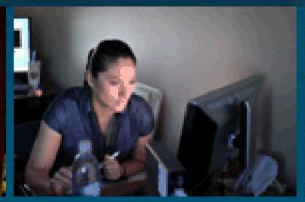
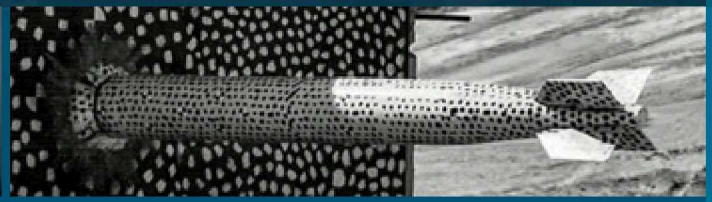
SAND2019-4451C

Heterogeneous Accelerators of the Memory, by the Memory, and for the Memory





Driving and Harnessing Extreme Heterogeneity

PRESENTED BY

A. Rodrigues(SNL)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Motivation

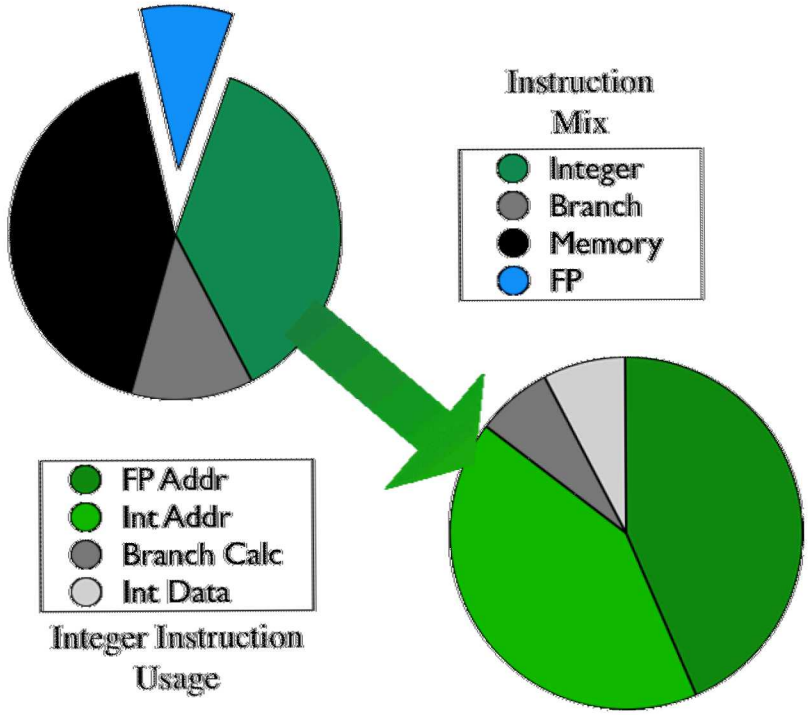
- Entering an Era of Extreme Heterogeneity
- New possibilities in accelerators
- New solutions, custom crafted to challenging problems

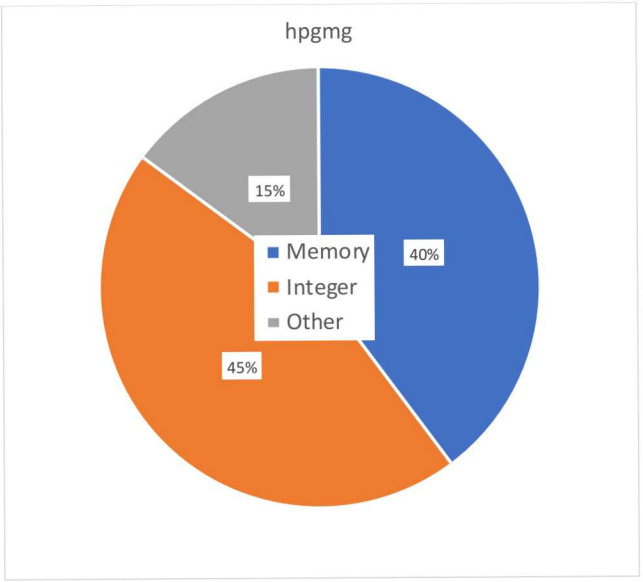
- All our old problems still remain

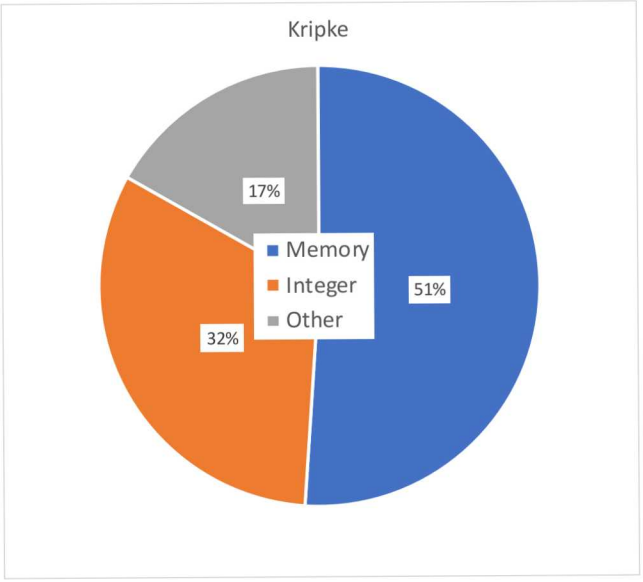
- Most of our old problems are memory

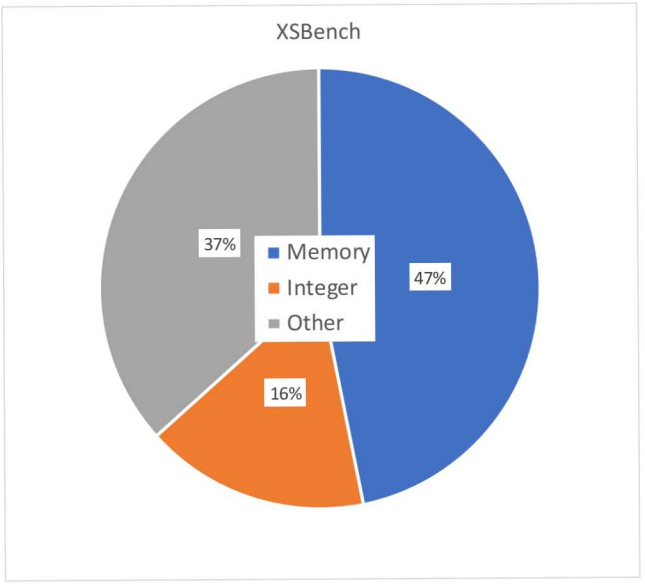
Memory is what our programs DO…

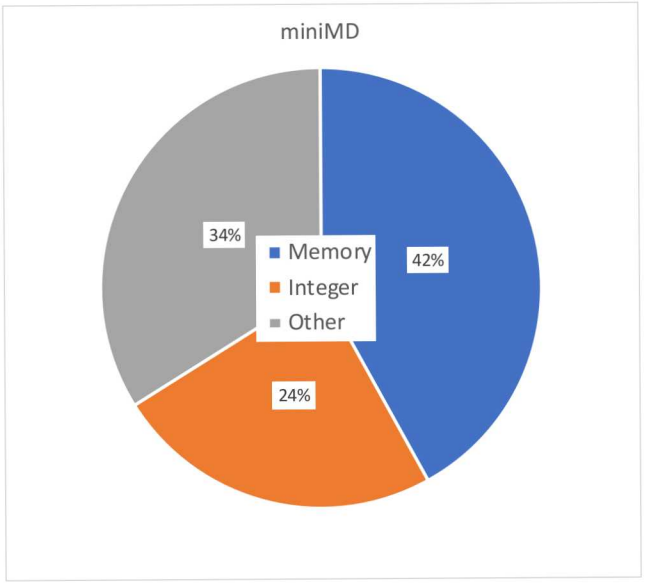
- Dynamic Instruction Mix
  - ~40-50% Memory
  - ~20-40% Integer
  - ~15-40% Other (Branch, FP, etc…)
- Most Integer instructions calculating memory addresses





