Performance Analysis of Runtime Handling of Zero-Copy for OpenMP® Programs on MI300A* APUs

Carlo Bertolli
AMD ROCm Team

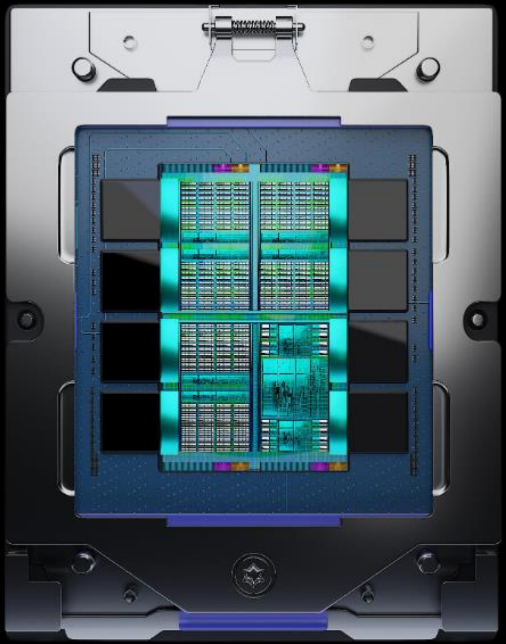
AMD
together we advance_

* AMD Instinct™ MI300A series accelerators

Motivation for Accelerated Processing Units (APUs)

By integrating ‘Zen 4’ CPU cores and GPU accelerators, you can achieve high efficiency by eliminating time consuming data copy operations, transparently managing CPU and GPU caches, offloading tasks easily between GPU and CPU, and efficient synchronization [..]

https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/data-sheets/amd-instinct-mi300a-data-sheet.pdf



