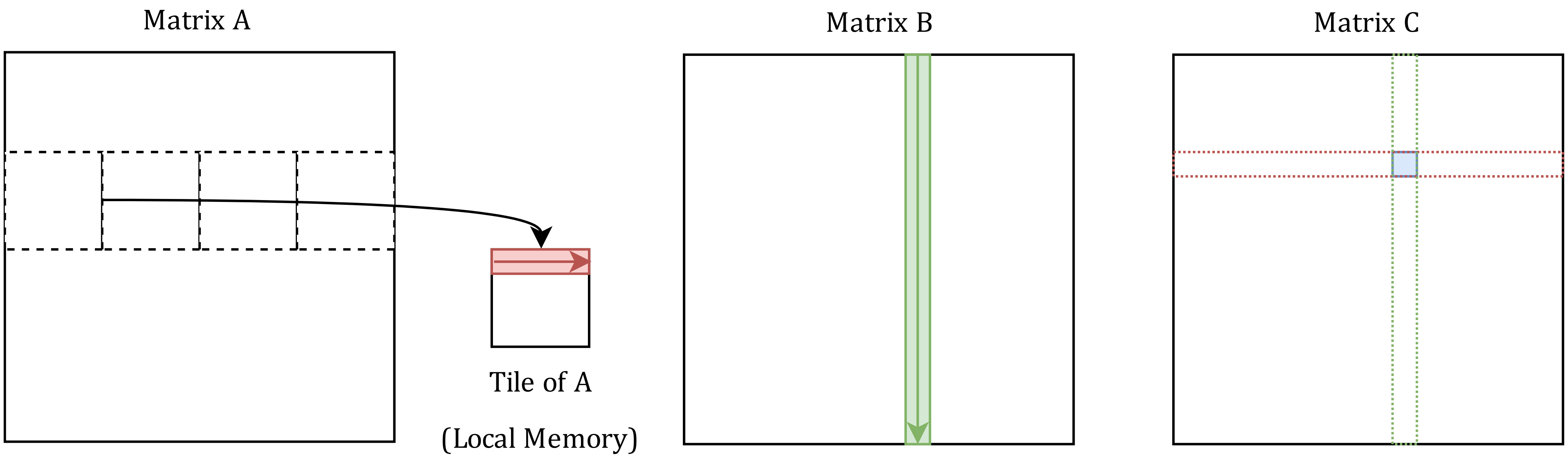
A Test-Case Kernel for Optimisation

```
__kernel void simpleMultiply(__global float *A,
                             __global float *B,
                             __global float *C,
                             int N)
{
    float acc = 0.0f;
    for (int k = 0; k < N; ++k) {
        acc += B[k * N + globalCol]
               * A[globalRow * N + k];
    }
    // Store the result
    C[globalRow * N + globalCol] = acc;
}
```

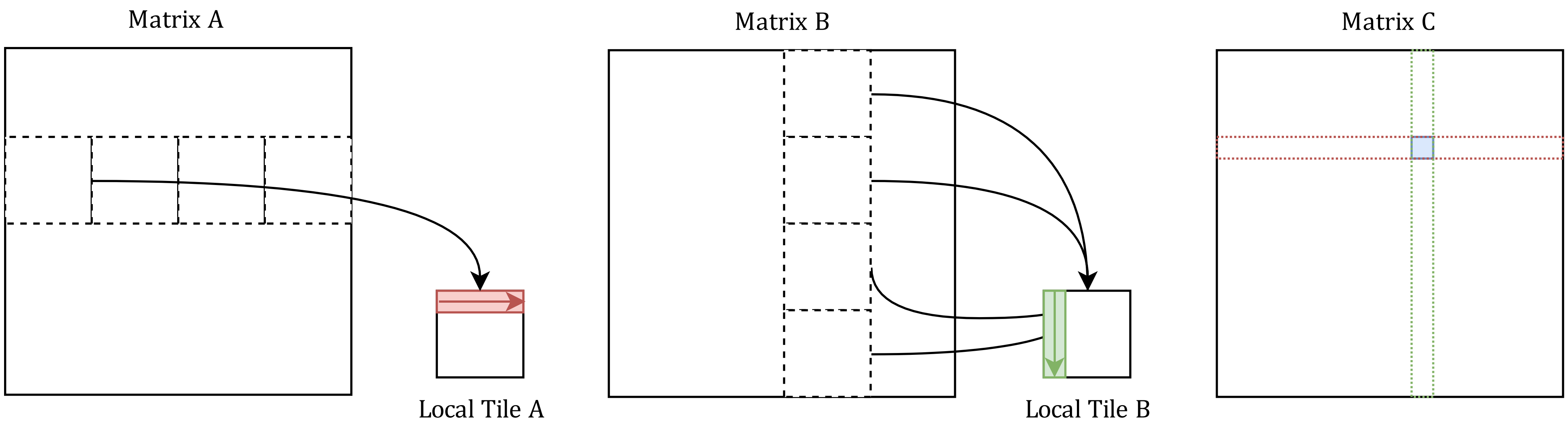


(NVIDIA Corporation. Cuda C best practices guide. 2019.)

