PennState

Pacific Northwest NATIONAL LABORATORY

Impact of Write-Allocate Elimination on Fujitsu A64FX

Yan Kang
The Pennsylvania State University
Pennsylvania, USA

Sayan Ghosh
Pacific Northwest National Laboratory
Washington, USA

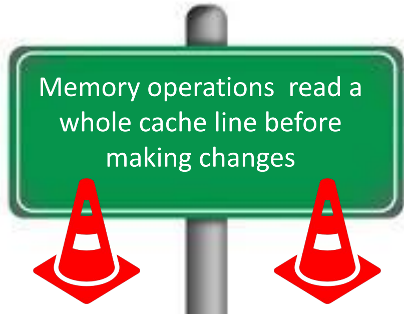
Mahmut Kandemir
The Pennsylvania State University
Pennsylvania, USA

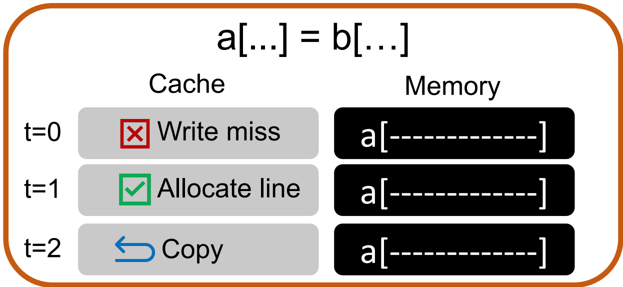
Andrés Marquez
Pacific Northwest National Laboratory
Washington, USA

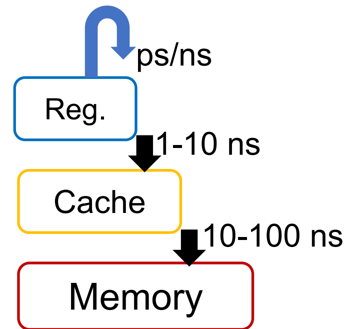
PennState College of Engineering

PennState High Performance Computing Lab

Motivation







If the data is not going to be reused soon, what is the point to cache it?

non-temporal write/stores:

Directly write new data into memory instead of read a cache line then modify it

Write-Allocate : Allocate a cache line on a write miss

Code modifications ???

Compiler automatic ???

Performance improvements ???

Write-Allocate Avoidance

Evasion: [Hardware detects if cache line is going to be overwritten] store cache line directly in memory (Intel, non-temporal stores, compiler hints or automatic SpecI2M)

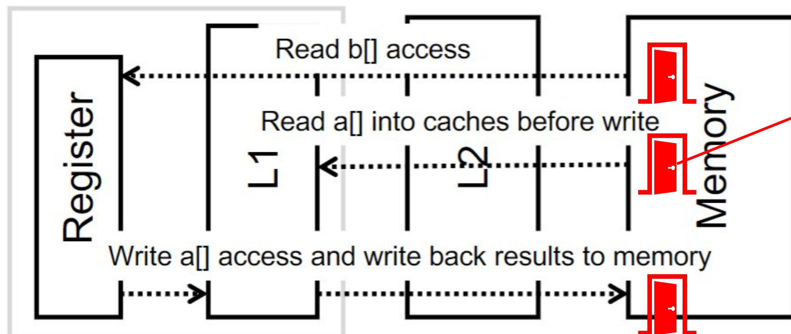
Elimination: [Hardware detects if cache line is going to be overwritten] directly write an L2 cache line with zeroes, processor loads cache line avoiding memory read

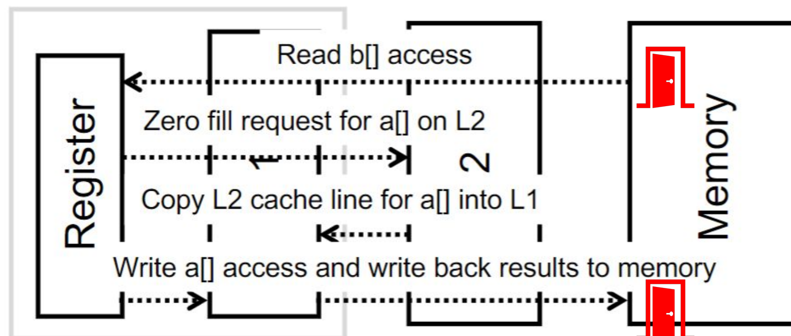
Fujitsu A64FX: Elimination is available through a special 64-bit instruction (DC ZVA) in the ARMv8-A

Can write-allocate elimination via "zero fill" improve the performance of various applications on Fujitsu A64FX?

