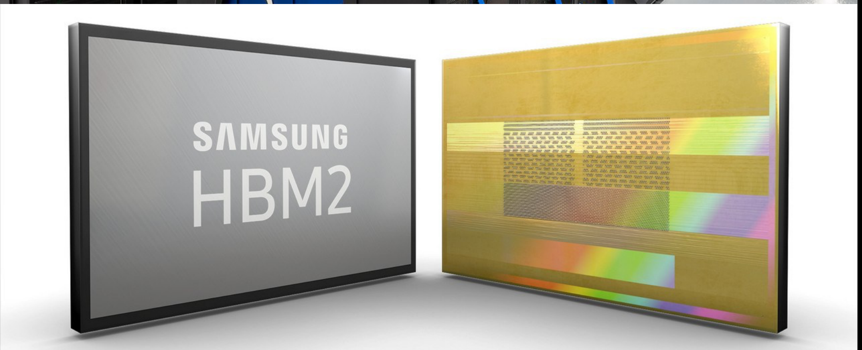
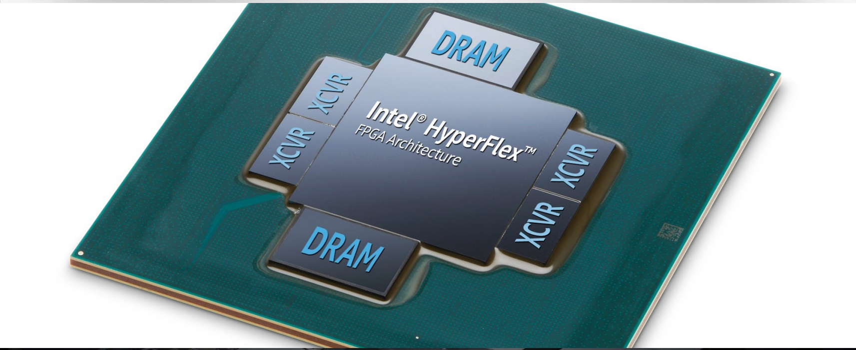
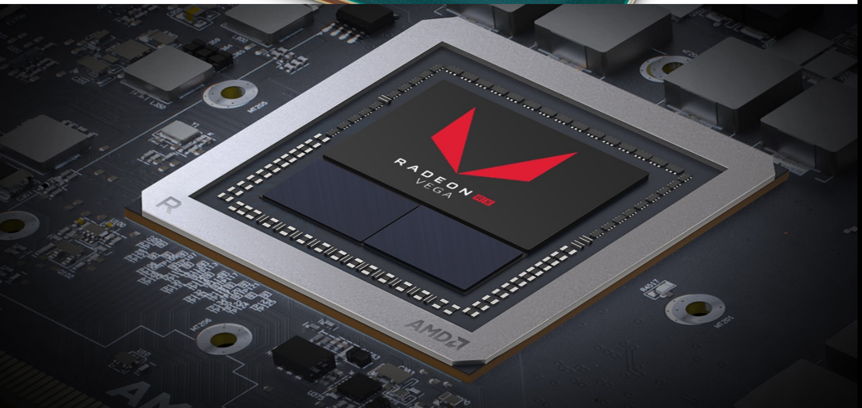
COMPILER-BASED MEMORY ALLOCATION FOR DRAM-HBM HYBRID MEMORY SYSTEM

Onkar Patil, Frank Mueller, Latchesar Ionkov, Michael Lang

North Carolina State University, Los Alamos National Laboratory







TECHNOLOGY

- Hybrid Memory systems with High Bandwidth Memory(HBM) exist
  - Summit @ ORNL -> Volta GPUs with HBM2, NV-link
- Potential devices for future systems
  - AMD Vega GPU architecture
  - Intel's Stratix 10MX FPGA
- HBM device manufacturers
  - Samsung
  - Micron
  - Intel
  - SK Hynix