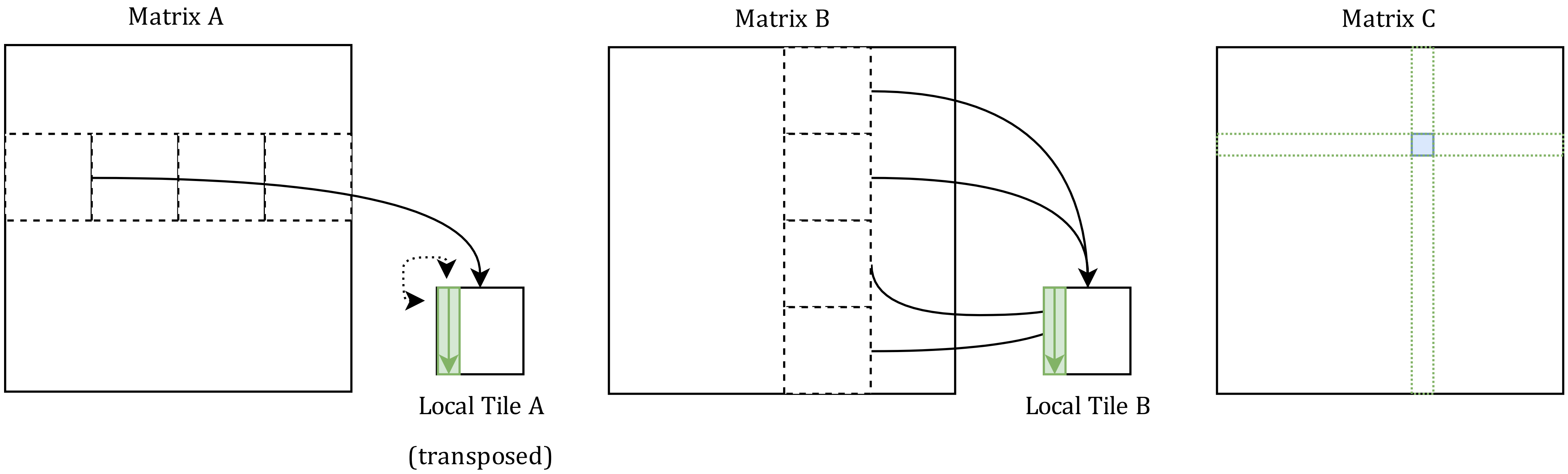
Coalescing Accesses to Matrix A (`coalescedA`)



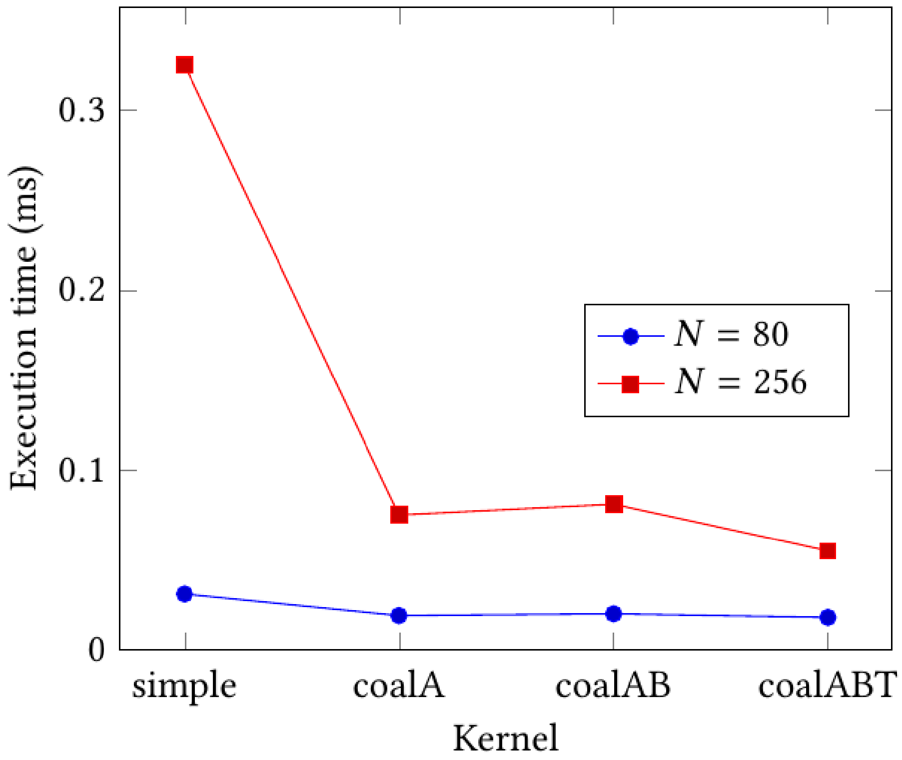
Coalescing Accesses to Matrix B (`coalescedAB`)