Read Dr. Georg Hager's blog post and paper:
https://blogs.fau.de/hager/archives/8997
https://onlinelibrary.wiley.com/doi/10.1002/cpe.6512

Zero Filling in Fujitsu A64FX

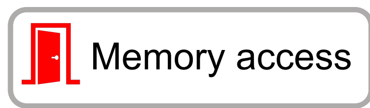




This is the memory access we are trying to eliminate!

"zero fill" on L2 Cache:
Upon receiving the DC ZVA request, the L2 cache secures the cache line corresponding to the specified virtual address and writes zero data

"zero fill" on L1 Cache:
zero data is written after data in the L1 cache is written back to the L2 cache.

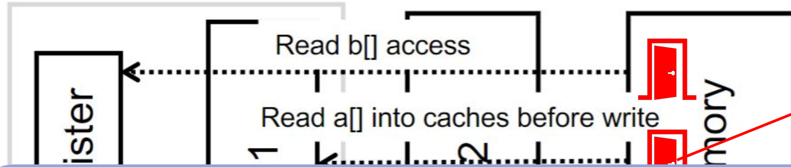


Pacific Northwest NATIONAL LABORATORY

PennState College of Engineering

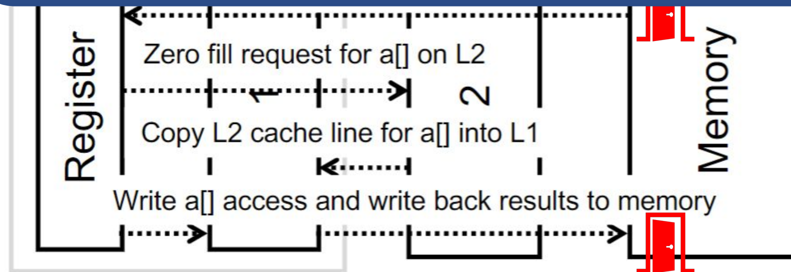
PennState High Performance Computing Lab

Zero Filling in Fujitsu A64FX

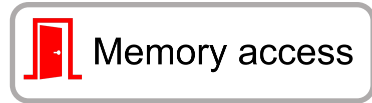


This is the memory access we are trying to eliminate!

the L2



zero data is written after data in the L1 cache is written back to the L2 cache.

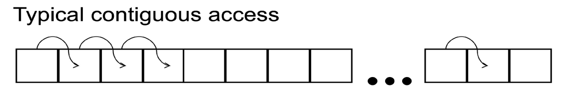


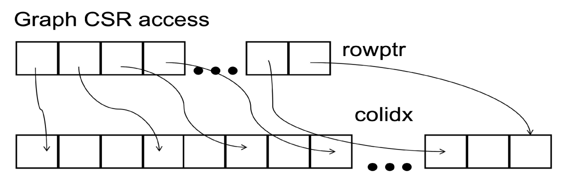
Saving memory traffic means improving memory b/w, what's that benchmark to study "sustainable memory b/w"?

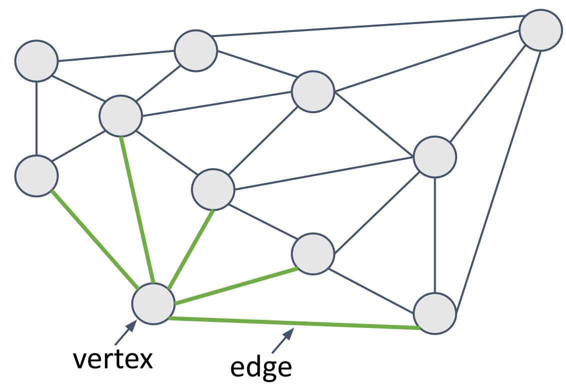
Pacific Northwest
NATIONAL LABORATORY

Benchmarking decisions

- STREAM is "best case" memory b/w benchmark
  - Does not represent irregular cases, most applications
- Graphs – irregular memory accesses
  - Applications perform repetitive *neighborhood accesses*
- NEVE is a benchmark, like STREAM for graphs (has COPY, SUM and MAX) - |V|*|E|*#ops / t







Can return MB/s!