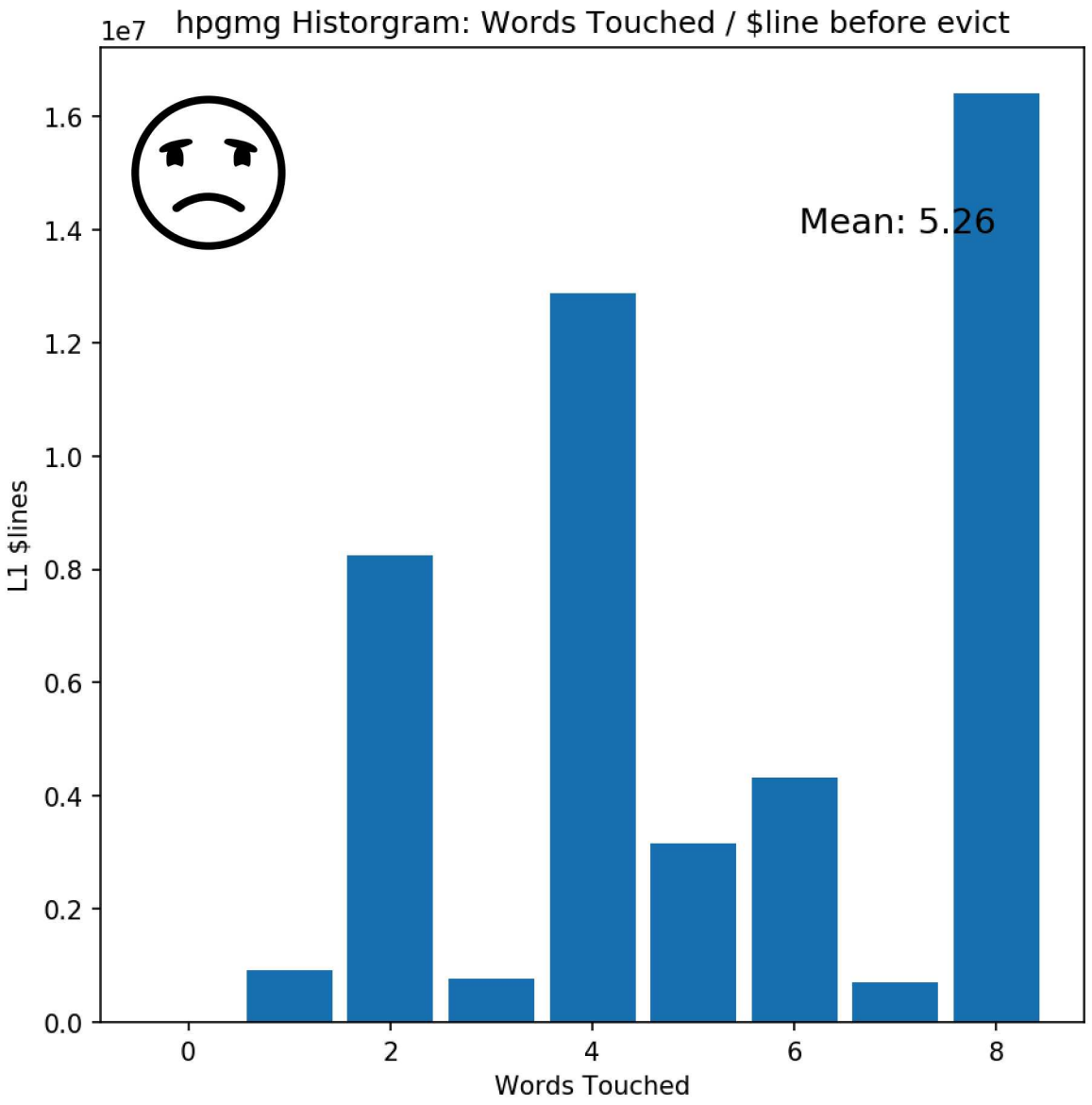


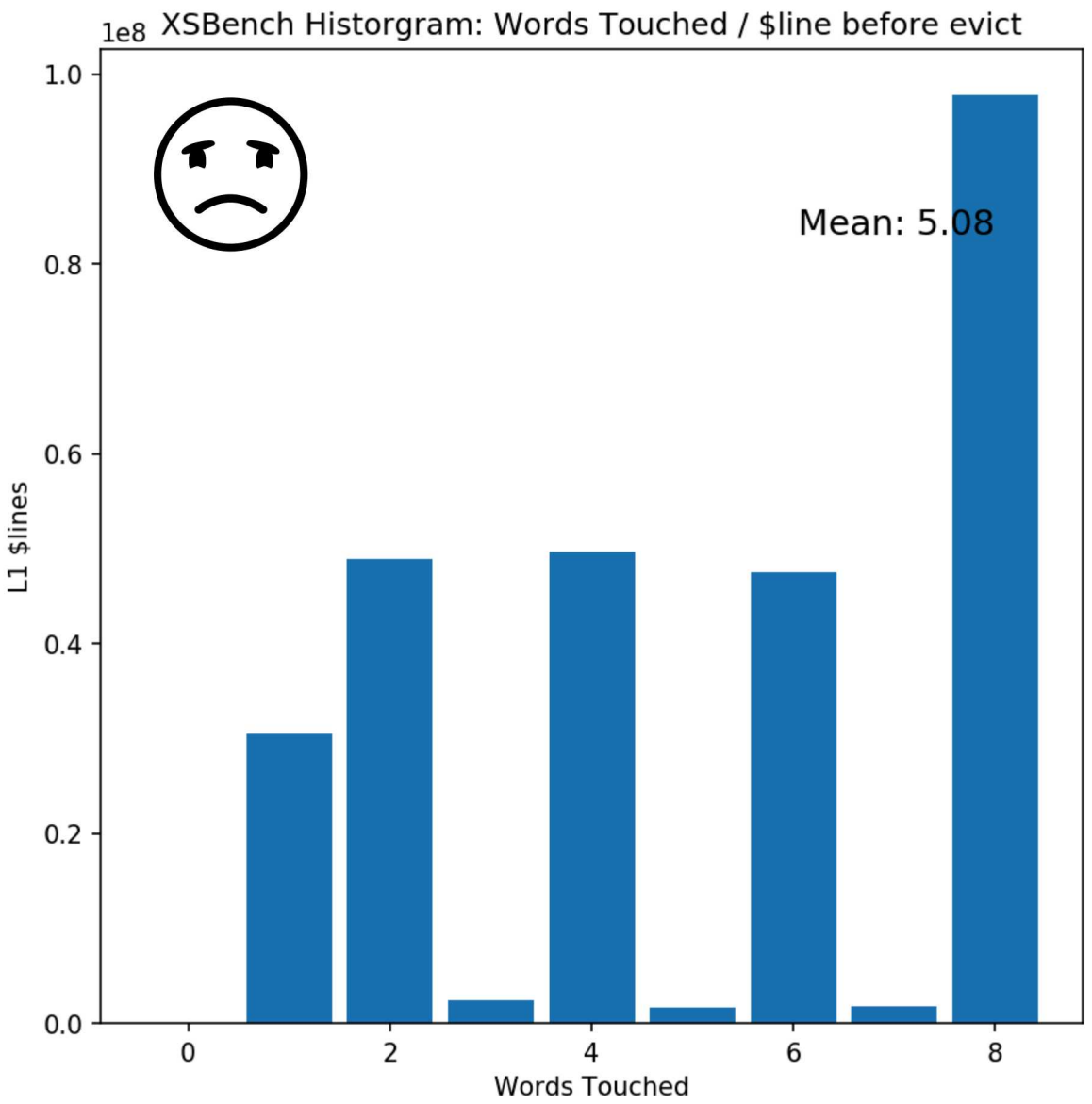




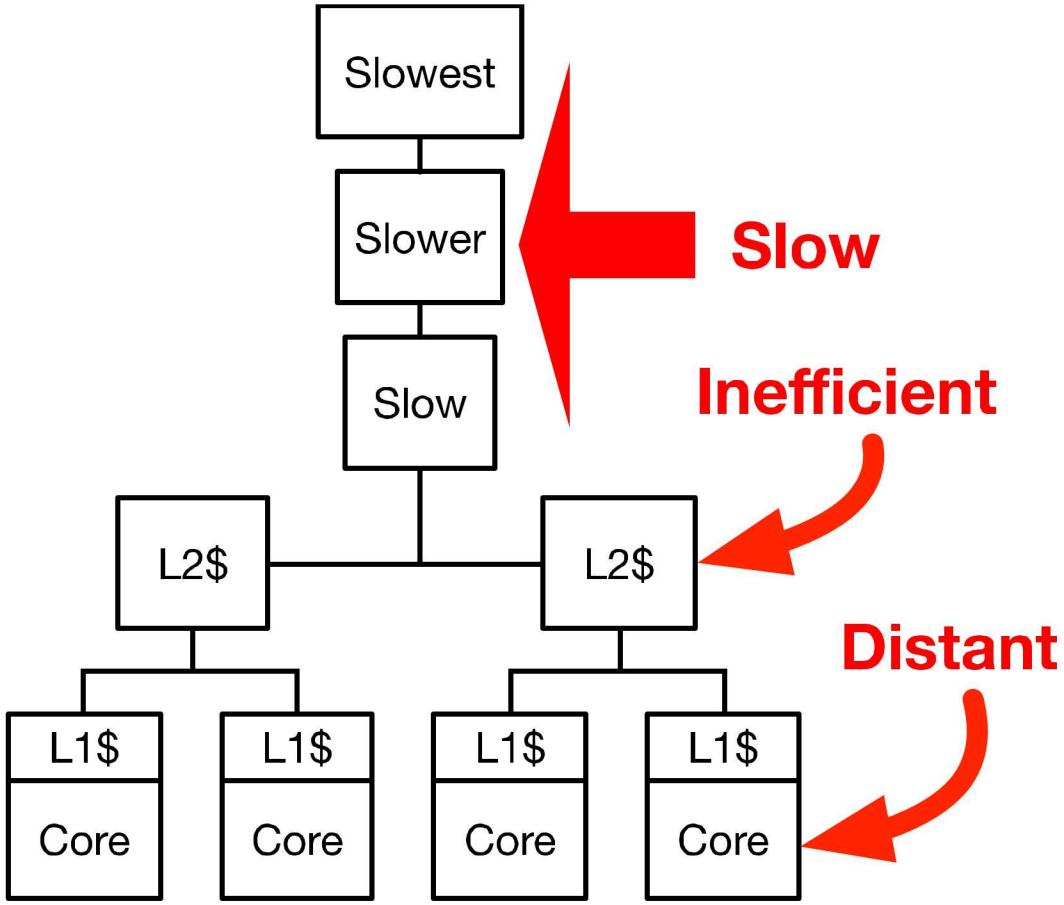
Sandia National Laboratories

…and we don't do it well



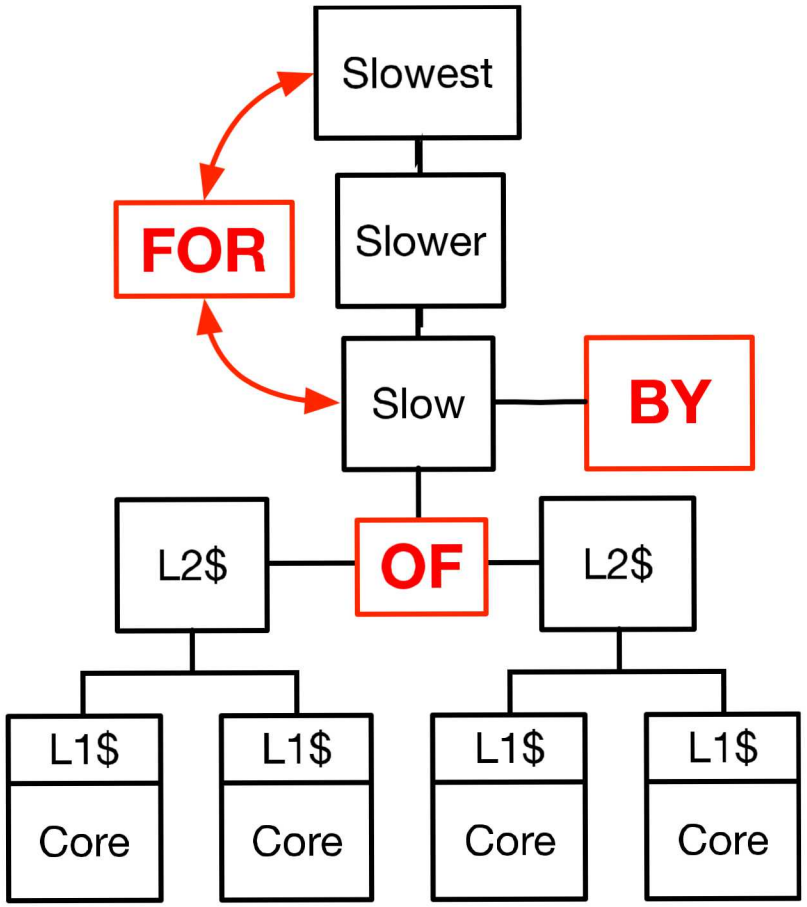


What we have: Slow, Inefficient, Distant



- Main Memory (esp. tiered memory) is slow
  - Often limiting factor
- Caches are inefficient
  - Many applications throw away most of the cache line
- Processors are distant
  - High latency, limited BW to memory

What we need: Of the Memory, By the Memory, For the Memory



- …**Of** the Memory
  - Make better use of scare cachelines, bandwidth
- …**By** the Memory
  - Perform operations closer to the memory
- …**For** the Memory
  - Make the memory system (appear) faster by continual optimization

Why didn't it work **Before**?: The Usual Suspects

- Cost
  - Extra Silicon too expensive
  - Vendors less interested in custom solutions
- Closed Ecosystems
  - ARM not viable for HPC
  - RISC-V not around
- Programmability
  - Chicken & Egg
  - Why customize for a one-off?