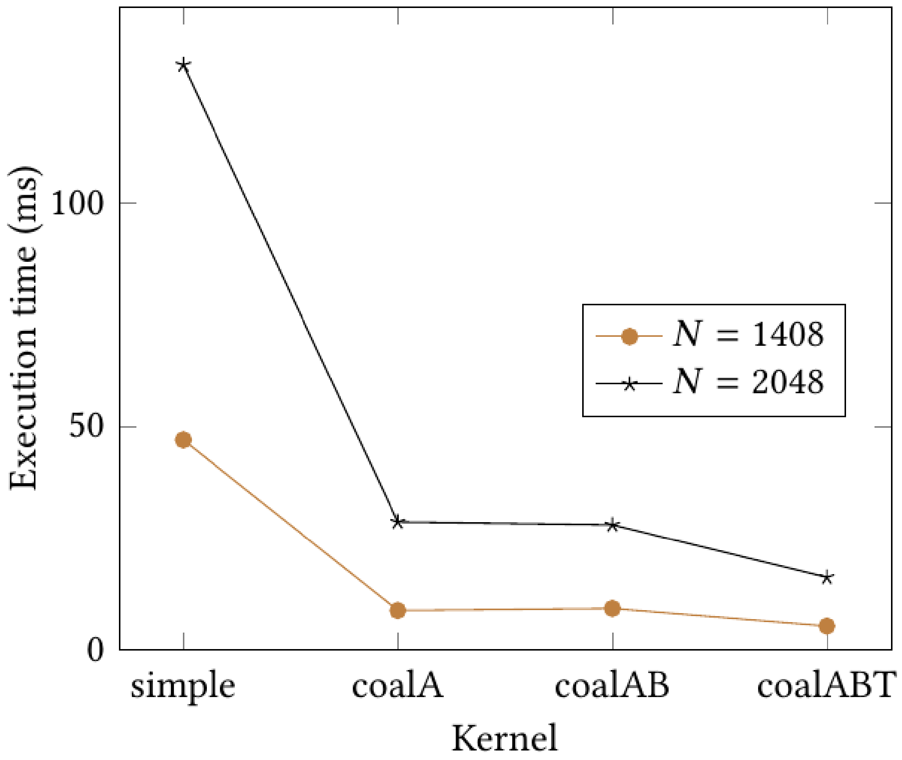


Efficient Local Memory Usage (`coalescedABT`)



Performance Results





Creating new AIWC Metrics!

- *Observation*: Accesses to “local” memory (or fast access on-chip memory) are good
  - **New Metric:** Relative Local Memory Usage (RLMU). (*Higher is better*)
- *Observation*: Parallel accesses to nearby memory addresses are good
  - **New Metric:** Parallel Spatial Locality (PSL).

10110110101001
10110110101010
01011001101011
11000100110101
11101101000010

Set of memory addresses

→

10110110101~~001~~
10110110101~~010~~
01011001101~~011~~
11000100110~~101~~
11101101000~~010~~

Set of memory addresses
3 bits skipped

→

Entropy Statistic
(Real number)

The Parallel Spatial Locality Metric

Formal Definition:
Calculate entropy (or spread) of memory addresses at each timestep.
Repeat entropy calculations at varying "skipped-bits"
Calculate the following:

$$PSL_{n-bits}(t) = \sum_{\alpha \in A_n(t)} p_\alpha \log_2(p_\alpha^{-1}) \qquad (1)$$