STREAM		Graph Neighborhood Access Kernels	
name	kernel	name	kernel
COPY:	a(i) = b(i)	Neighbor Copy:	$a(i, j) \leftarrow COPY(\forall_{i\in\|V\|}\forall_{j\in neighbor(i)} weight(j))$
SCALE:	a(i) = q*b(i)	Neighbor Add:	$b(i) \leftarrow SUM(\forall_{i\in\|V\|}\forall_{j\in neighbor(i)} weight(j))$
SUM:	a(i) = b(i) + c(i)	Neighbor Max:	$c(i) \leftarrow MAX(\forall_{i\in\|V\|}\forall_{j\in neighbor(i)} weight(j))$
TRIAD:	a(i) = b(i) + q*c(i)		

Sayan Ghosh, Nathan R. Tallent, and Mahantesh Halappanavar. "Characterizing performance of graph neighborhood communication patterns." *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2021): 915-928.

PennState College of Engineering

PennState High Performance Computing Lab

Explicit “Zero Fill” formulation for graph neighborhood accesses

```
static const int DISTANCE = 100;
static const int ELEMS_CACHE_LINE = 256 / sizeof(double);
static const int OFFSET = DISTANCE * ELEM_CACHE_LINE;

static inline void zfill(double * a) {
  asm volatile("dc zva, %0": : "r"(a));
}

#pragma omp parallel
{
  int const tid = omp_get_thread_num();
  int const nthreads = omp_get_num_threads();
  int chunk = nvertices / nthreads;
  double* const zfill_limit = c + (tid+1)*chunk - OFFSET;

  #pragma omp for schedule(static)
  for (int j=0; j<nvertices; j+=ELEMS_CACHE_LINE) {
    int const * __restrict__ const jrowptr = rowptr + j;
    double * __restrict__ const jbuf = buf + j;
    double sum = 0.0;

    if (jbuf+OFFSET < zfill_limit)
      zfill(jbuf+OFFSET);

    for (int i=0; i<ELEMS_CACHE_LINE; ++i) {
      for (int e=jrowptr[i]; e<jrowptr[i+1]; ++e) {
        sum += colidx[e].weight;
      }
      jbuf[i] = sum;
    }
  } // loop over vertices
}   // openmp
```

Explicit assembly to invoke DC ZVA

Each thread works on fixed chunk of iterations over |V| (work is variable)

Block outermost loop over vertices

Invoke zero fill in strides larger than L2 prefetch distance

Inner loop, where the zfill virtual address will be invoked several times (trip count unknown)

Pacific Northwest NATIONAL LABORATORY
PennState College of Engineering
PennState High Performance Computing Lab

Pacific Northwest
NATIONAL LABORATORY

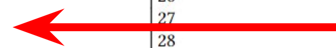
Explicit “Zero Fill” formulation for graph neighborhood accesses

```
static const int DISTANCE = 100;                                1
static const int ELEMS_CACHE_LINE = 256 / sizeof(double);      2
static const int OFFSET = DISTANCE * ELEM_CACHE_LINE;           3
                                                                4
static inline void zfill(double * a) {                          5
  asm volatile("dc zva, %0": : "r"(a));                         6
}                                                               7
                                                                8
#pragma omp parallel                                            9
{                                                               10
...
    double sum = 0.0;                                           20
                                                                21
    if (jbuf+OFFSET < zfill_limit)                              22
      zfill(jbuf+OFFSET);                                       23
                                                                24
    for (int i=0; i<ELEMS_CACHE_LINE; ++i) {                    25
      for (int e=jrowptr[i]; e<jrowptr[i+1]; ++e) {             26
        sum += colidx[e].weight;                                27
      }                                                         28
      jbuf[i] = sum;                                            29
    }                                                           30
  } // loop over vertices                                       31
}   // openmp                                                   32
```

Explicit assembly to invoke DC ZVA

over vertices

Invoke zero fill in strides larger than L2 prefetch distance



Inner loop, where the zfill virtual address will be invoked several times (trip count unknown)

"Zero filling" may not yield performance benefits for irregular graph workloads unless the number of vertices in a graph is significantly larger than the cache line size and the standard deviation of the vertex degrees is relatively low.