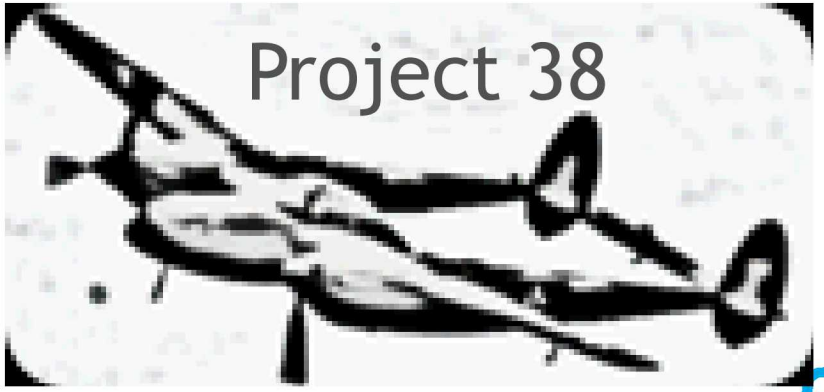
Why it will work **NOW**

- Cost
  - Implementations require less Silicon
  - Vendors looking for customization / differentiation
- Ecosystem
  - ARM
  - RISC-V
- Programmability
  - Still difficult
  - GPUs forcing a shift
  - More interest in flexible data programming (e.g. Raja, Kokkos, OpenMP extensions, C++ Standard)

…Of the Memory

Project 38

- Why?
  - Budgets always tight, need to share resources, understand each other's constraints
  - Discussions outside a specific procurement / existing project
  - Identify potentially useful architectures
  - Areas of interest / overlap
  - Guide future collaboration
  - Guide future interactions with vendors
- Architectures
  - Scatter/Gather (Broadly)
  - Scratchpad memories
  - Atomics



Key Institutions

BERKELEY LAB

Pacific Northwest NATIONAL LABORATORY

Sandia National Laboratories

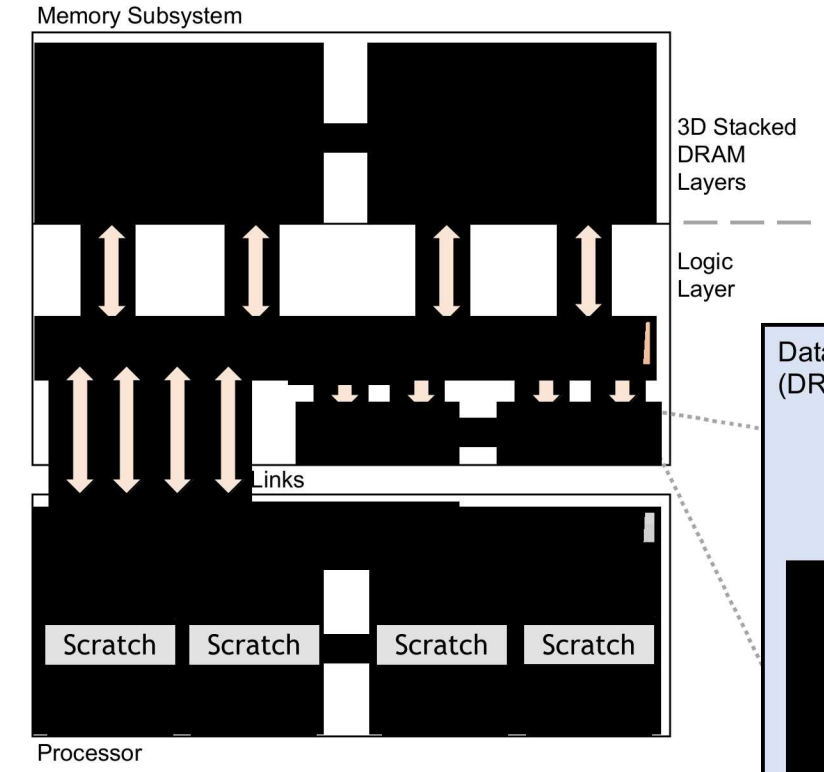
Lawrence Livermore National Laboratory

BROOKHAVEN NATIONAL LABORATORY

Argonne NATIONAL LABORATORY

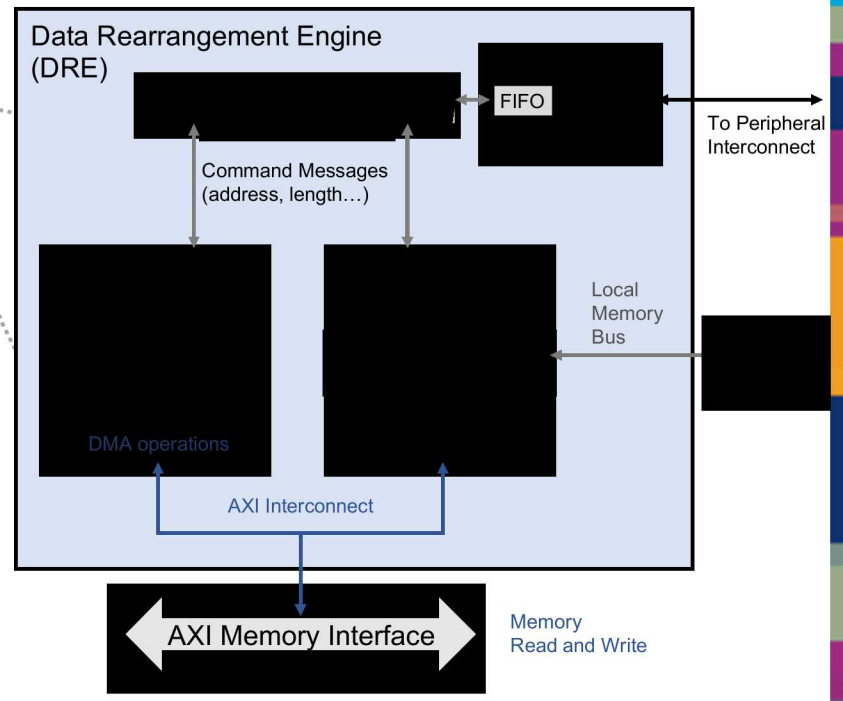
Architecture

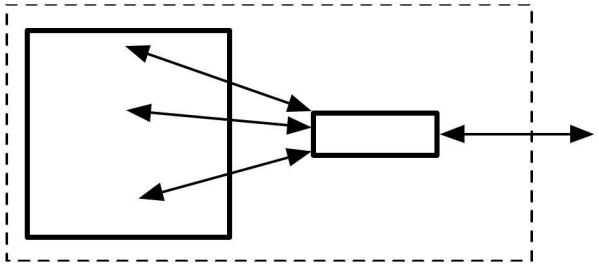
- Key features
  - Flexible architecture
  - Can target scratchpad
  - In/Near Memory
  - Programmable
- Abilities
  - Pointer chase / indirection
  - Sparse -> dense
  - Complex data structures

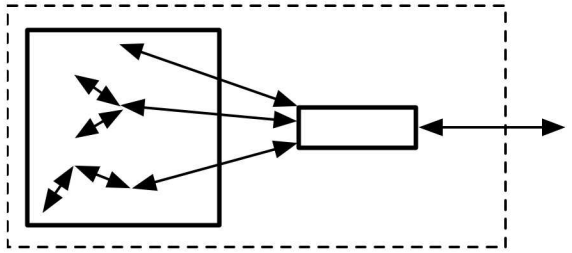


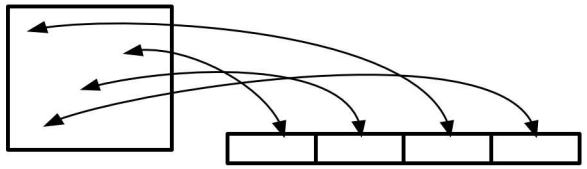
Lawrence Livermore National Laboratory

Gohkale, Lloyd DRE









Sandia National Laboratories

Initial Applications: Kripke, hpgmg, (XSBench)

- From Initial set
- Recommended by steering committee as important
  - Also, XSBench
- Difficult! High bar:
  - Kripke: 98% L1 hit, high L2, multiple "hot" s/g regions, ~70% accesses in s/g, thread local
  - hpgmg: 98-99% L1 hit
  - XSBench: upto 5% L1 miss, 30% s/g, complex s/g region
- If we can show improvement here, we can probably improve many apps

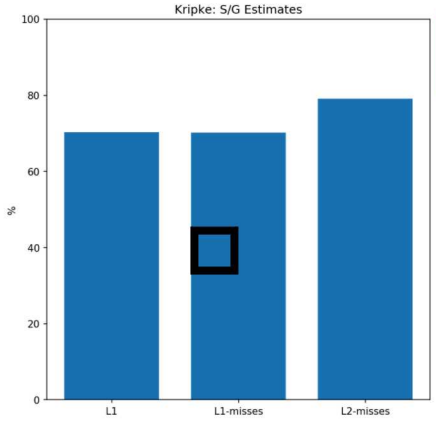
```
Kernel_3d_DGZ.cpp:
void Kernel_3d_DGZ::LTimes():
foreach (subdomain){
 foreach (moment){
  foreach (direction) {
   for(int gz = 0;gz < num_locgz; ++ gz){
    phi_nm[gz] += ell_nm_d * psi_d[gz];
  }}}}
Kernel_3d_DGZ::LPlusTimes():
foreach (subdomain * moment * direction * gz)
  rhs_d[gz] += ell_plus_d_nm * phi_out_nm[gz];
void Kernel_3d_DGZ::scattering():
foreach (subdomain * moment * group * group *
zone * mix) {
  int material = mixed_material[mix];
  double fraction = mixed_fraction[mix];
  phi_out_nm_gp_z += sigs_n_g_gp[material] *
   phi_nm_g[zone] * fraction;
}
```