PROBLEM

- Language key words extensions or macros help in memory allocation, but…
  - Onus of utilizing the memory technologies optimally, falls on the skill and knowledge of the programmer
- Not every data structure and compute kernel benefits from the same memory

PROPOSED SOLUTION

- Use Static analysis from the compiler to automatically identify and classify critical data structures and kernels based on
  - Scope
  - Nesting(Nesting score)
  - Access Pattern(r, w, rw)
  - Proximity
  - Effective bandwidth ratios
- Perform source-to-source transformation to change the allocation and then recompile
  - Allocate memory using a single interface for all memory devices(SICM)

ASSUMPTIONS

- Memory architecture consistent with the envisioned Exascale Node architecture
- Data Flow can exist in two ways
  - Partitioned address space
  - Cached Memory system
- NUMA access to all memory devices

Ang, James A., et al. "Abstract machine models and proxy architectures for exascale computing." *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*. IEEE Press, 2014.

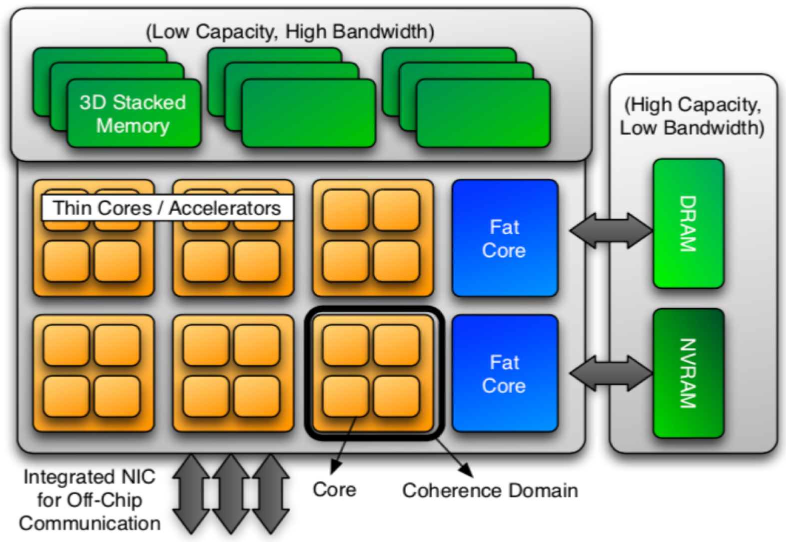
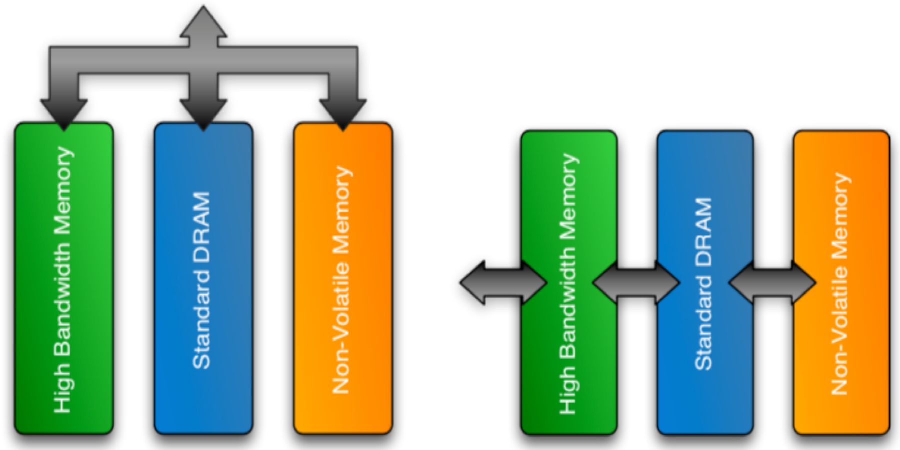