PennState College of Engineering

PennState High Performance Computing Lab

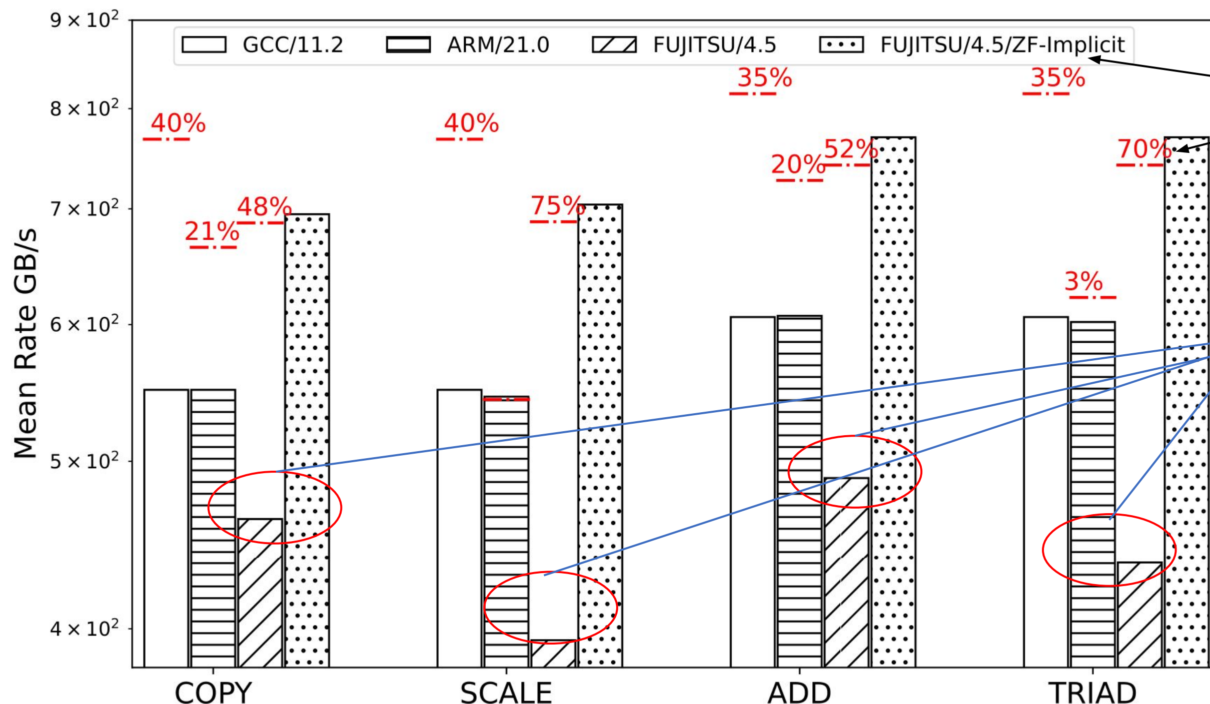
Benchmarks and applications for evaluations

Benchmark Scenarios	Tested Kernels
STREAM	Copy
	Scale
	Add
	Triad
Graph Neighborhood Kernels(NEVE)	Add
	Copy
	Max

Expecting STREAM to be the best case!