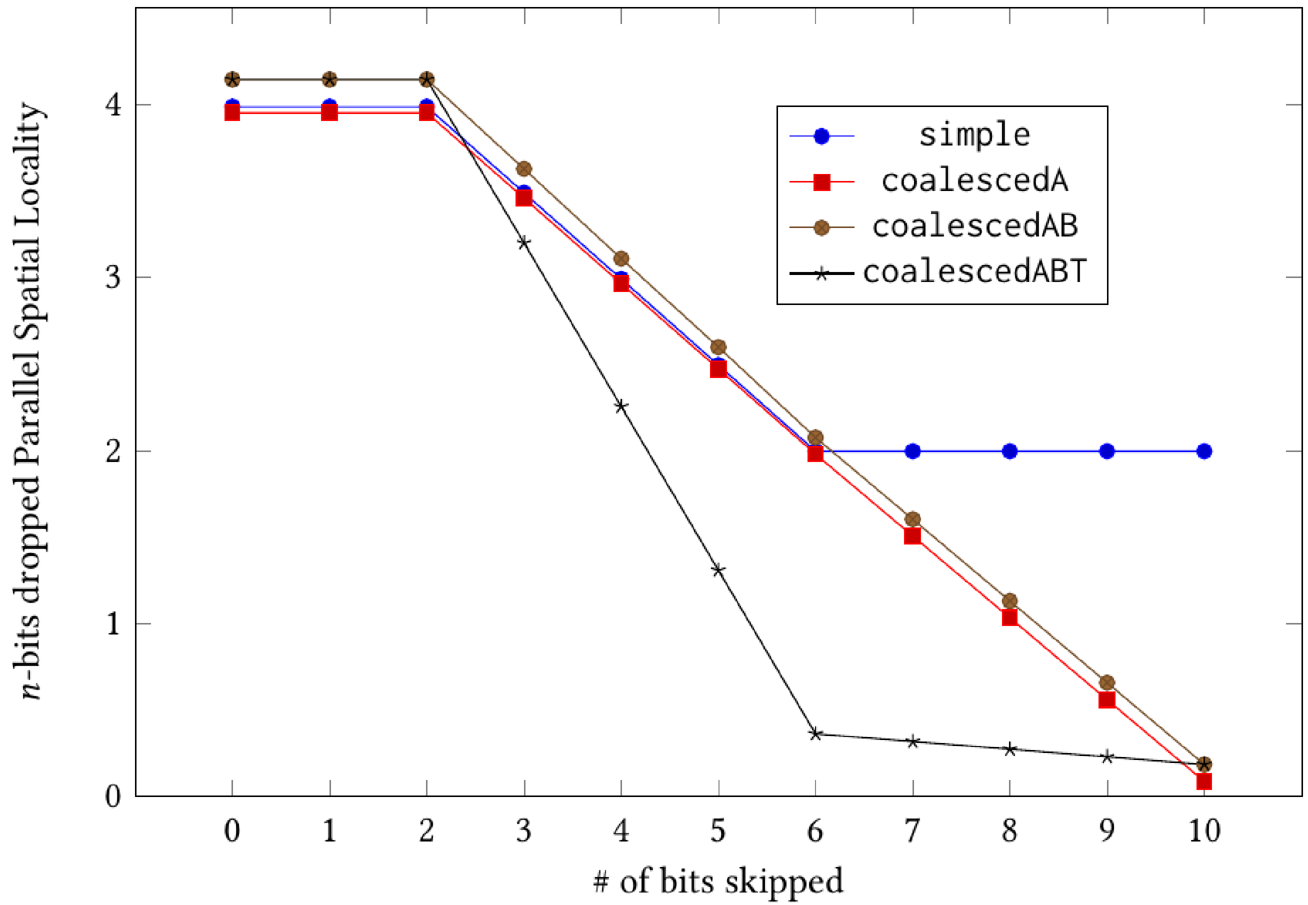
with $A_n(t)$ the set of addresses accessed at time t accessed after skipping n bits, p_α the probability of a specific address.
Average this value across all timesteps of program execution to obtain PSL_{n-bits}.
A higher number of threads in an OpenCL workgroup leads to higher PSL_{n-bits} values. We normalise the PSL_{n-bits} by dividing by $\log_2(n_{threads-per-group})$.

- Main takeaway: the *steeper the drop* in PSL as the number of bits skipped increases, the more localised the memory accesses are.

Preliminary Findings

	simple	coalescedA	coalescedAB	coalescedABT
Total memory footprint	196608	196608	196608	196608
90% Memory Footprint	118196	56176	489	489
Global MAE	17.02	13.18	9.78	9.78
LMAE #bits=3	16.02	12.18	8.78	8.78
LMAE #bits=10	9.02	5.18	1.78	1.78
Relative Local Memory Usage	0	0.50	0.94	0.94

Findings



simple
coalescedA
coalescedAB
coalescedABT
n-bits dropped Parallel Spatial Locality
# of bits skipped
0
1
2
3
4
5
6
7
8
9
10

Testing on Extended OpenDwarfs (EOD) Benchmark Suite

- The EOD benchmarks are a set of diverse OpenCL codes satisfying each of the 13 *Berkeley Dwarfs:*
  - N-body methods
  - Dense Linear Algebra
  - Finite State Machines
  - Structured Grids
  - Graph Traversal
  - and more...
- OpenCL codes representative of each dwarf typically induce similar memory access patterns.