https://www.hpcwire.com/2023/01/05/amd-shows-off-mi300-chip-for-the-first-time/

https://asc.llnl.gov/exascale/el-capitan

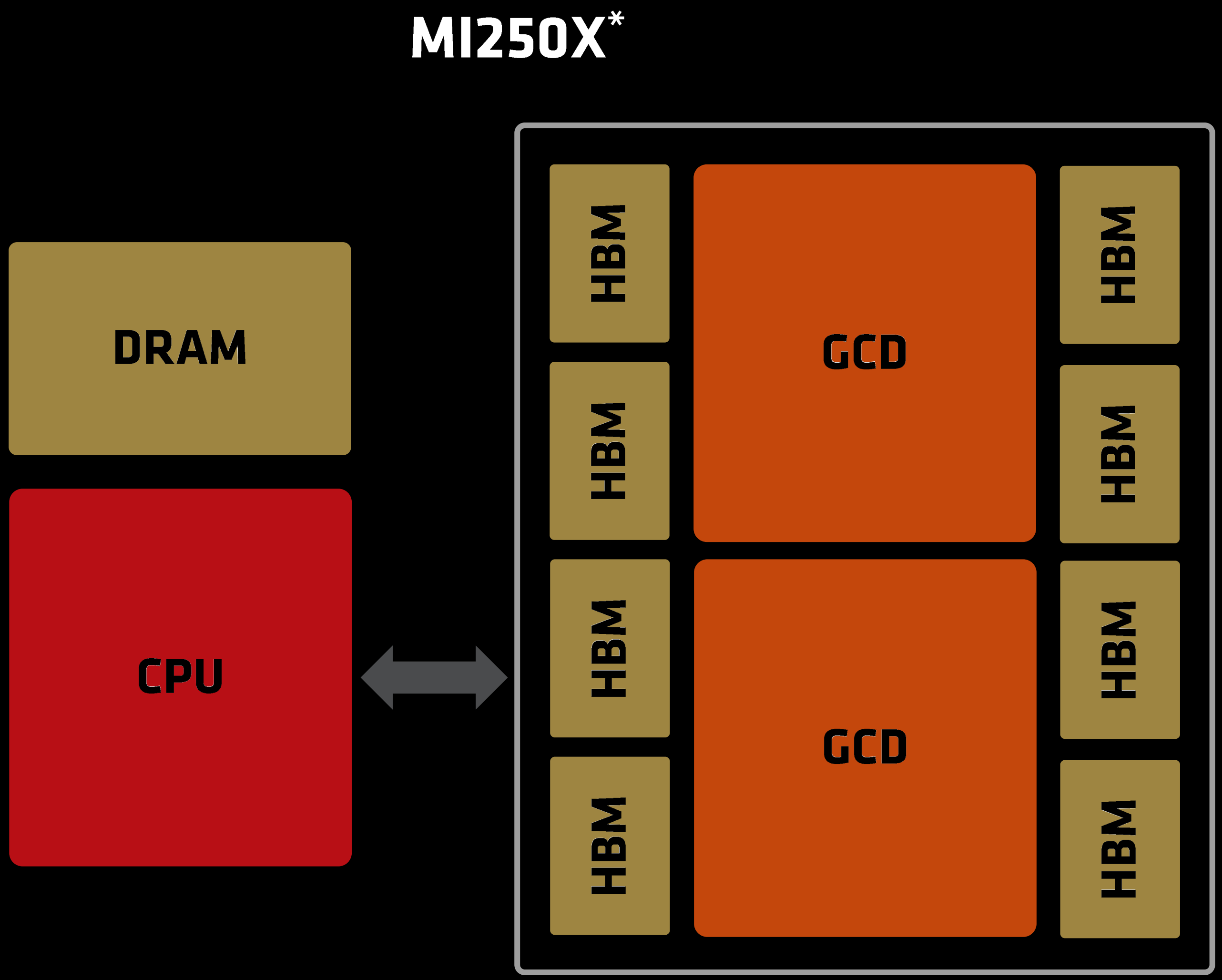
HPE Cray Supercomputing EX255a

The features of this accelerator blade are as follows:

- Two 4-socket AMD Instinct™ MI300a Accelerator APU nodes
- 128GB HBM3 per APU
- Up to 8 HPE Slingshot 200Gbit/sec ports per blade
- 0 or 1 local NVMe M.2 SSD per node (up to 2 per blade)
- 2 Board Management Controllers (BMC) per blade
- Cooled with cold plate

https://www.hpe.com/psnow/doc/a00094635enw

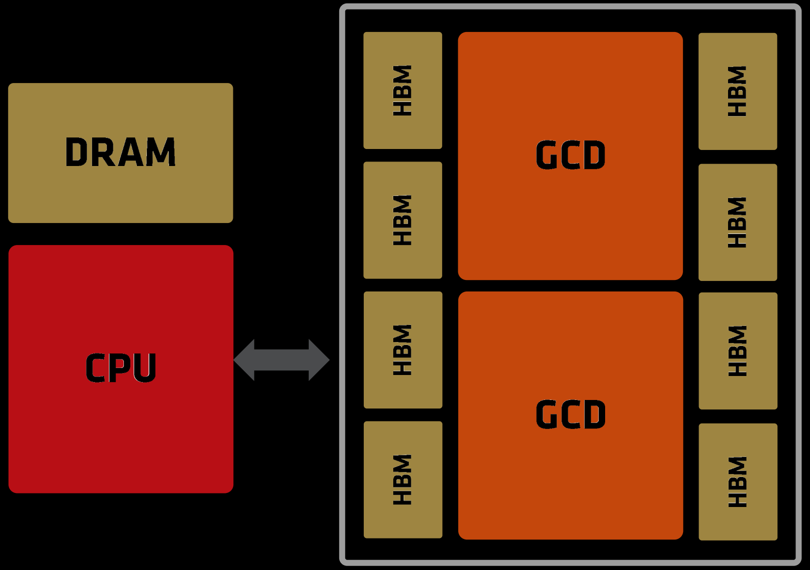
Discrete GPU...