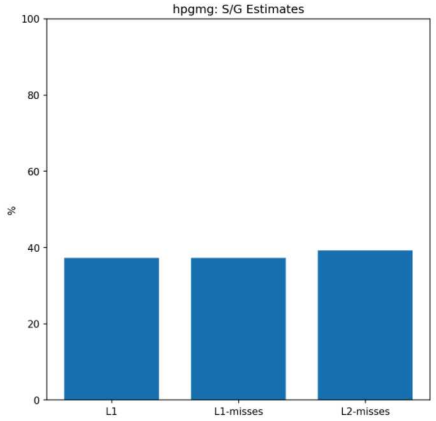
```
finite-volume/source/operators/gsrb.c:
void smooth(…){
…;
for(k=klo;k<khi;k++){
  for(j=jlo;j<jhi;j++){
    for(i=X;i<ihi;i+=2){ //stride-2 GSRB
      int ijk = i + j*jStride + k*kStride;
      double Ax = apply_op_ijk(x_n);
      x_np1[ijk] = x_n[ijk] +
                   Dinv[ijk]*
                   (rhs[ijk]-Ax);
    }
  }
}
```

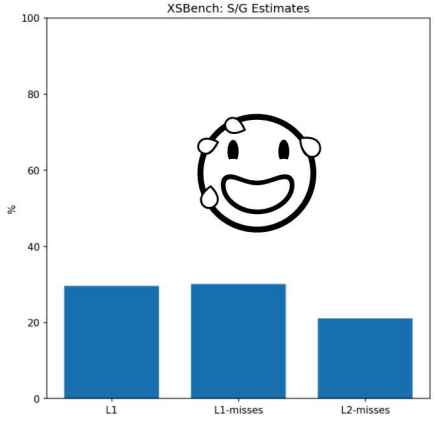
```
CalculateXS.c: calculate_micro_xs():

  low =
    &nuclide_grids[nuc][energy_grid[idx]
    .xs_ptrs[nuc] - 1];

calculate_macro_xs():
  p_nuc = mats[mat][j];
  conc = concs[mat][j];
```



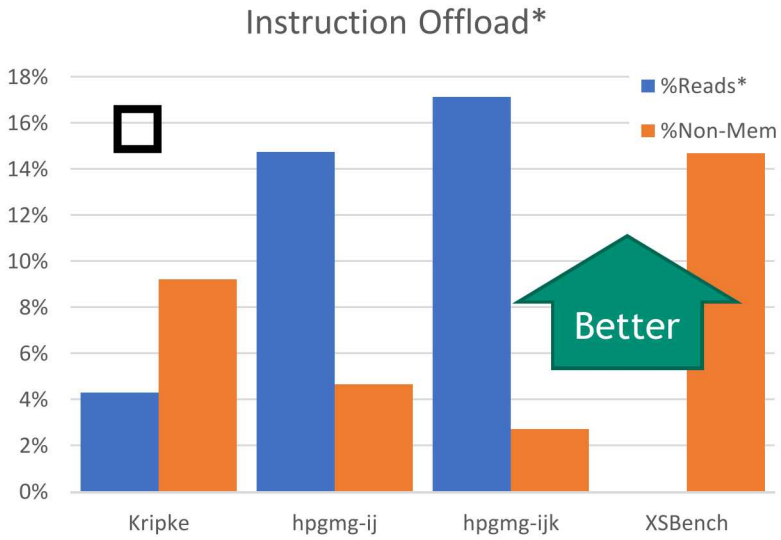


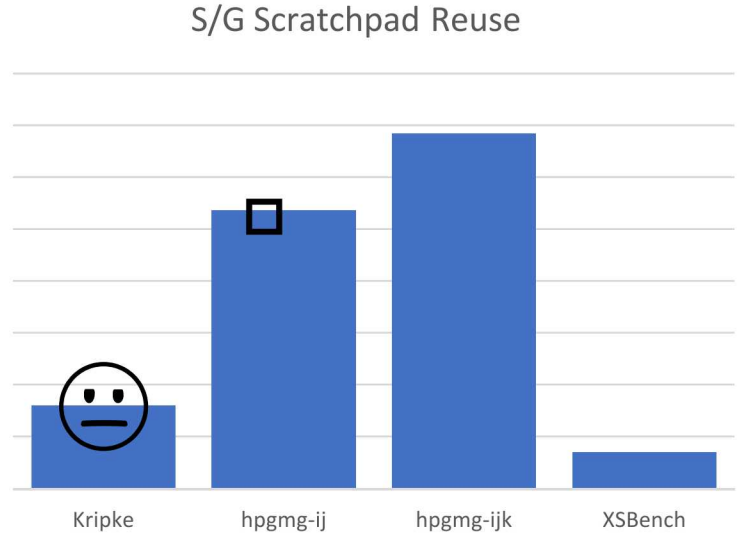


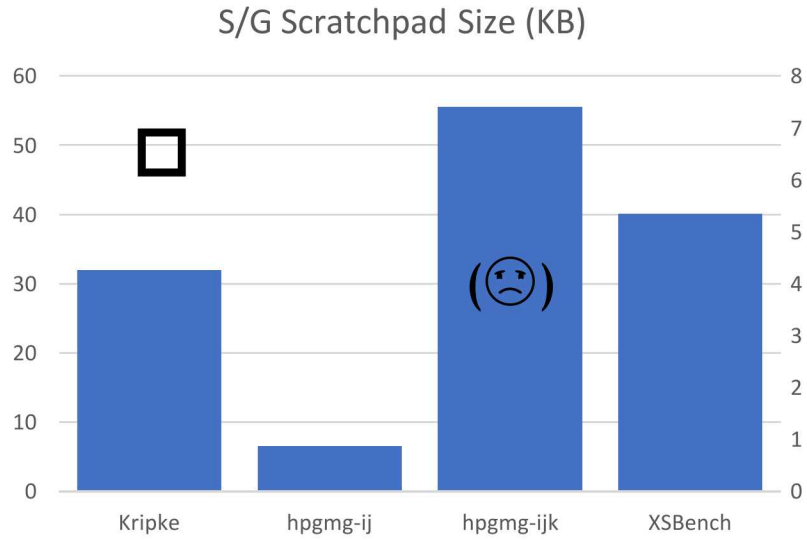
ndia
ational
boratories

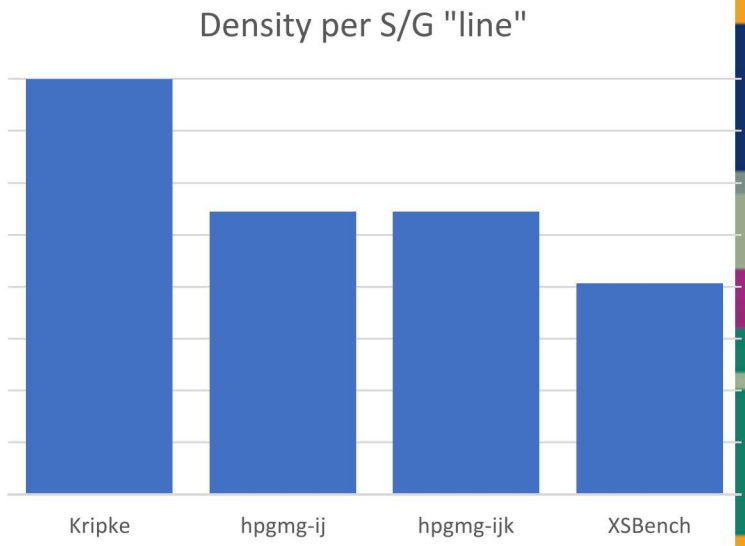
S/G to Scratchpad

- Identify regions to S/G to scratchpad, offload
- Performance decrease!
  - L2 Cache misses increase
  - Simulation artifacts
- Causes for Hope:
  - Able to offload substantial % of integer operations
  - Good reuse of data when in scratchpad
  - Scratchpad sizes reasonable (L1 sized)
  - S/G data less "dense" – can be smarter / more efficient than Cache