Fig. 1: Overarching Abstract Machine Model, Exascale Node Architecture



(a) Physical Address Partitioned Memory Subsystem

(b) Multi-Level Cached Memory System

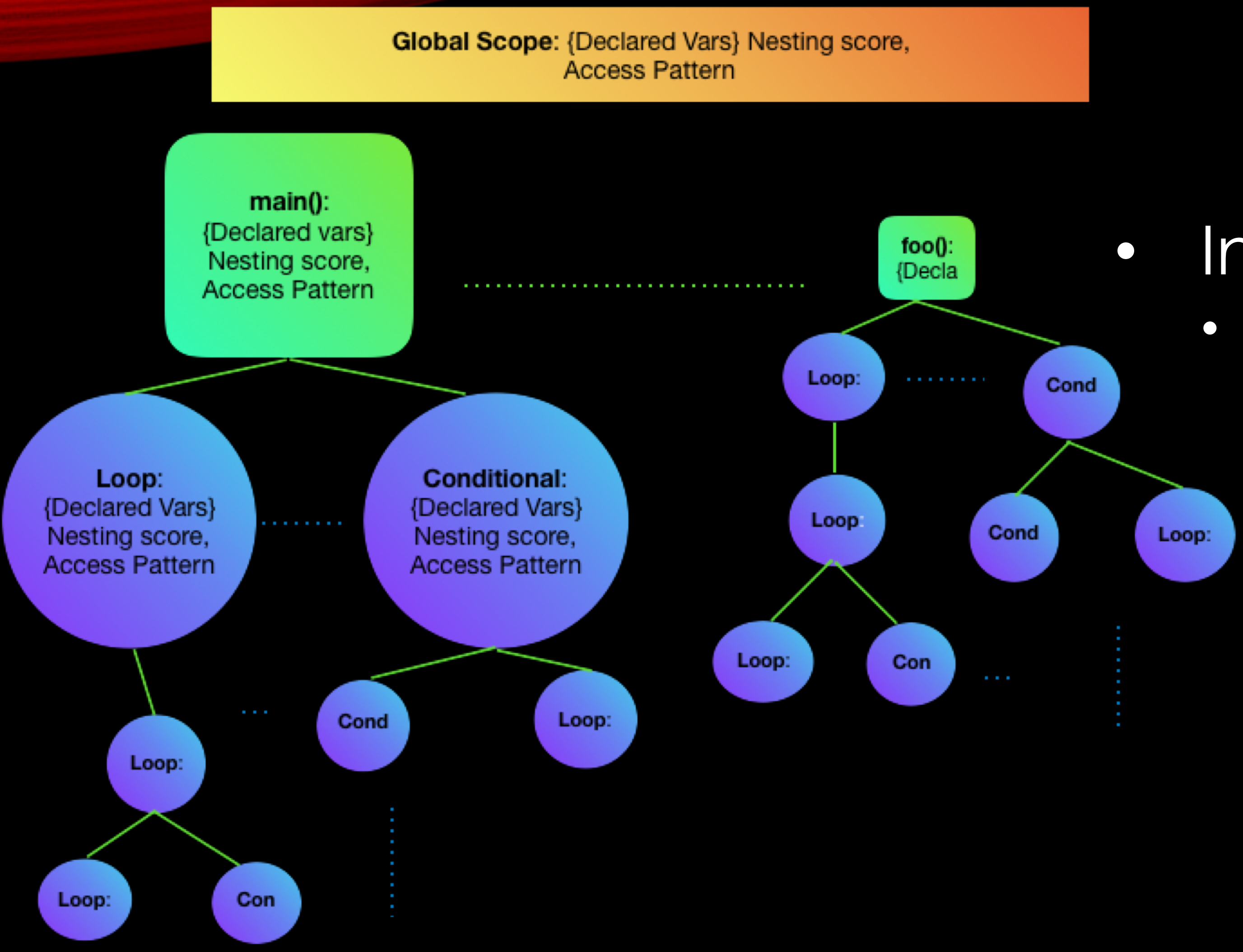
Fig. 8: Memory Subsystem Layouts

MEMORY DEVICE CHARACTERIZATION

- Run micro-benchmark once
  - Based on STREAM
  - Double loop timed
  - 512 MB workload
  - Averaged over 30 runs
- Identify NUMA devices with the underlying memory technology
  - Classification using K-means
  - Memory classes provided by user(HBM, DRAM, NVM)
  - Bandwidth and Latency numbers for different types of operations