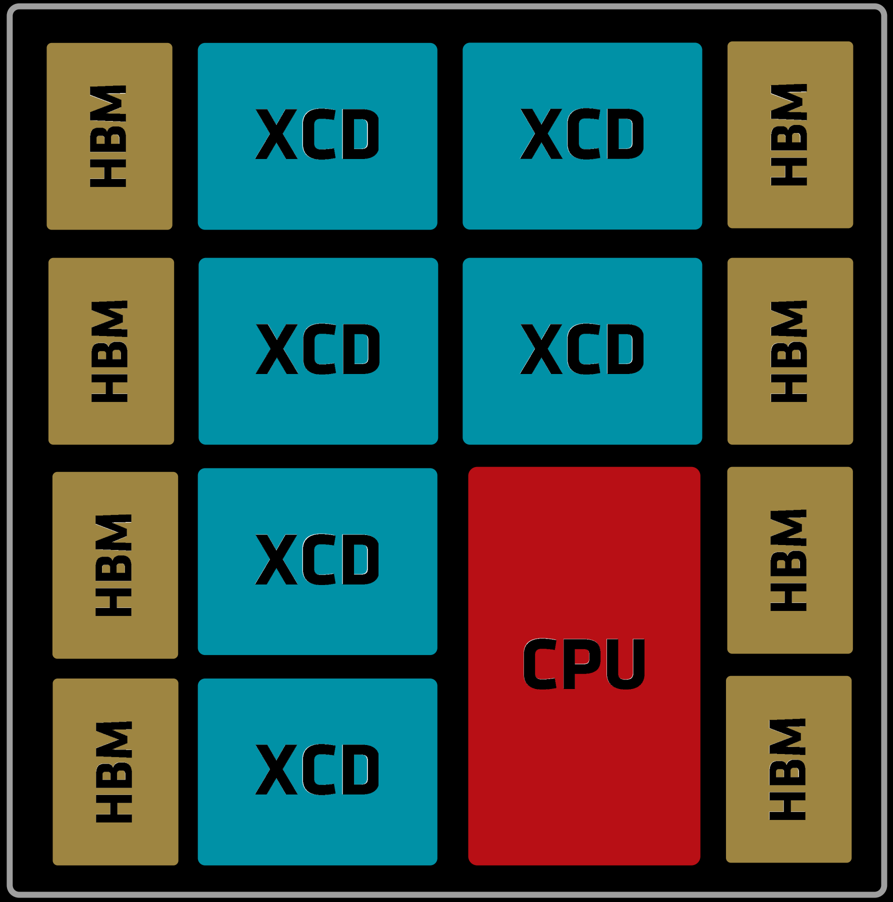
* AMD Instinct™ MI200 series accelerators

Discrete GPU… and APU Architecture

MI250X



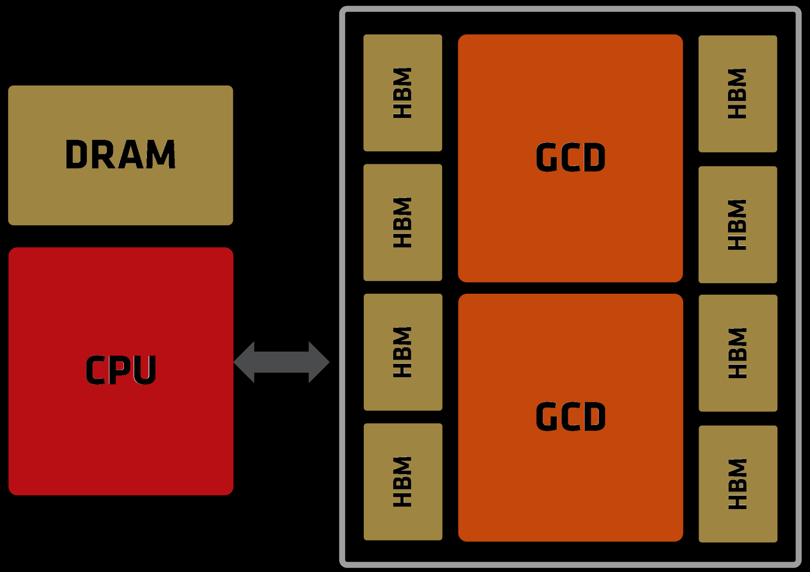
MI300A "APU"