Results

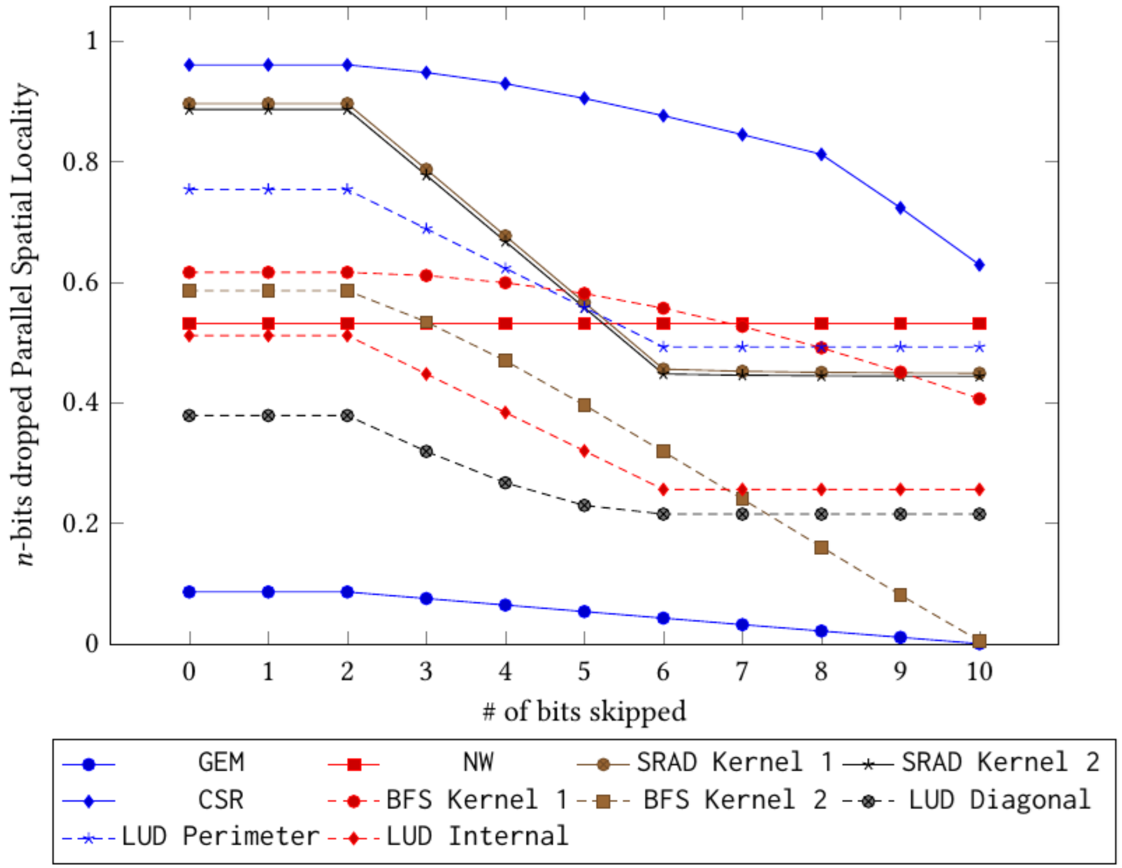
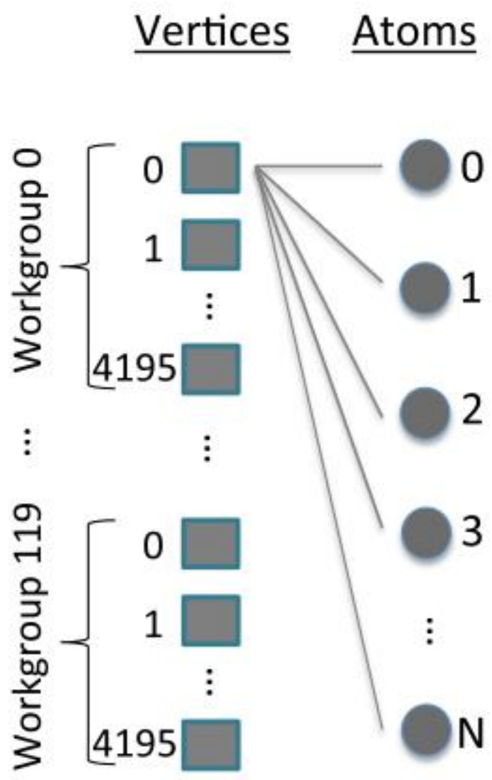
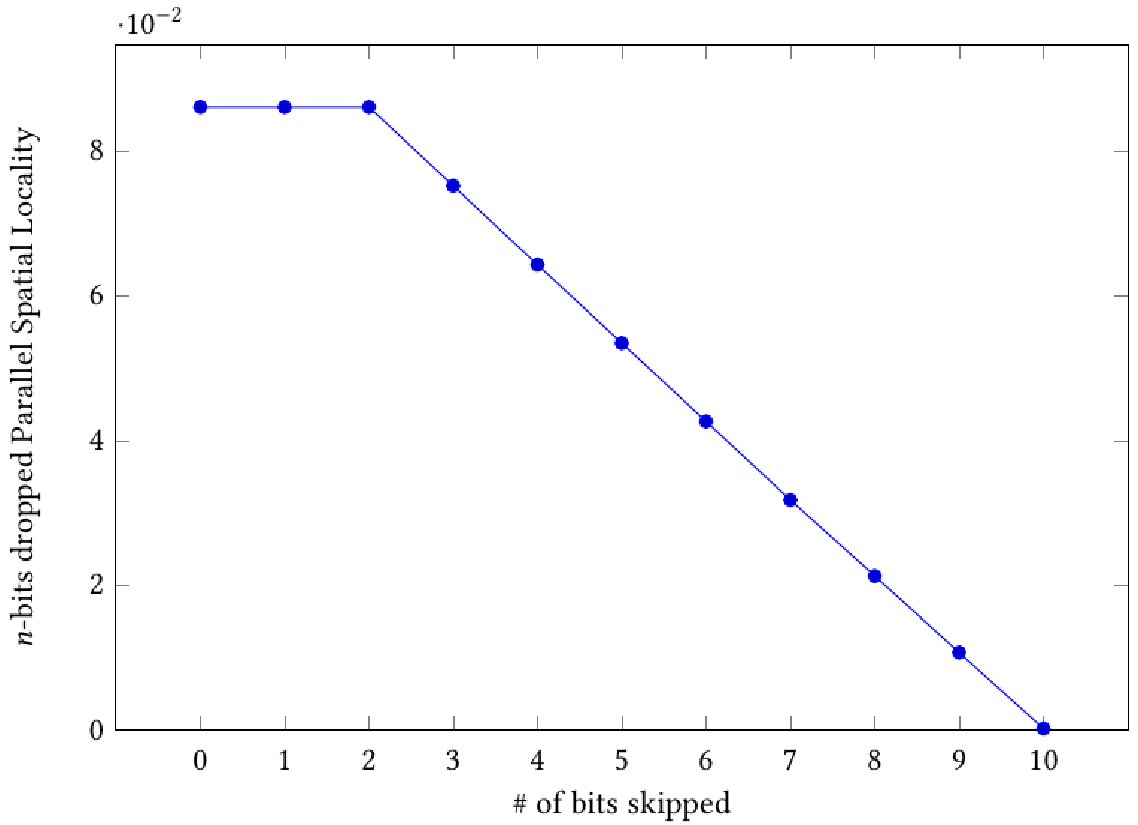