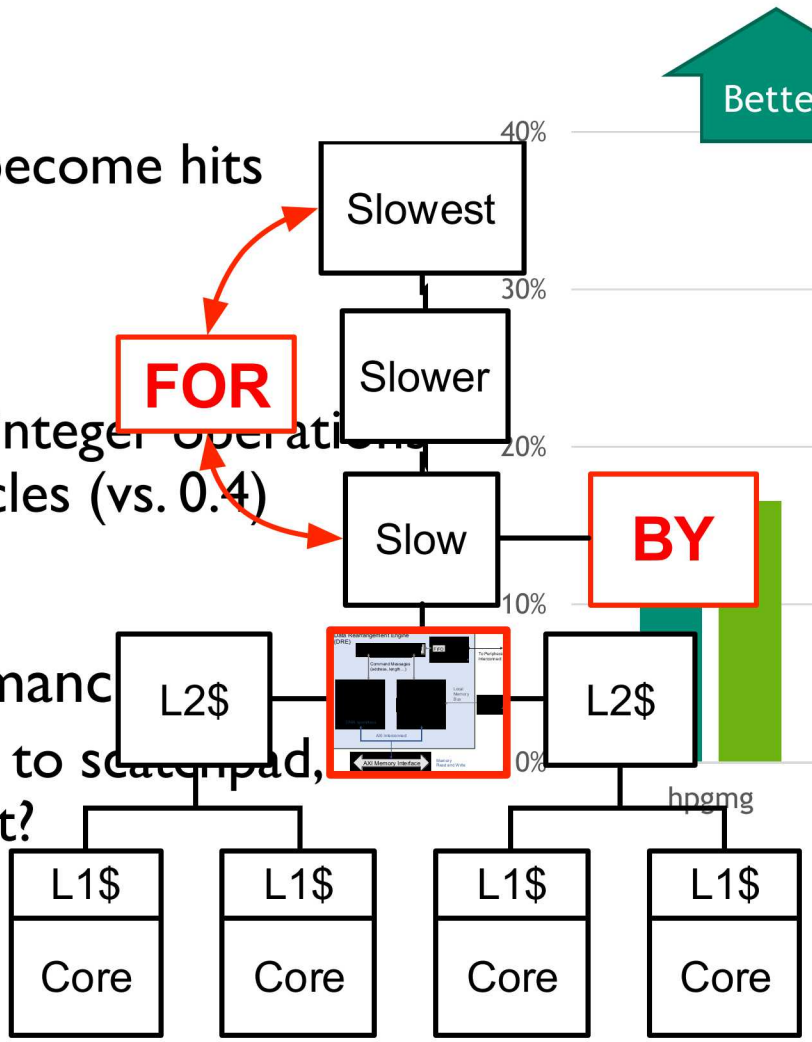


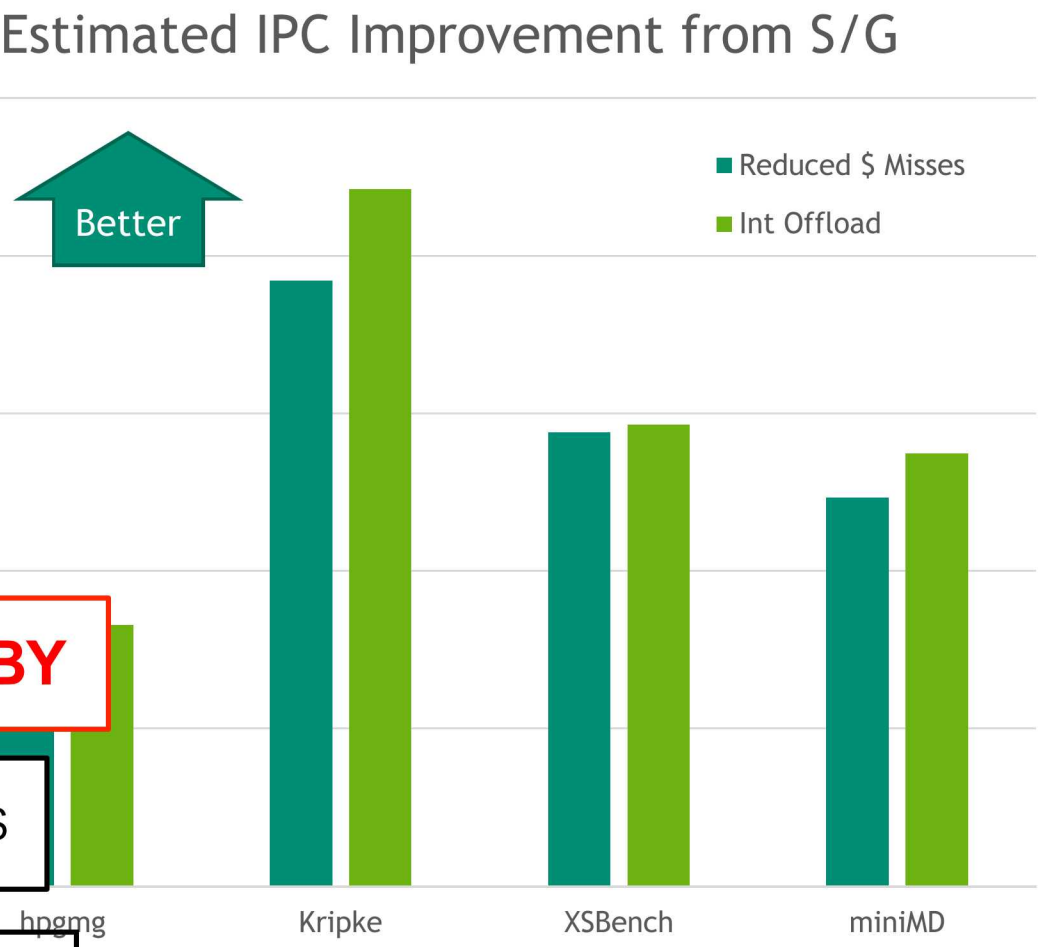


Sandia National Laboratories

Analysis (Motive)

- Simple IPC Calculation
  - Lots of caveats
- Effect 1: Reduce $ Misses
  - Assume covered L1 misses become hits
  - 13-40% IPC improvement
- Effect 2: Integer offload
  - Assume half of address-calc integer operations are offloaded, require 0.1 cycles (vs. 0.4)
  - 17-44% IPC Improvement
- Effect 3: Improved $ performance
  - If covered cache accesses go to scratchpad, other accesses more efficient?



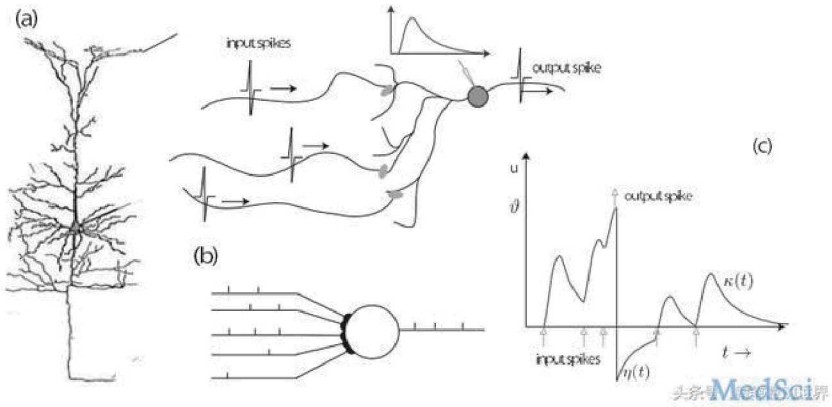


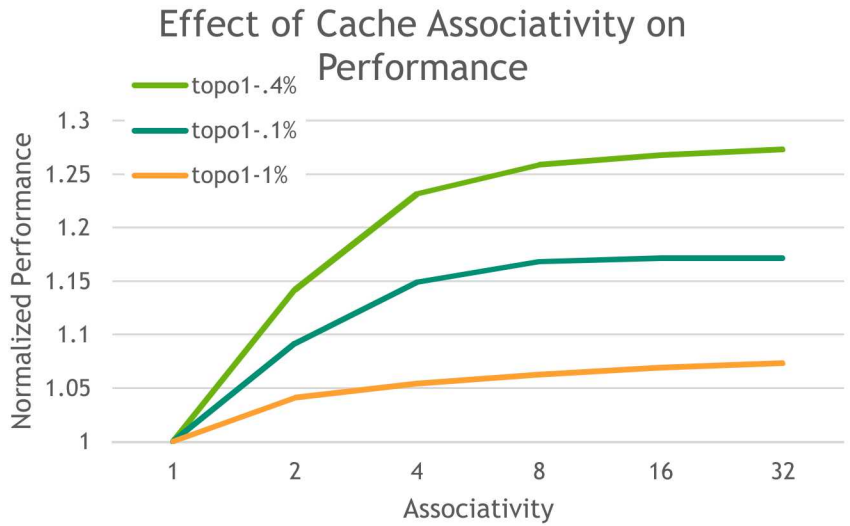
Sandia National Laboratories

…By the Memory

Case Study: Spiking Neural Network

- More closely mimic natural biological neural networks
  - Neurons fire when membrane potential reaches threshold
- Simple Neuron Model
  - Potential, threshold (data)
  - Accumulate spikes (I/O)
  - ”Leak” potential (compute)
  - Fire (I/O)
- Very data dependent
- Low compute
- Performance is very topology dependent

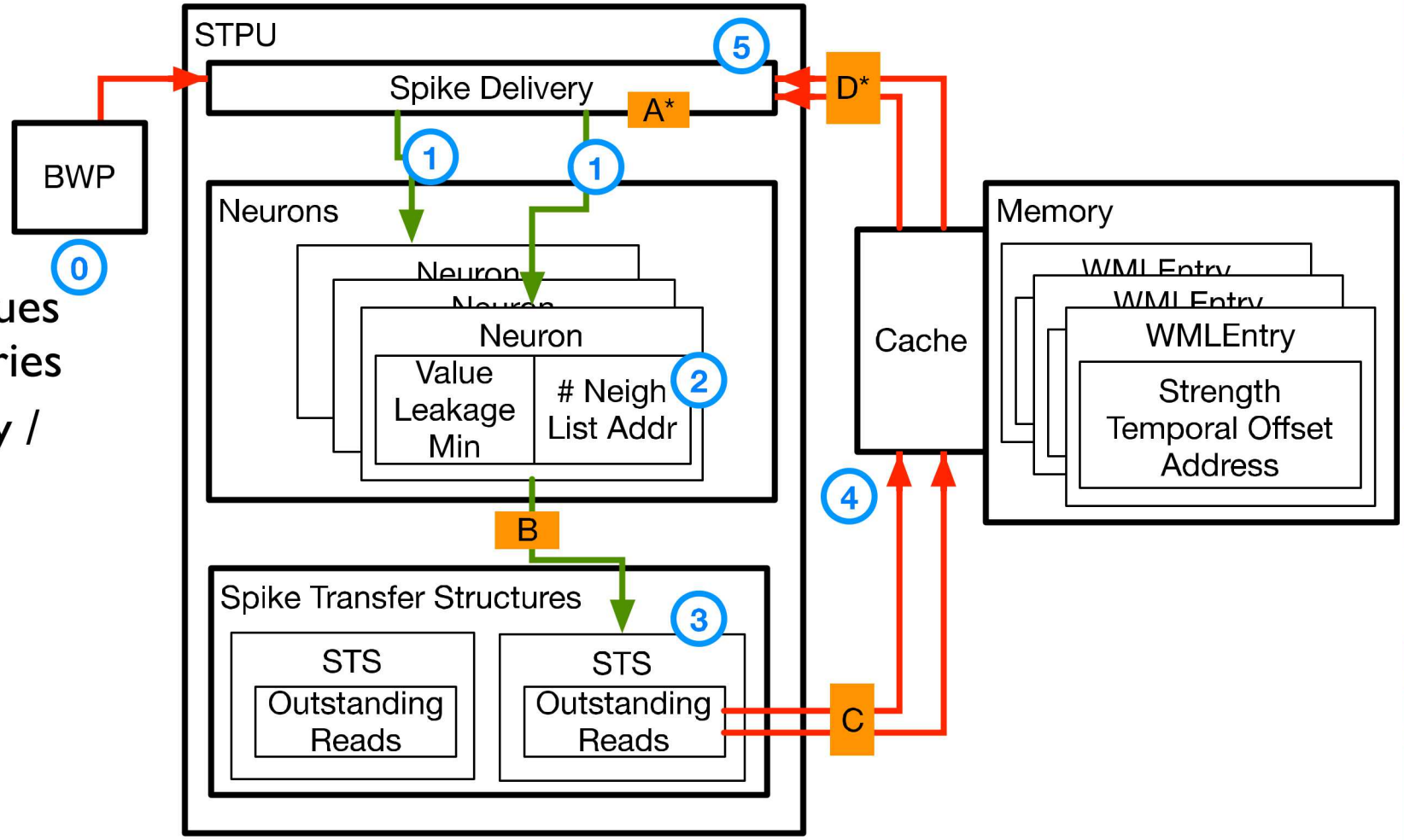




Sandia National Laboratories

Sandia Spiking Temporal Processing Unit

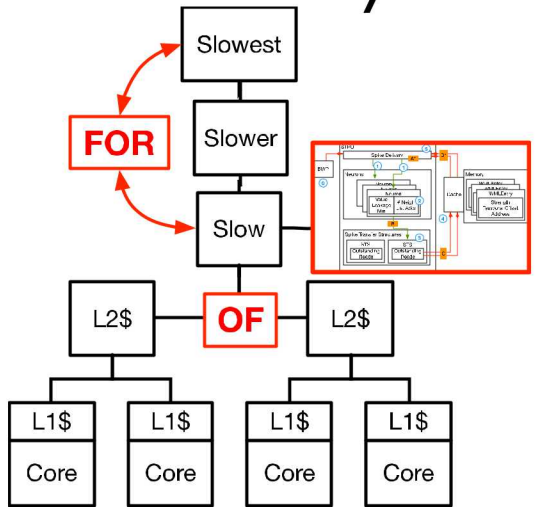
- Process
  - 0: 'Brain Wave Pulse' input
  - 1: Spike delivered to Neuron(s)
  - 2: LIF, possible neuron spike
  - 3: Spike assigned to STS, STS issues reads for White Matter List entries
  - 4: WMLEs fetched from memory / cache
  - 5: WMLEs converted to spikes; goto (1)