Application scenarios	Targeted kernels
Graph500 Breadth First Search	Next frontier list update is similar to graph neighborhood Copy
Louvain graph clustering	Modularity computation requires summing data, similar to graph neighborhood Add.
GAP benchmark suite	
Breadth First Search (BFS)	Next frontier list update is similar to graph neighborhood Copy.
PageRank (PR and PR(SPMV))	Score update is similar to STREAM Copy.
Connected Components (CC and CC(SV))	Singleton partition assignment is similar to STREAM Copy.
Betweenness Centrality (BC)	Aggregation of betweenness scores similar to graph neighborhood Add.
Rodinia benchmark suite	
HotSpot & HotSpot3D	Grid cell update of tiled 2D/3D transient iterative solver similar to STREAM Copy.
LavaMD	Particle updates in the home box similar to STREAM Add.
SRAD(V2)	Image data update is similar to STREAM Add.
Needleman-Wunsch (NW)	Global copy from local reference is similar to STREAM Copy.

STREAM benchmark evaluations (GCC, ARM and FCC)



Fujitsu has a compiler option (-Kzfill), referred as implicit version [does not work for C++ compiler]

Fujitsu is showing a huge performance gap comparing with GCC/ARM ???

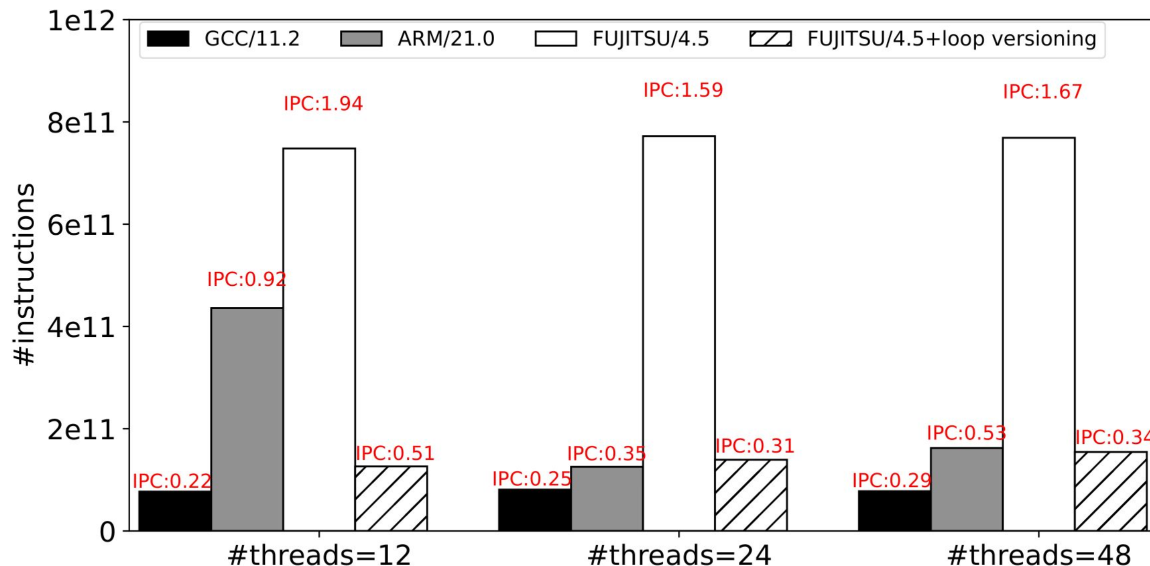
All compilers demonstrate improvements, FCC up to 70%!