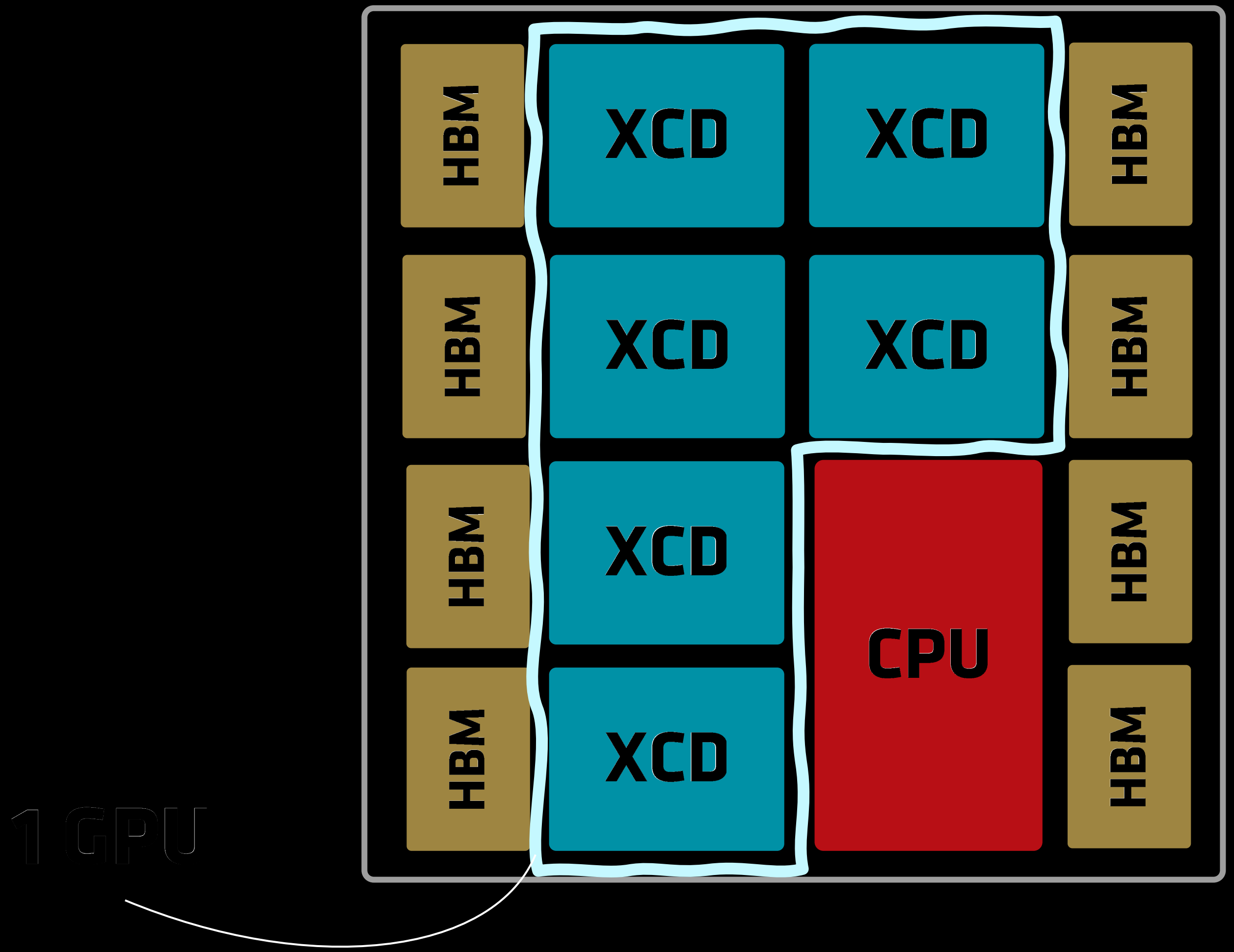
AMD
together we advance_

Discrete GPU... and APU Architecture

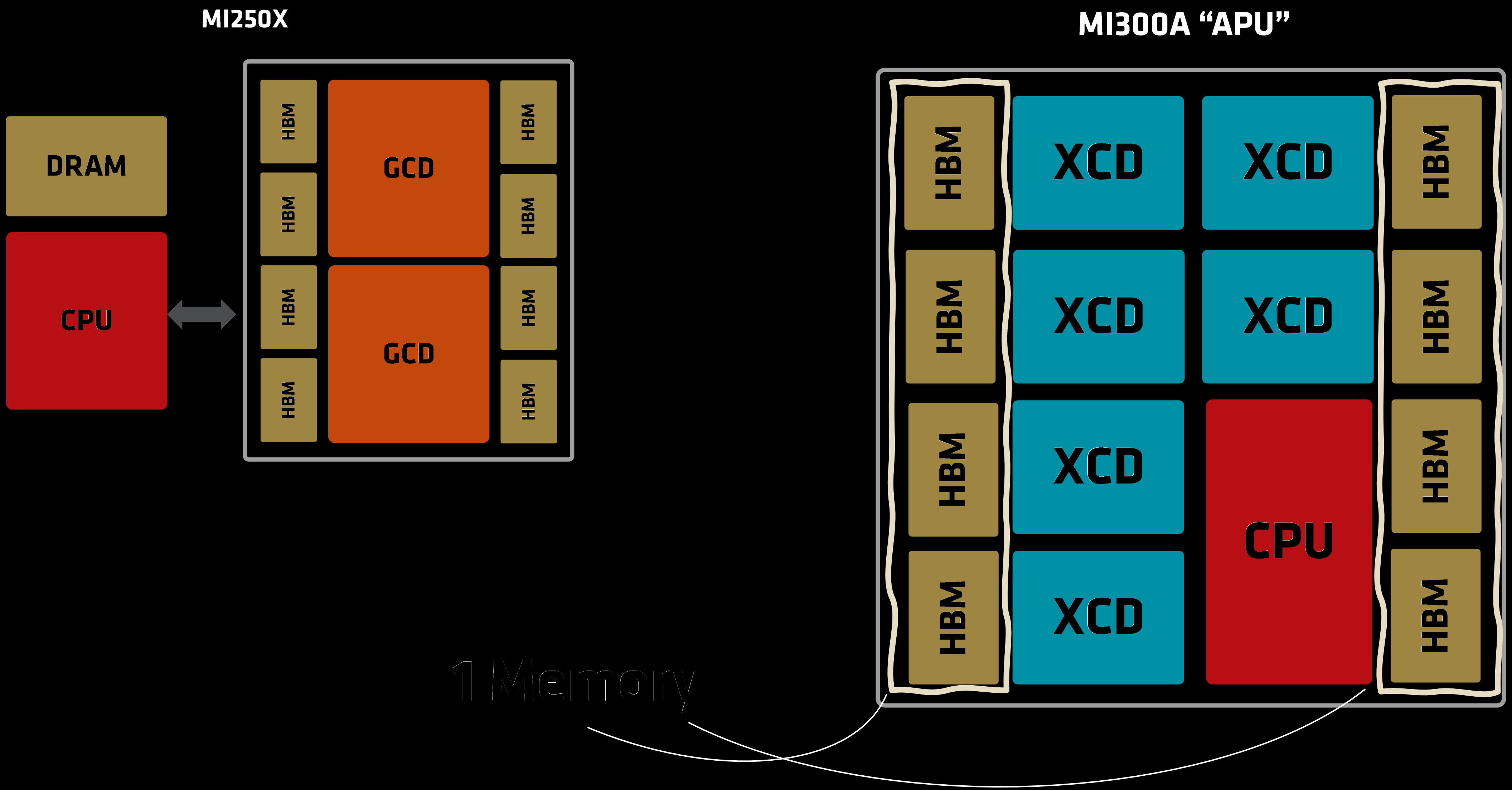
MI250X



MI300A “APU”



Discrete GPU… and APU Architecture



AMD
together we advance_

Programming an APU in 2024

- HIP Applications
  - Abstraction layers hiding memory management
  - Re-implementation for APU should be relatively straightforward

- DSL and high level languages
  - Raja, Kokkos, DeVito, SYCL++®
    - Flip a switch

- OpenMP memory mapping

```
double *ptr = malloc(1024*sizeof(double));
#pragma omp target map(ptr[:1024])
  ptr[0] = 1.0;
```

- `map(ptr[0:1024])`
  - Memory ptr[0] to ptr[1023] is added to device data environment
  - Implementations
    - dGPU: device memory allocation, D2H/H2D copies (**copy**)
    - APU: just pass the pointer (**zero-copy**)