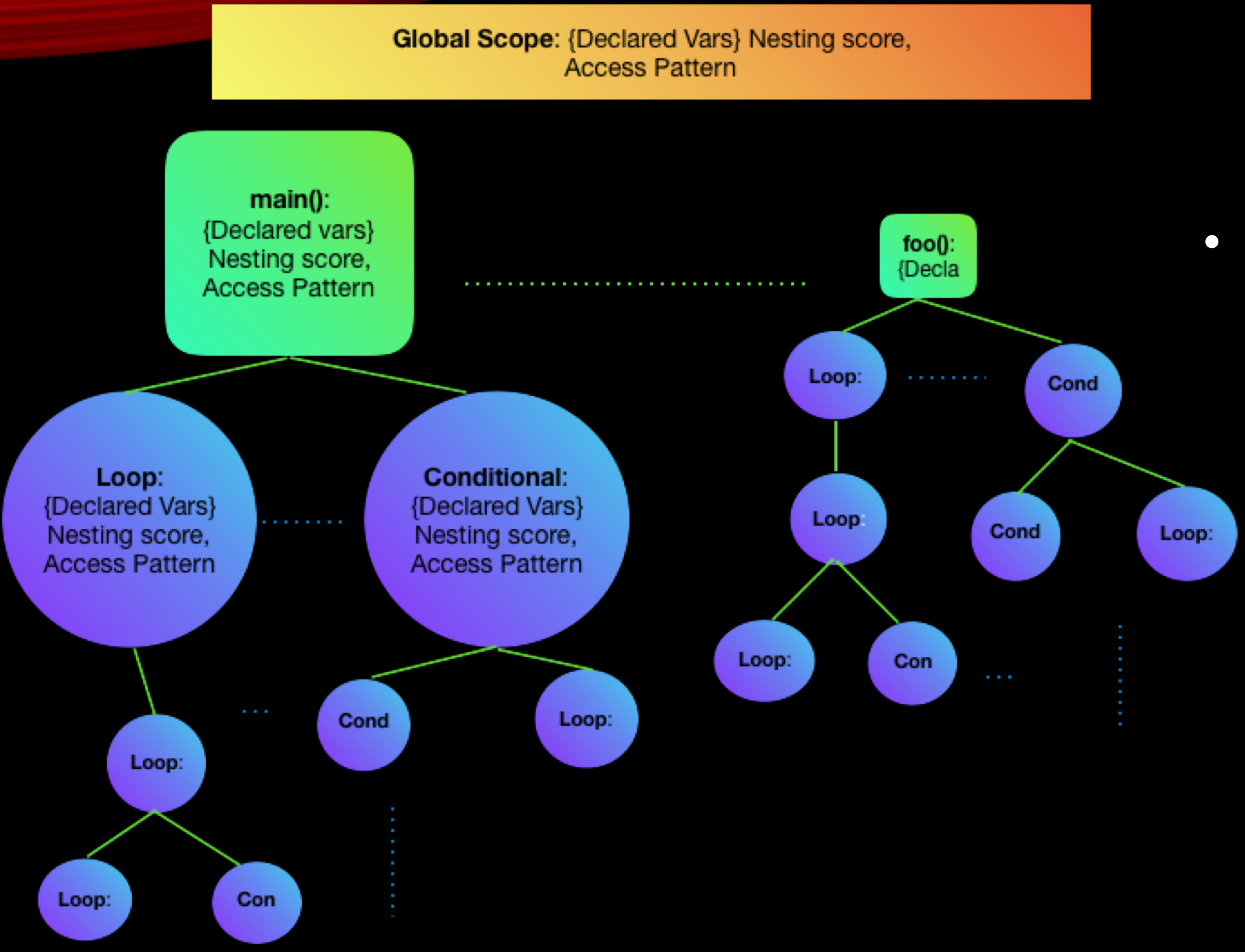
	HBM	DRAM
Avg_BW	1521.722599	1273.181041
R_BW	1012.372162	991.869843
W_BW	2031.073037	1554.492239
RW_BW	294.457057	279.220723
Ran_BW	7.235049	8.159102
Lin_BW	53.822347	53.39672
Same_BW	273.052658	266.690019
Diff_BW	132.100891	130.594896
Avg_lat	2.9E-09	3.4E-09
R_lat	3.8E-09	3.9E-09
W_lat	0.000000002	2.9E-09
RW_lat	0.000000013	1.37E-08
Ran_lat	5.273E-07	4.675E-07
Lin_lat	7.09E-08	7.14E-08
Same_lat	0.000000014	1.43E-08
Diff_lat	2.89E-08	2.92E-08
BW - MB/s		
lat - s		