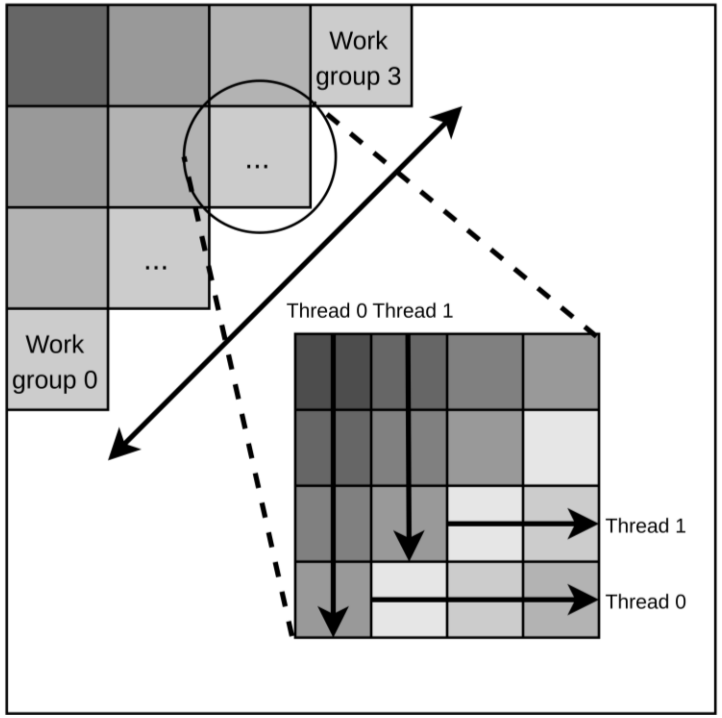
Figure 3: Parallel spatial locality metric for selected OpenDwarfs benchmark kernels