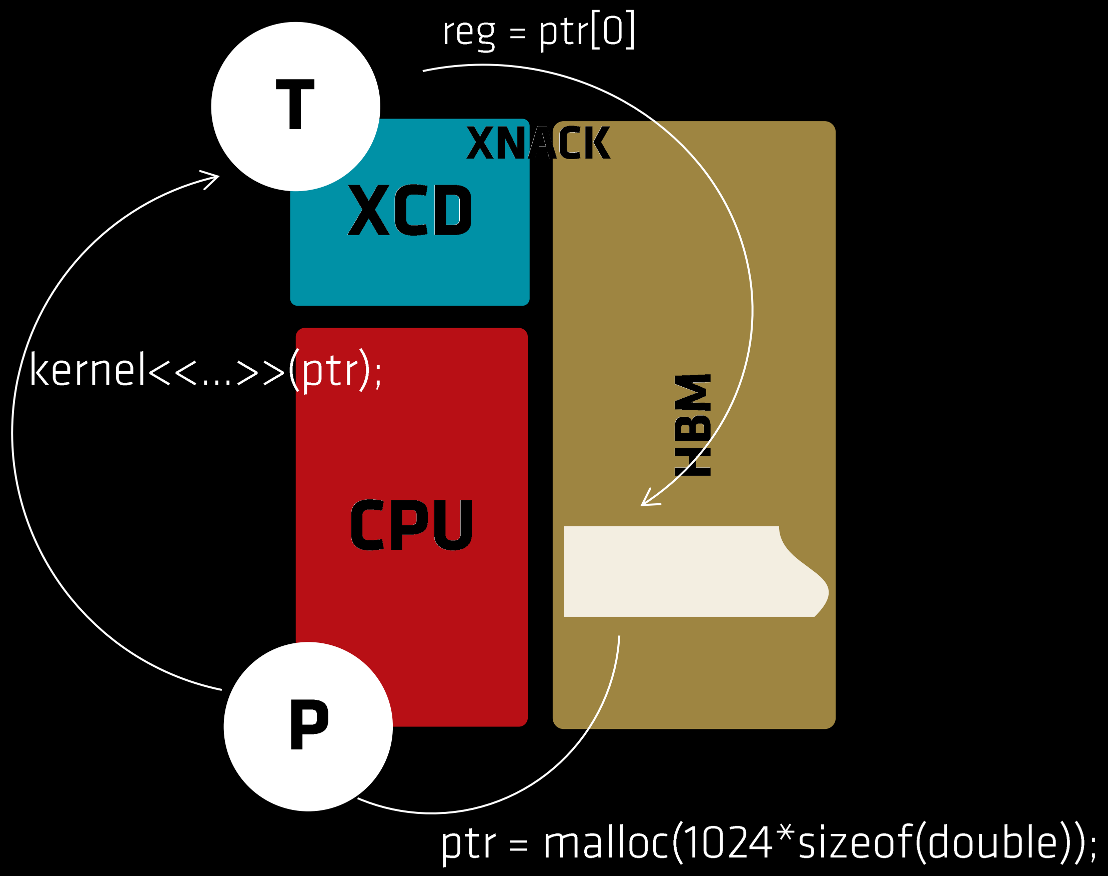
Programming an MI300A 'APU' with OpenMP

Compiler Flags -fopenmp -offload-arch=gfx942		Programming Mode: Default non-unified_shared_memory using map clauses	Programming Mode: unified_shared_memory #pragma omp requires unified_shared_memory or --fopenmp-force-usm
Runtime State	Unified Memory Enabled HSA_XNACK=1	Zero-copy	Zero-copy
Runtime State	Unified Memory Disabled HSA_XNACK=0	Copy	Runtime Error