STATIC ANALYSIS



- Information Gathered
  - Data Structures
    - Scope → global, function, loop, conditional
    - Memory operation→ r, w, r+w
    - Access Pattern → sequential, strided, linear, random
    - Aliasing → pass by reference
    - Code Location → place of declaration
    - Proximity → accessed in the same expression or scope

STATIC ANALYSIS

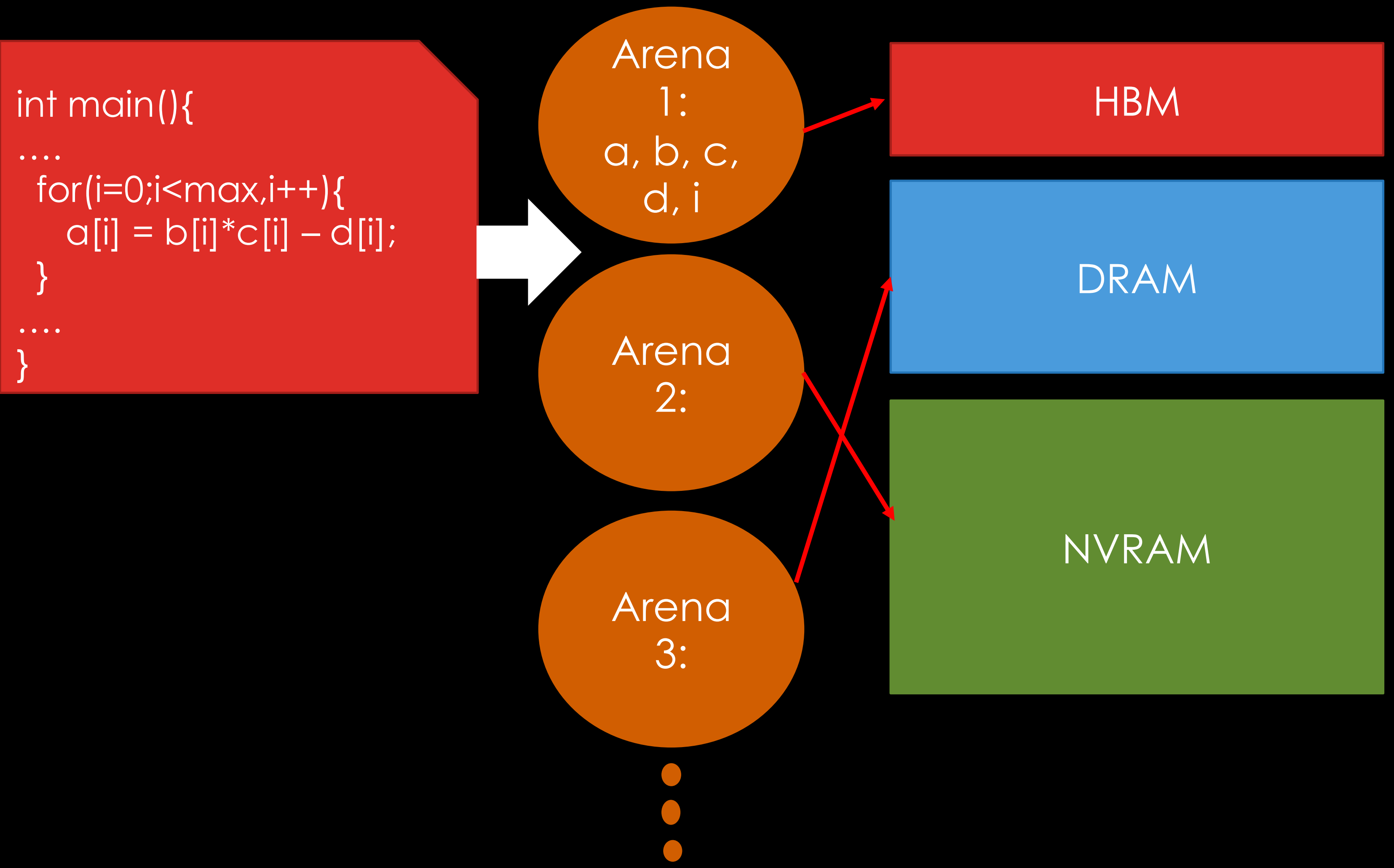


- Loops and Conditionals
  - Code Location → Start and end of code, line & column nos.
  - Nesting score → How nested is the loop/conditional? Function =1, Loops/conditionals = 2, 3, …

PROXIMITY SCORE

- Metric for
  - Grouping data structures that are frequently accessed together
  - Indicating appropriate memory allocation
- **Proximity_score(a,b) = p(a,b)*LF^{lns}/CF^{cns}*$\sum_{x=\{a,b\}}$(os(x)*ps(x))**
  - **p** → proximity {Normalized to different expression BW}
  - **LF** → loop factor {10}
  - **CF** → conditional factor {2}
  - **lns** → loop nesting score {1…n}
  - **cns** → conditional nesting score {1…n}
  - **os** → operation score {Normalized to read-only BW}
  - **ps** → pattern score {Normalized to random access BW}

ARENA ALLOCATION



- Clusters of data structures
  - Proximity scores
  - Group them using K-means clustering algorithm
  - Allocate higher scored clusters on HBM

IMPLEMENTATION

- Microbenchmark characterization
- Clang obtains code locations and scope of all data structures and loops