Pacific Northwest NATIONAL LABORATORY

PennState College of Engineering

PennState High Performance Computing Lab

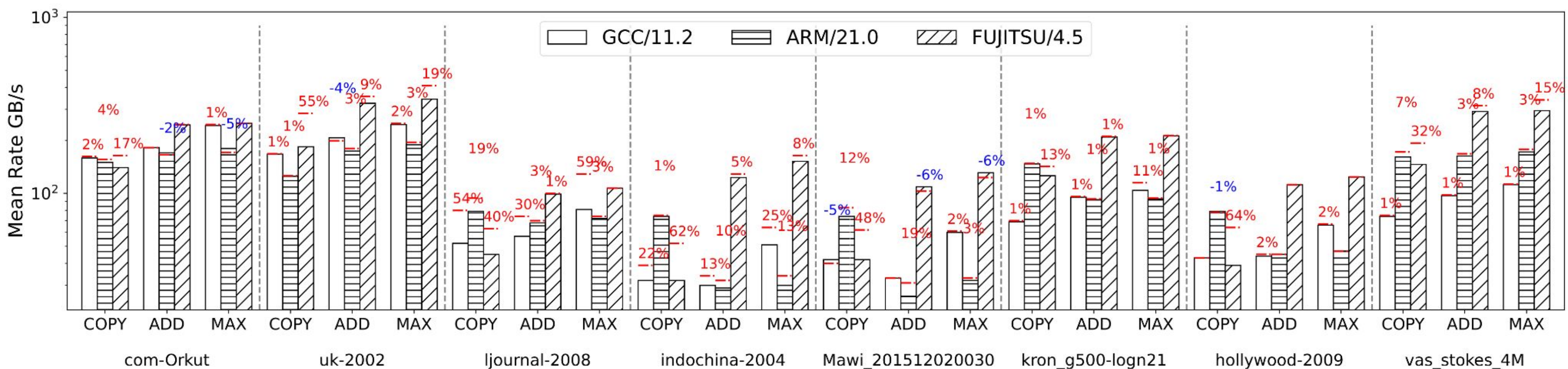
Loop versioning for STREAM



Enabling loop versioning option allows the compiler to perform software pipelining optimizations, bringing the overall performance close to ARM/GCC.

Pacific Northwest
NATIONAL LABORATORY

Graph benchmark evaluations (GCC, ARM and FCC)

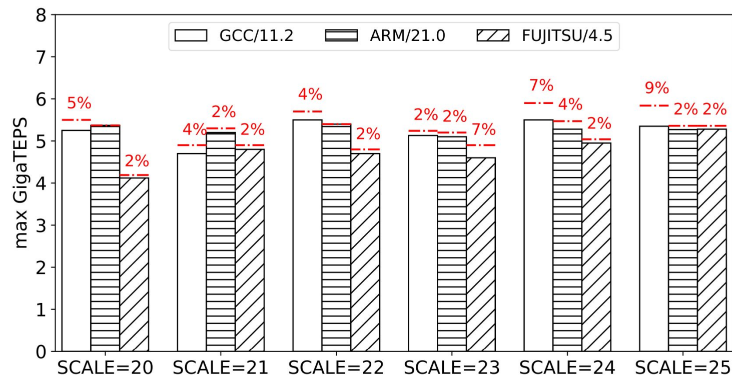


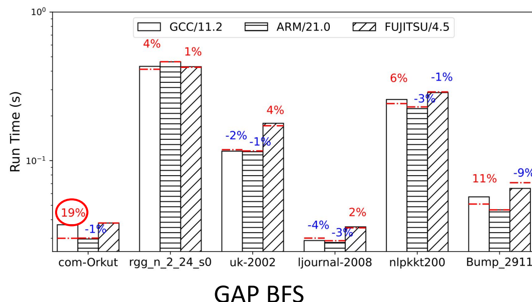
- Used different graphs – implies different structure/work-per-loop
- ZFILL: degradation of up to 6% but also up to 64% improvement (FCC)
- Since the “zero fill” stride length can be greater than the median #edges for certain graphs, it can have a limited impact and, in some cases, may incur overheads
- No compiler automatic DC ZVA guaranteed

PennState
College of Engineering

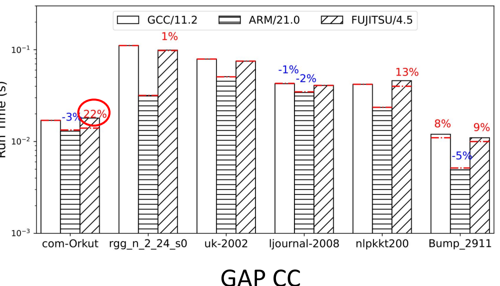
PennState
High Performance
Computing Lab