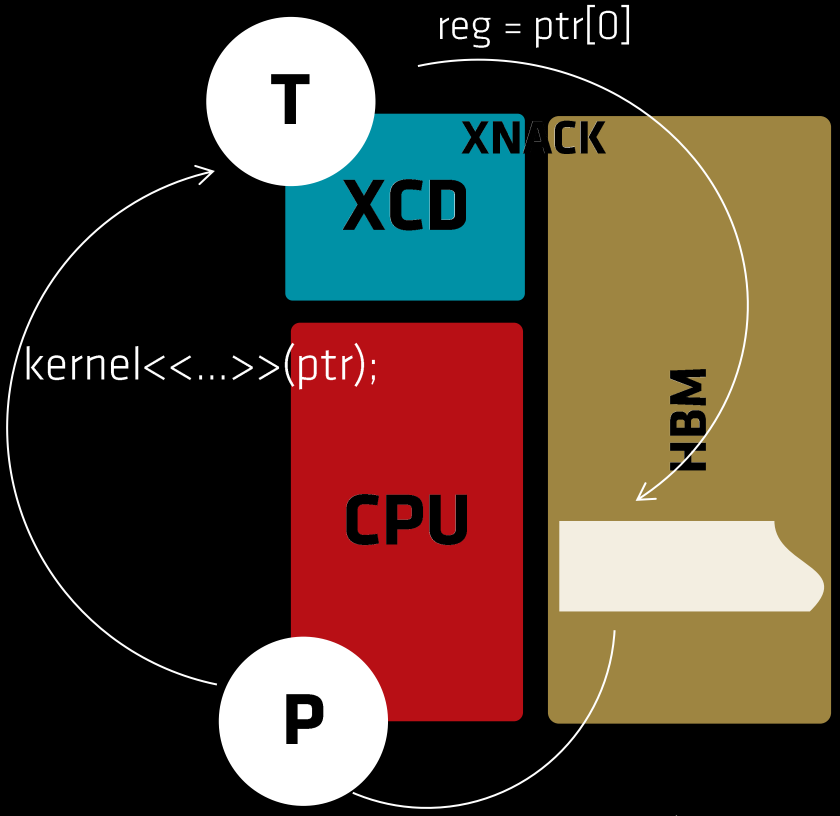
Programming an MI300A 'APU' with OpenMP

Compiler Flags -fopenmp -offload-arch=gfx942		**Programming Mode**		
		Default non-unified_shared_memory using map clauses		**unified_shared_memory** #pragma omp requires unified_shared_memory or --fopenmp-force-usm
Runtime State	**Unified Memory Enabled** HSA_XNACK=1	**Zero-copy**		**Zero-copy**
	Unified Memory Disabled HSA_XNACK=0	OMPX_EAGER_ZERO_COPY_MAPS=0 **Copy**	OMPX_EAGER_ZERO_COPY_MAPS=1 **Zero-Copy**	**Runtime Error**

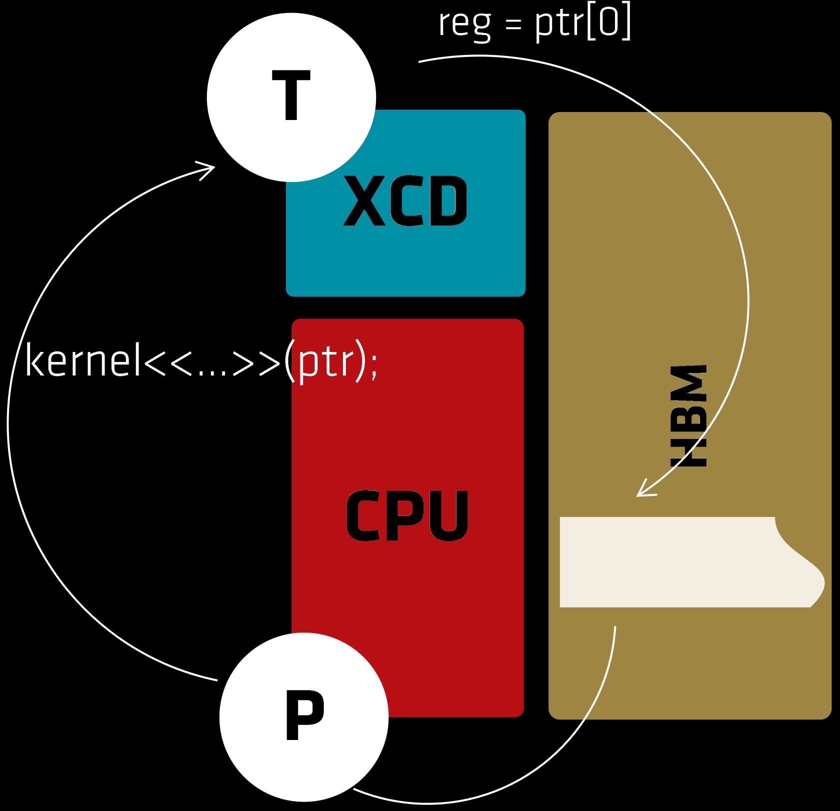
How to access CPU-allocated Memory on the GPU? XNACK



How to access CPU-allocated Memory on the GPU? XNACK or Prefault



ptr = malloc(1024*sizeof(double));



ptr = malloc(1024*sizeof(double));
gpu_page_table_prefault(ptr, 1024*sizeof(double));