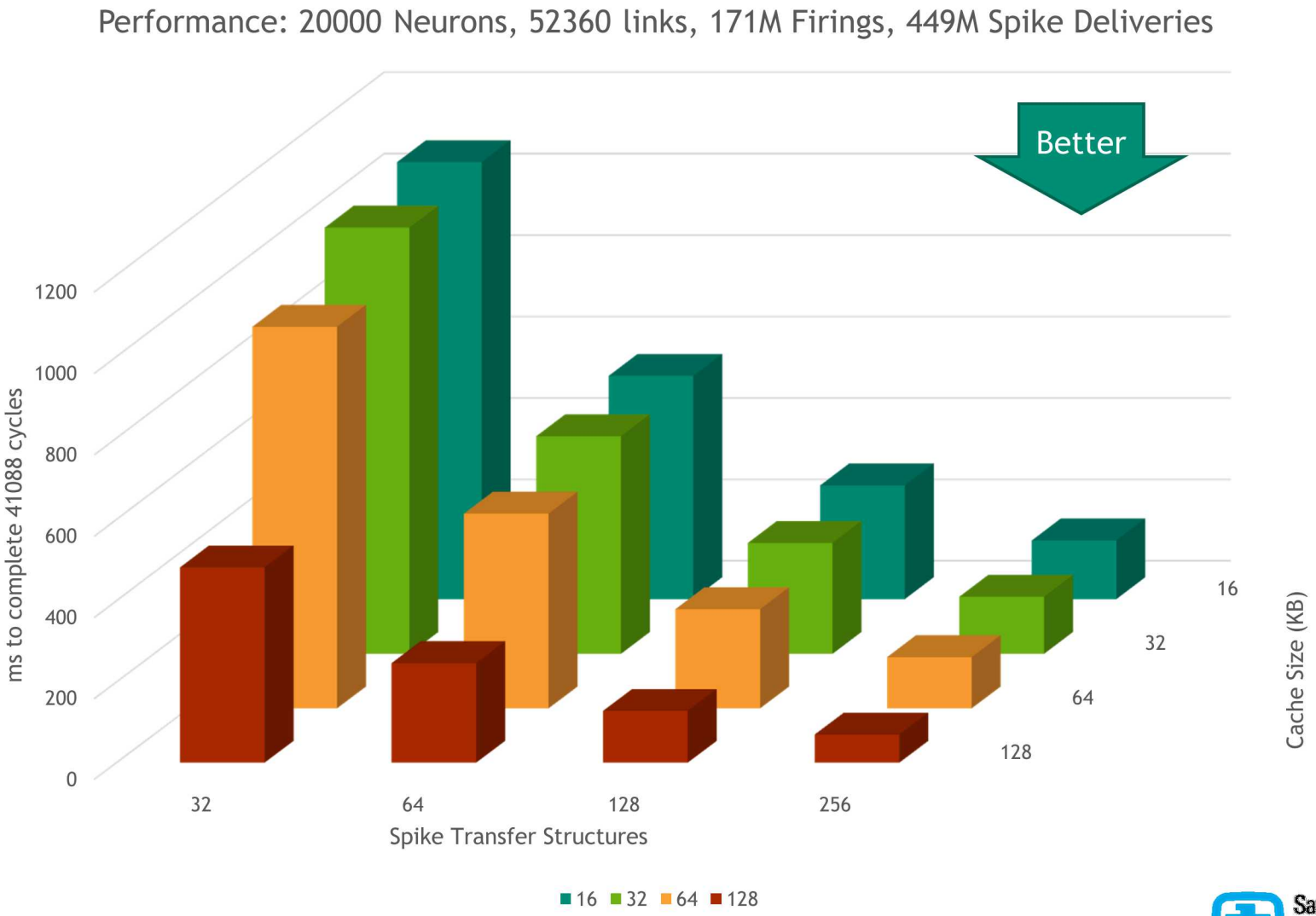


Sandia National Laboratories

Another (Larger) Study

- Parallelism & BW are key
- Caching shows diminishing returns (8x cache improves performance 2x)
- Increasing Parallelism & BW 8x leads to 7-7.5x improvement
- Good candidate for acceleration **BY** memory





Sandia National Laboratories

…For the Memory

The Case of the Multi-Level Memory

- For
  - Lots of potential
  - Lower cost
  - More "Effective" bandwidth

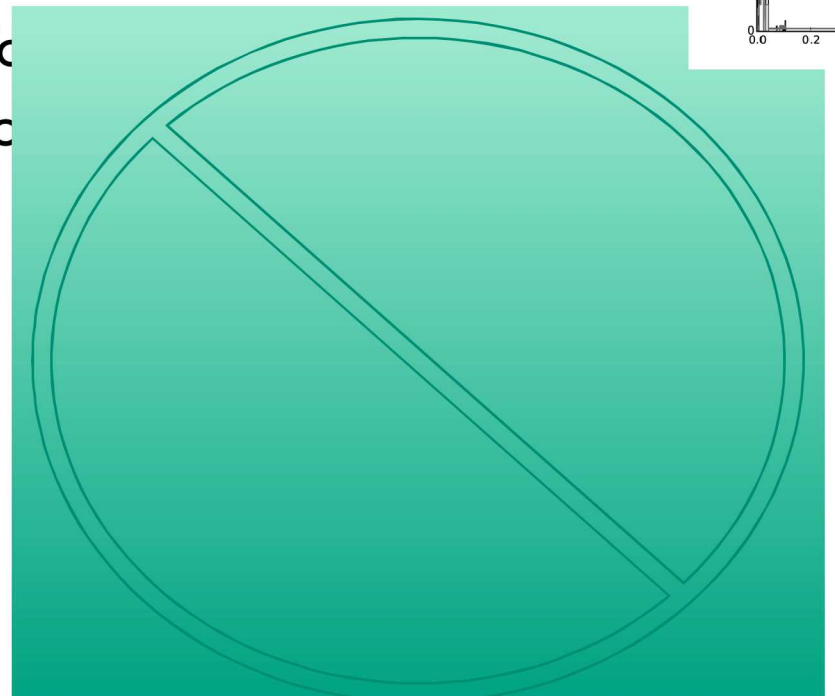
Memory	Size	Cost/bit	Total Cost
"HMC"	5%	3.0	0.15
DDR	30%	1.0	0.3
Flash	65%	0.15	0.1
Total Cost			0.54X

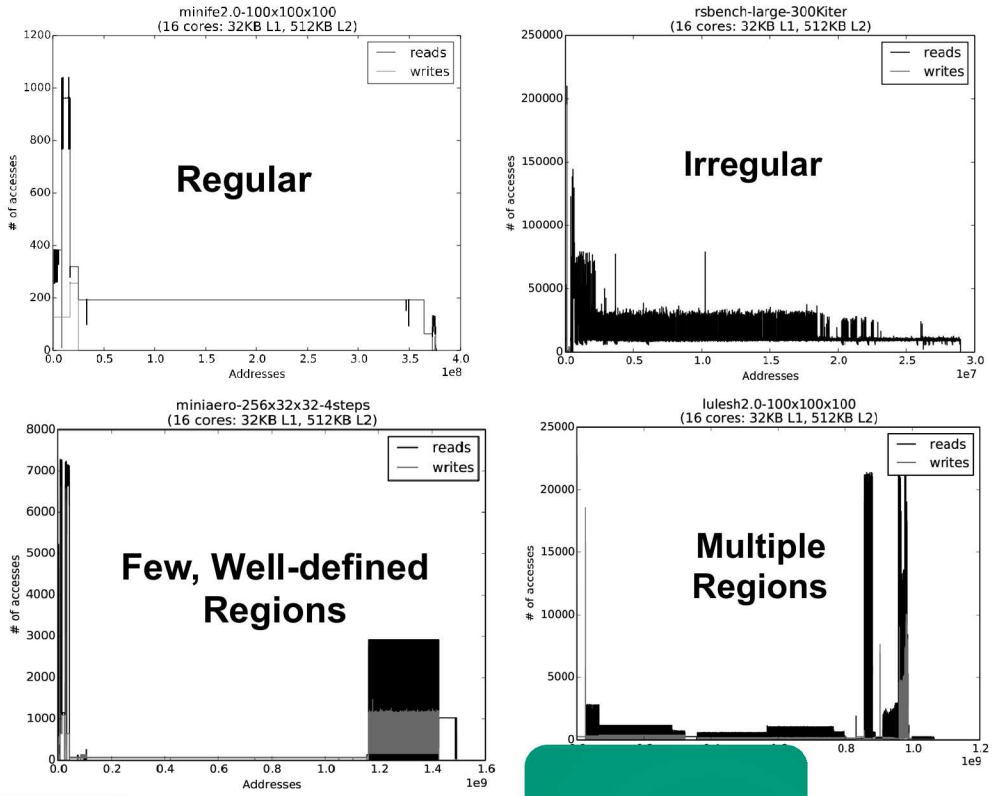
- Against
  - Potential – To waste $$$
  - How do you actually use it?
  - BW not enough – latency effects
  - Can one memory configuration "fit" all?