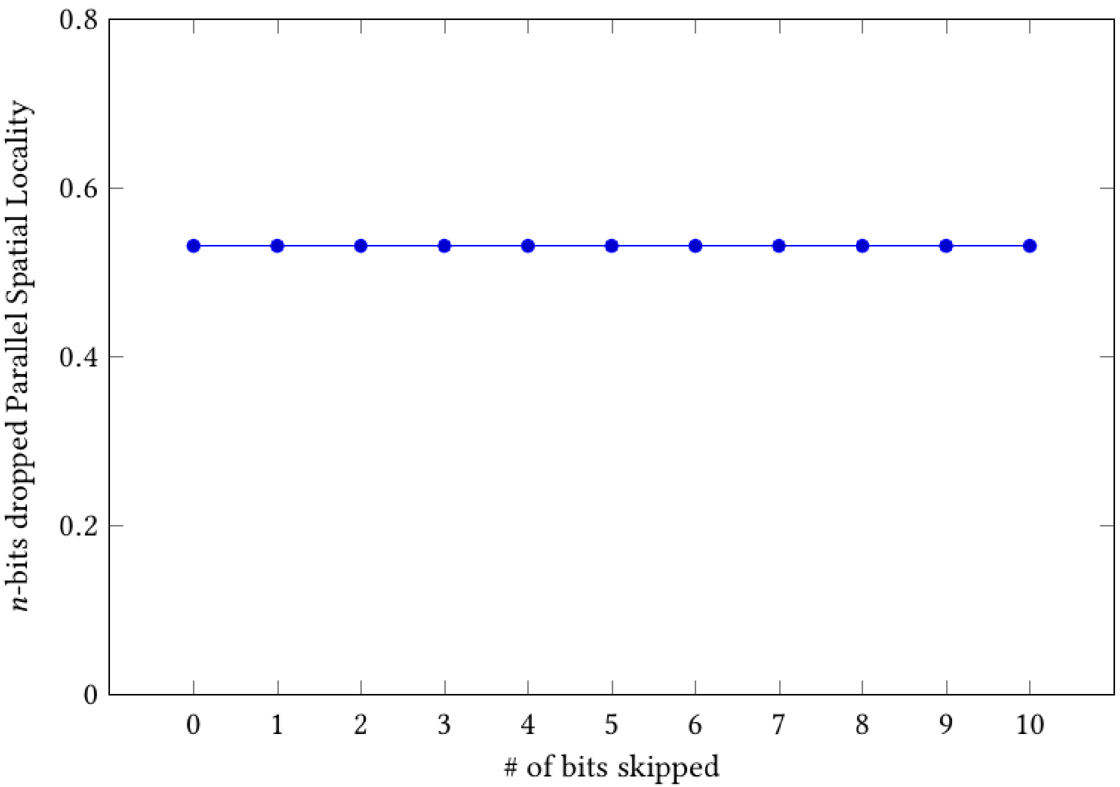
GEM: N-body Methods OpenDwarfs Benchmark