Graph Application Evaluations

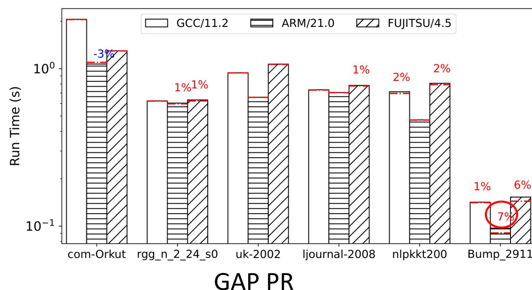




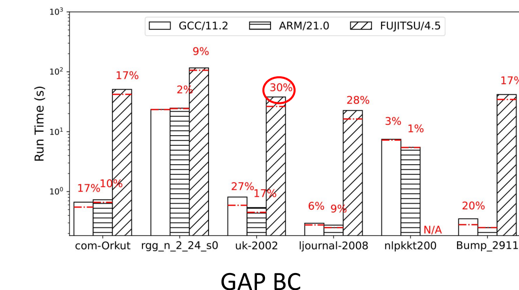
GAP BFS



GAP CC



GAP PR



GAP BC

- Does not improve performance where there is limited work in the ZFILL section
- ~15% improvement when there is sufficient work
- No compiler automatic DC ZVA guaranteed

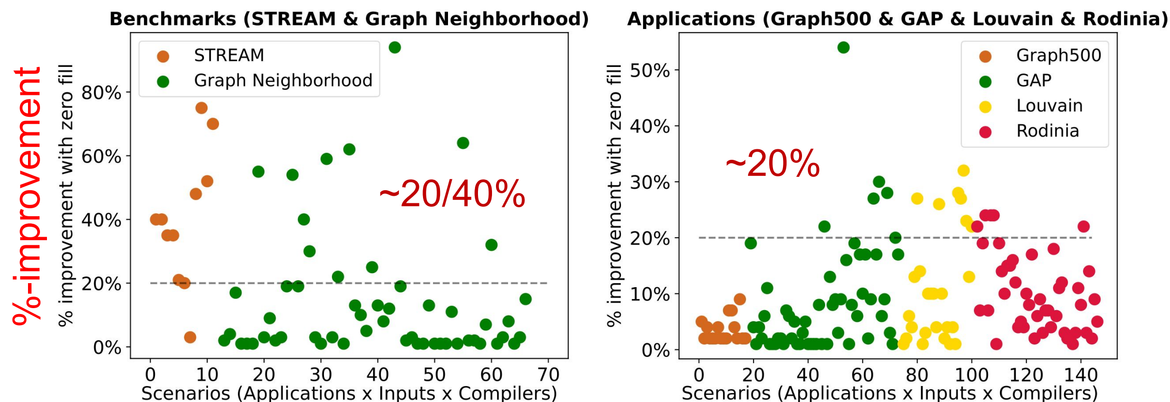
Non-temporal store patterns may not positively impact overall application execution times if they are not on the critical performance path.

PennState College of Engineering

PennState High Performance Computing Lab

Observations

- NEVE exhibit about 2–5x performance degradation compared to STREAM
- End-to-end improvements between 5-20% for benchmarks and diverse application scenarios due to "zero fill" adaptations
- Performance improvements of up to 32% in Louvain clustering and median improvements of 5–17% in GAP PR, CC, and BC benchmarks



Pacific Northwest
NATIONAL LABORATORY

Limitations of Zero Filling

- Number of vertices in a graph may not larger than the cache line size.
- Regular prefetching may benefit more for short buffer streaming write.
- Non-temporal store may not be on the critical performance path.
- Applications may not be written to exploit non-temporal stores.

Summary & Future Steps

Demonstrated the impact of write allocate elimination on A64fx for various applications and show cased the improvement brought by “Zero fill”

Identified possible causes might lead to our observations on the varied performance improvements, such as software pipelining optimization, SIMD extensions, etc.

What exactly reasons behind each or all applications variations on A64FX ???

In what condition it maximize improvement?

Where to apply?

In what condition it requires minimal modifications ?

Compiler automatic DC ZVA generation?

Pacific Northwest
NATIONAL LABORATORY

Acknowledgements

- PNNL LDRD Data Model-Convergence (PI: Sayan Ghosh, PNNL)
- DOE ASCR Advanced Memory to Support Artificial Intelligence for Science (AIAMS, PI: Andrés Márquez, PNNL)
- Penn State HPCL (Prof. Mahmut Kandemir)
- Ookami testbed support (Dr. Eva Siegmann and team, SBU)
- IWAHPCE'24 paper reviewers

PennState
College of Engineering

PennState
High Performance
Computing Lab