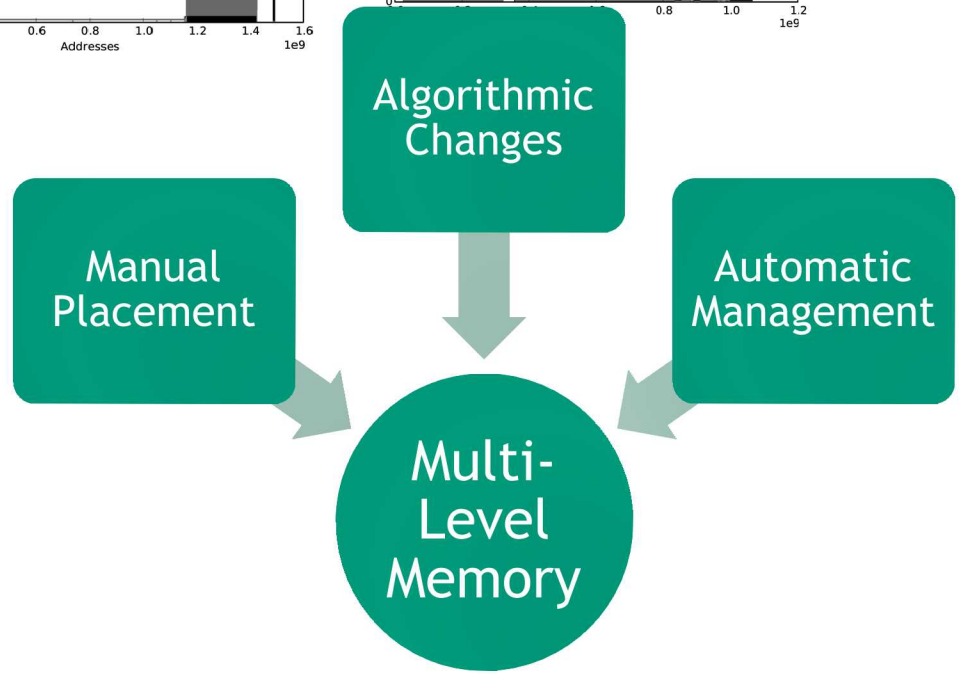
Memory	Accesses	Bandwidth
"HMC"	65%	14.1
DDR	30%	1.0
Flash	5%	0.18
Weighted Harmonic Mean		1.6X

3 Paths

- Algorithmic
- Manual (Malloc)
- Automatic (Paged)
  - Analysis
  - Policies
  - Perfo
  - Recc



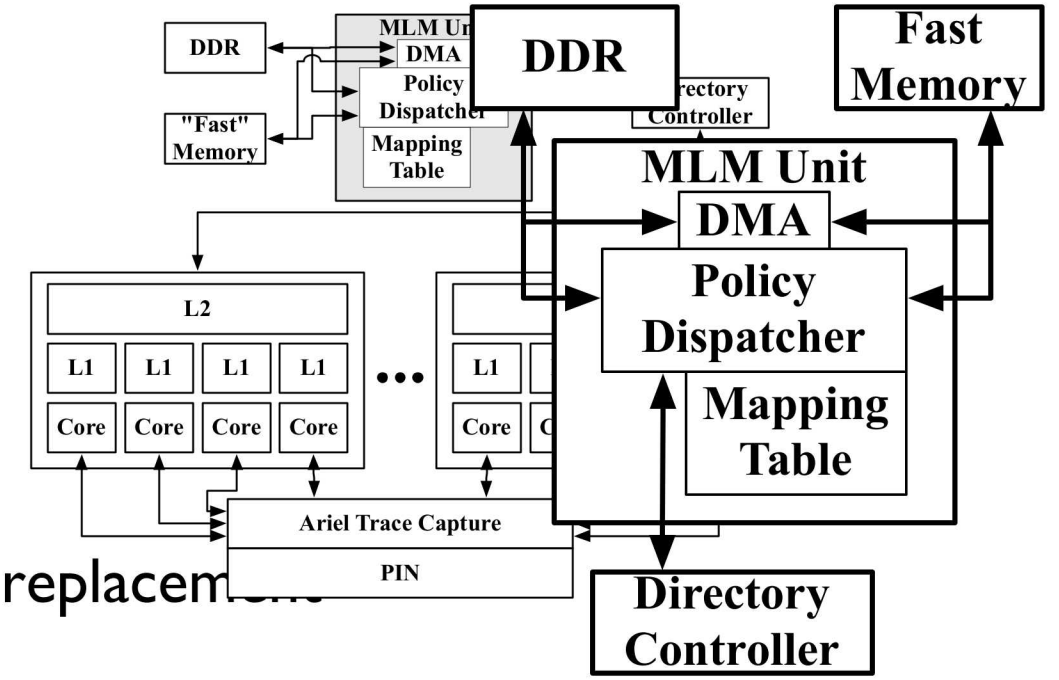




Sandia National Laboratories

Automatic Page-Level Swapping

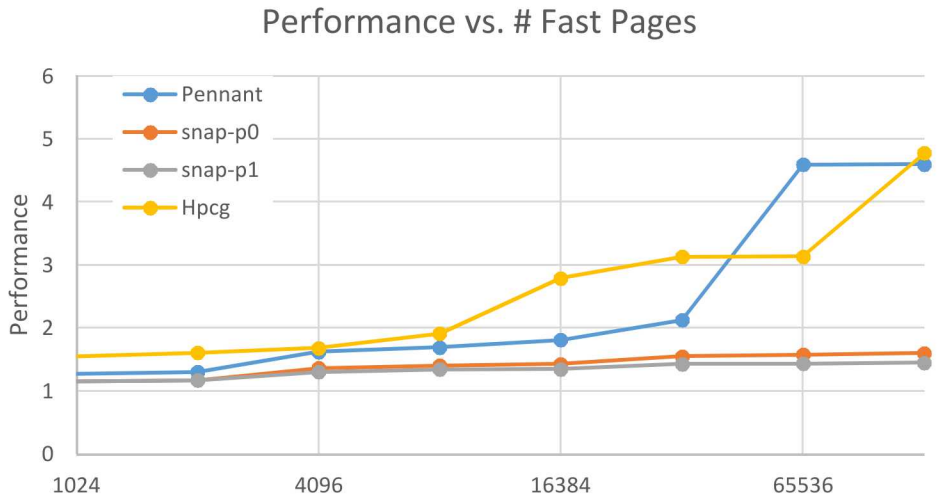
- Architecture
- Difference from "normal" caching
- Addition Policies
- Replacement
- Finding: Addition more important than replacement

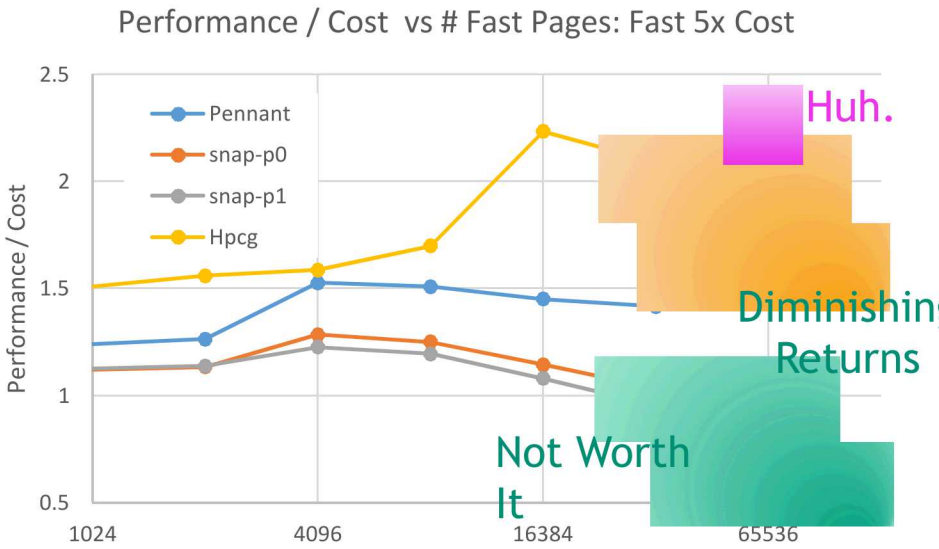


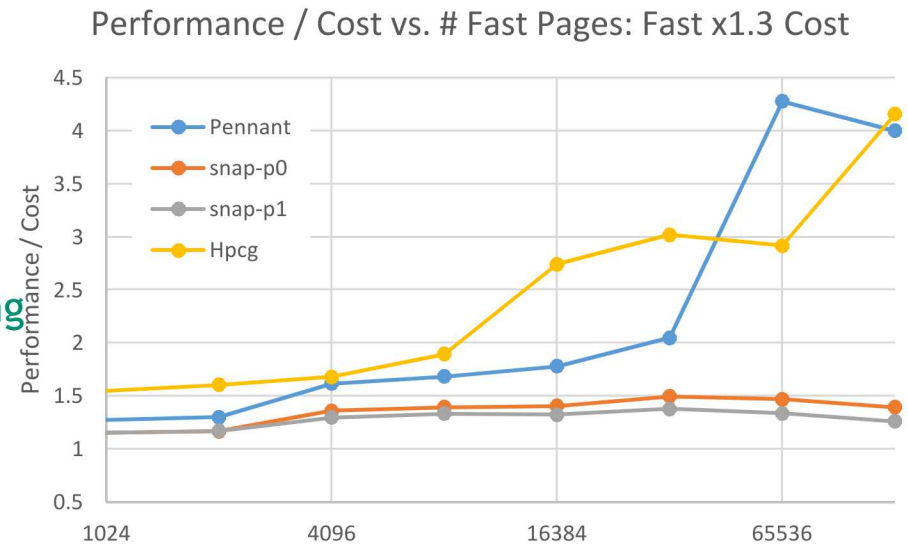
Addition Policies	Replacement Policies
• addT: Simple Threshold • addMFU: Most Frequenly Used • addRAND: 1:8192 chance • addMRPU: More Recent Previous Use • addMFRPU: More Frequent + More Recent Previous Use • addSC: Deprioritize streams • addSCF: as addSC + More Frequent	• FIFO: First-in, First-out • LRU: Least Recently Used • LFU: Least Frequently Used • LFU8: LFU w/ 8-bit counter • BiLRU: BiModal LRU • SCLRU: Deprioritize streams

Cost & Performance

- Ultimate FoM
- 40-380% performance improvement
- Will Cost Kill it?
- Recommendations for HW