- LLVM opt obtains SSA form and perform analysis
  - Obtain access patterns, memory operations, aliasing and nesting score for all data structures and loops
  - Make allocation decisions
- Shell script to make source-to-source transformations and re-compile

EXPERIMENTATION

- Test on Benchmarks(C, OpenMP, MPI)
  - LULESH
  - VPIC
  - SNAP
  - HPCC
  - KRIPKE
  - CLAMR
  - AMG2013
  - MCB
  - QMCPACK
  - CAM-SE
- Compare the framework with manual allocation for DRAM-HBM system

RELATED WORK

- Khaldi, Dounia, and Barbara Chapman. "Towards automatic HBM allocation using LLVM: a case study with knights landing." *LLVM Compiler Infrastructure in HPC (LLVM-HPC), 2016 Third Workshop on the*. IEEE, 2016.
  - BCDA analysis in LLVM
  - Uses hbw_malloc
- Wang, Haojie, et al. "Spindle: informed memory access monitoring." *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. USENIX} Association}, 2018.
  - Creates a memory monitoring tool using LLVM
- Alvarez, Lluc, et al. "Runtime-Guided Management of Stacked DRAM Memories in Task Parallel Programs." *Proceedings of the 2018 International Conference on Supercomputing*. ACM, 2018.
  - Checks a directory if data is present before access and moves data accordingly

FUTURE WORK

- Finish the framework
  - Handling of complex variables
  - Implement the proximity score calculation
  - Source-to-source translation
- Evaluate on the benchmarks shortlisted
- Write up the paper

THANK YOU

- Questions?