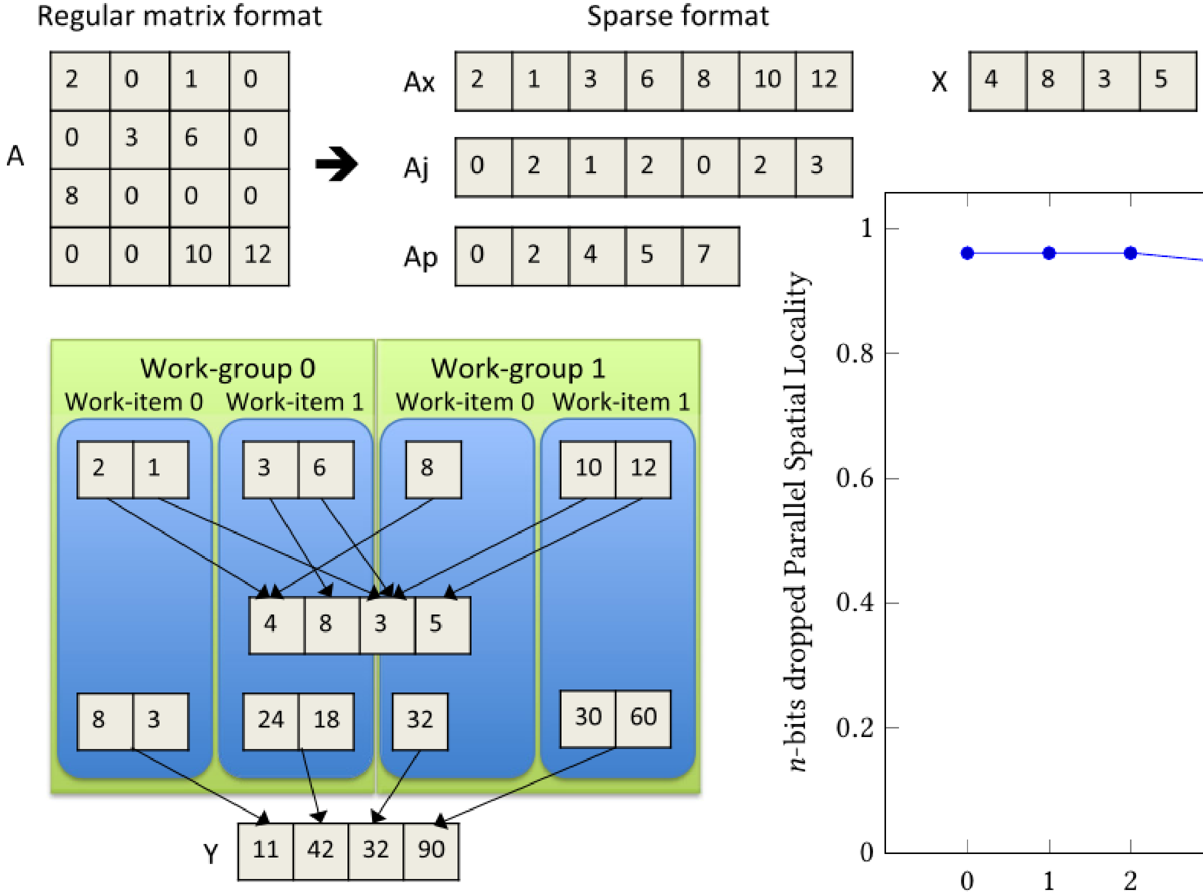


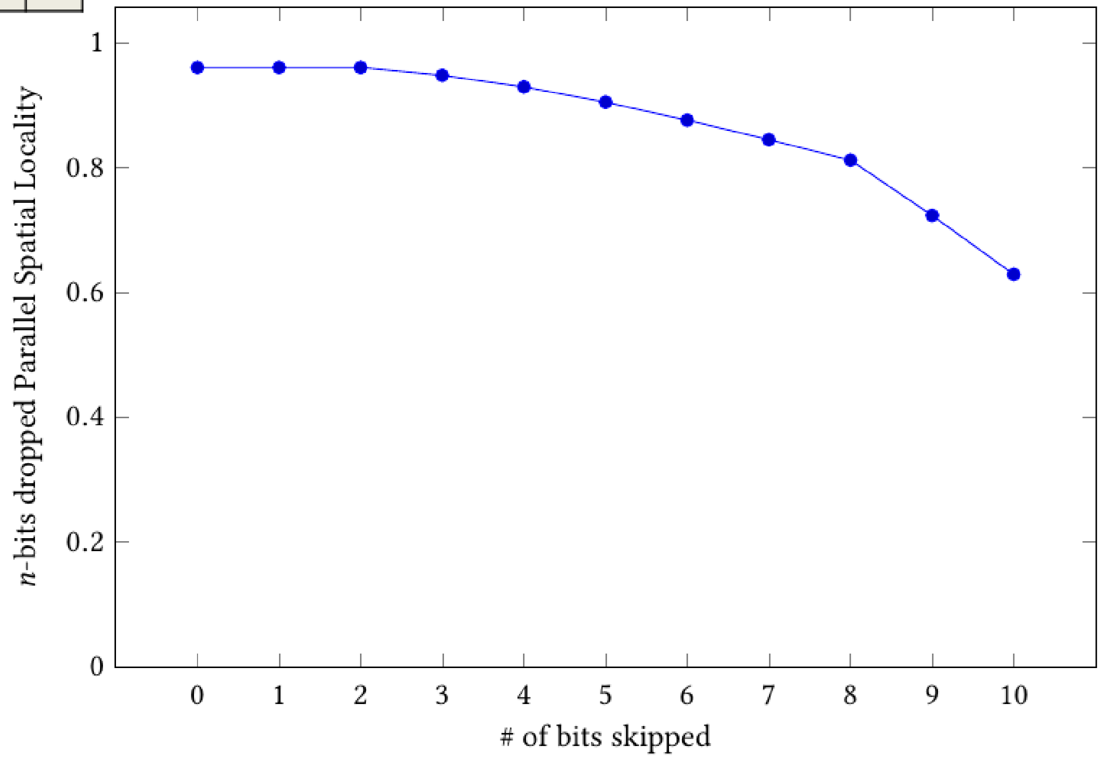
Needleman-Wunsch: Dynamic Programming OpenDwarfs Benchmark





CSR: Sparse Linear Algebra OpenDwarfs Benchmark





Conclusions and Future Work

- Proposed two new metrics to AIWC framework.
- Parallel Spatial Locality is the first architecture independent metric of its kind for parallel programs.
  - Tested the metric against the Extended OpenDwarfs Benchmarking Suite.
- Improve AIWC to help HPC developers better understand (and optimise) their complex codes.
- Extend current methodology to create metrics for:
  - Different optimisation strategies (not only memory-based ones).
  - Different target architectures – CPUs and FPGAs.