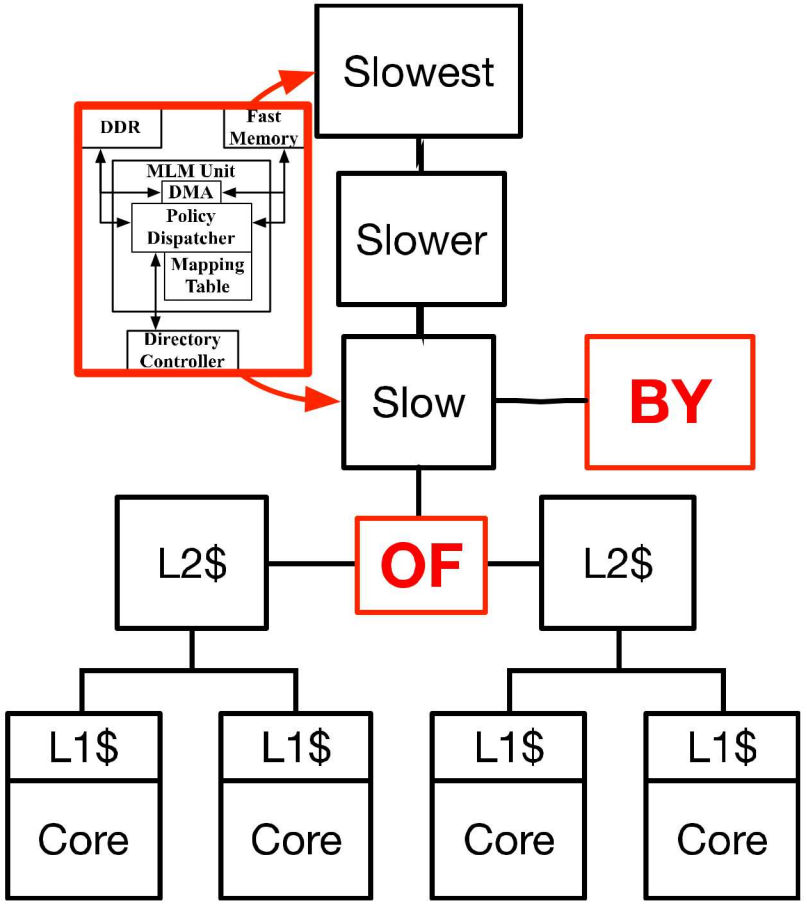




Sandia National Laboratories

Acceleration For the Memory

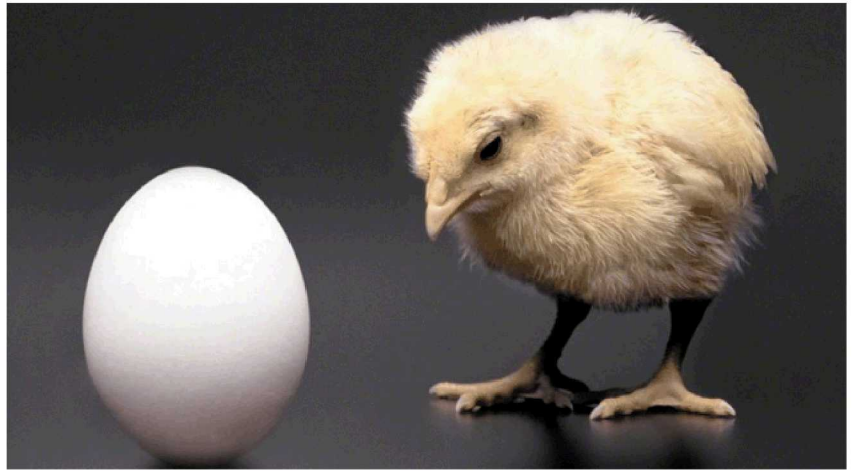
- Multi-level memory can improve performance, lower cost
- Needs management mechanism
- Hardware-assisted memory management shows promise



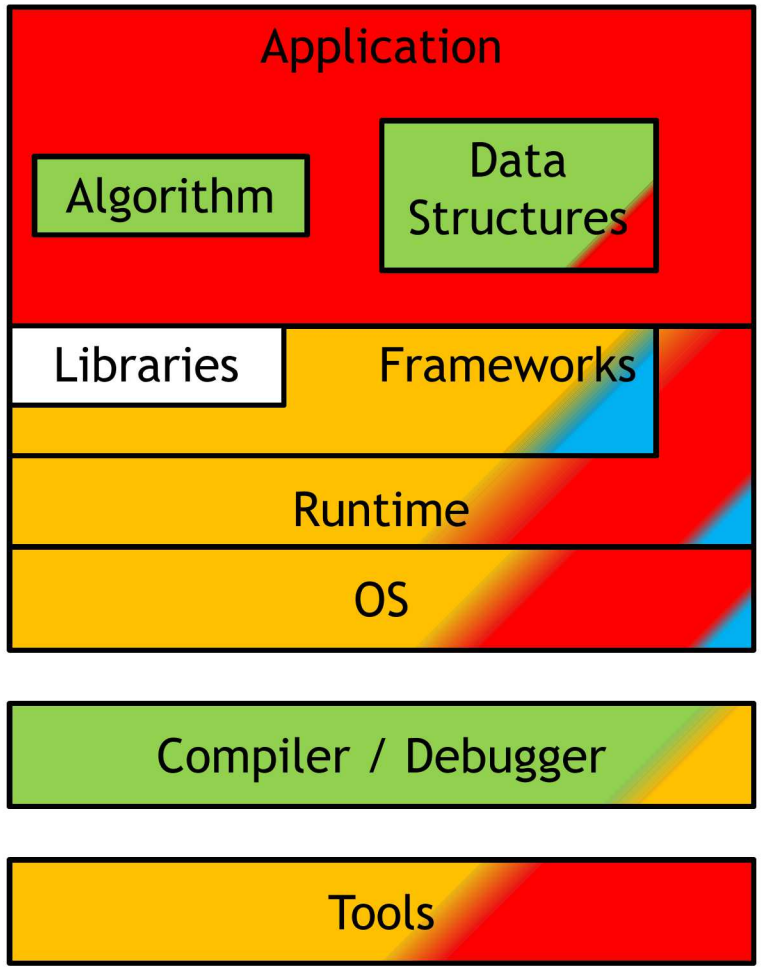
Barriers

Applications

- Expand number of Applications
- Look at 1+ full applications?
  - Are proxies sufficient?
- What we NEED from Apps team
  - Help exploring Algorithmic changes
  - Better input decks



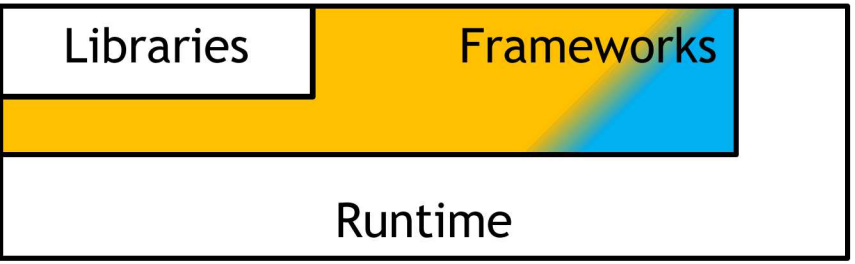
Summary



- Application will bear the brunt of change
- But, there is hope! Work done for GPUs, frameworks (Raja, Kokkos) will help
- Frameworks will require work, but may already have abstractions that will help, may be chances for improvement
- Runtime/OS
  - Straightforward for scratchpad
  - S/G require work on V2P mapping, notification
  - Possible optimizations with FEB/notification mechanisms, S/G communication
- Tools will be needed to support, but good start on the problem
- Compilers will continue to be compilers

- **Red**: Substantial
- **Yellow**: "usual" effort
- **Green**: Minimal effort.
- **Blue**: Make things easier?

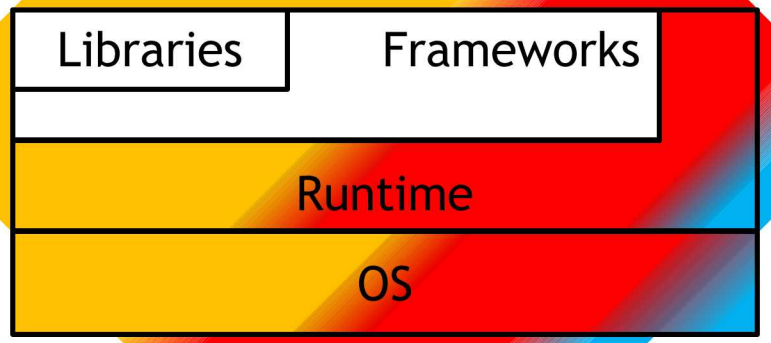
Frameworks (to the Rescue?)

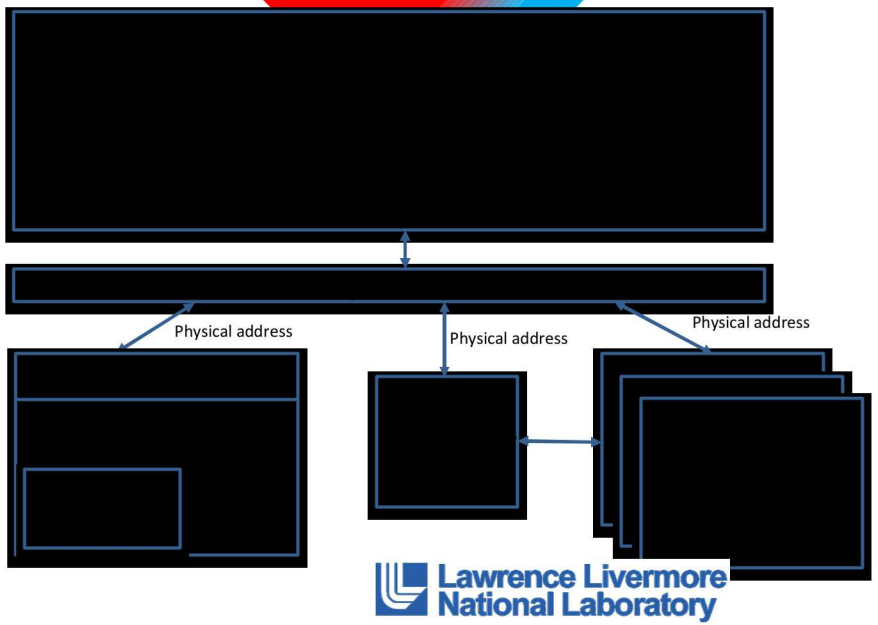


- Frameworks (Kokkos, Raja) may make changes easier
- Encourage / force user to use abstracted data types, make synchronization explicit (e.g. Kokkos barriers), define task dependencies
  - Already have support for scratchpads
  - Use of lambdas to define tasks
  - Support for futures (Trying to push on C++2*x* standard)
  - Arbitrary load mechanisms (e.g. Kokkos views)
- **Many key abstractions already there**
- **Easier** implementation: FEBs for futures

```
Kokkos::parallel_for( Kokkos::ThreadVectorRange( teamMember, M ), [&] ( const int i ) {
  Prefetch(A( i ))
  Prefetch(B( e, i, j ));
});
Kokkos::parallel_for( Kokkos::ThreadVectorRange( teamMember, M ), [&] ( const int i ) {
  A( i ) = c + B( e, i, j );
});
```

R…mpact





- Scatter/Gather
  - **Protection**: mechanisms exist (similar to DMA)
  - Need **notification** mechanism
  - Need Virtual to Physical **Mapping**
    - IOMMU too heavy weight
    - Have written S/G Device Driver (DRE)
    - **Some techniques exist**, needs more **exploration**
- **FEBs**
  - Useful to notify? Similar to monitor mwait instruction
  - Fast signaling between cores?
  - Useful for load balancing / Eureka notification
  - Lots of Producer/consumer relationships in runtime/OS

Sandia National Laboratories