Programming an MI300A 'APU' with OpenMP

<table>
<tr><td colspan="2" rowspan="2">Compiler Flags
-fopenmp -offload-arch=gfx942</td><td colspan="3">Programming Mode</td></tr>
<tr><td colspan="2">Default
non-unified_shared_memory
using map clauses</td><td>unified_shared_memory
#pragma omp requires unified_shared_memory
or
--fopenmp-force-usm</td></tr>
<tr><td rowspan="3">Runtime State</td><td>Unified Memory Enabled
HSA_XNACK=1</td><td colspan="2">Implicit (or Auto) Zero-copy</td><td>Unified Shared Memory</td></tr>
<tr><td rowspan="2">Unified Memory Disabled
HSA_XNACK=0</td><td>OMPX_EAGER_ZERO_COPY_MAPS=0</td><td>OMPX_EAGER_ZERO_COPY_MAPS=1</td><td rowspan="2">Runtime Error</td></tr>
<tr><td>Copy</td><td>Eager Maps</td></tr>
</table>

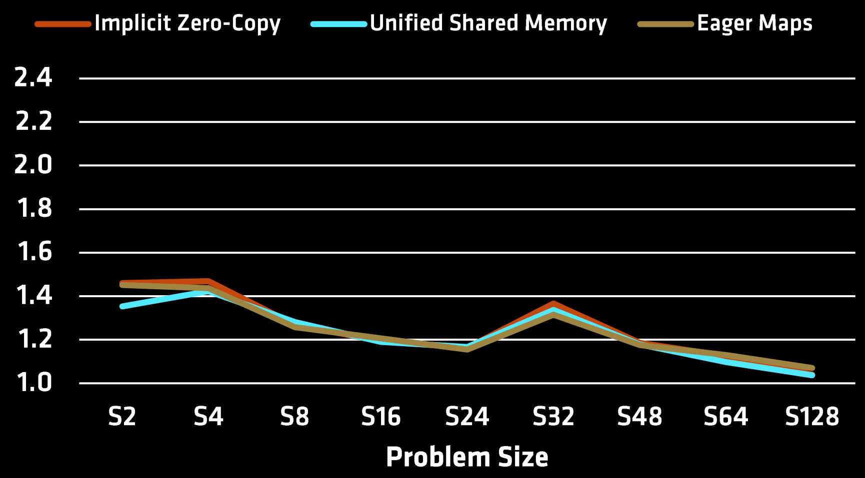
Experiments

- Platform
  - Single socket MI300A node
  - ROCm 6.1.1 or later
  - Transparent Huge Pages enabled for 2MB pages
  - Ubuntu® 22.04
- QMCPack NiO performance tests, S2-S128 data sizes
  - Effects of data prefetching and streaming
- SPECaccel® 2023 C/C++ benchmarks
  - Corner cases
- All Results are ratios: Copy/* (* = Implicit Zero-Copy, USM, Eager Maps)

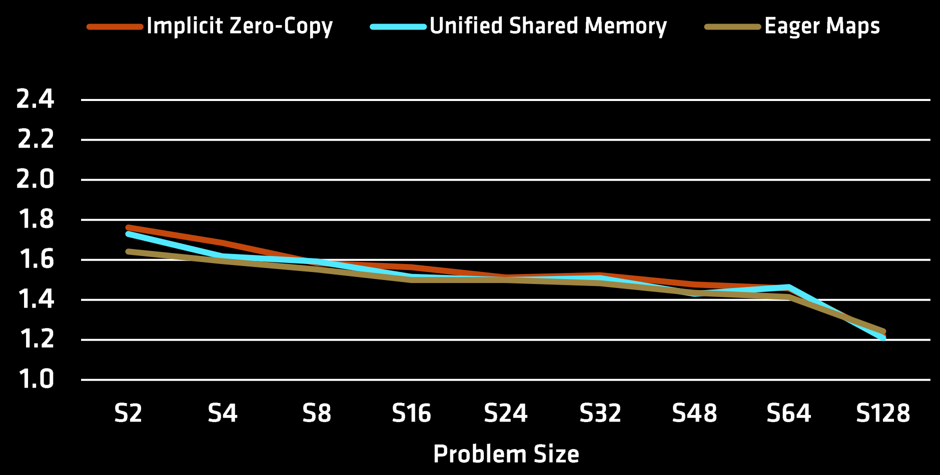
QMCPack Problem Size Scaling

Ratio between Copy/Zero-Copy configuratoins

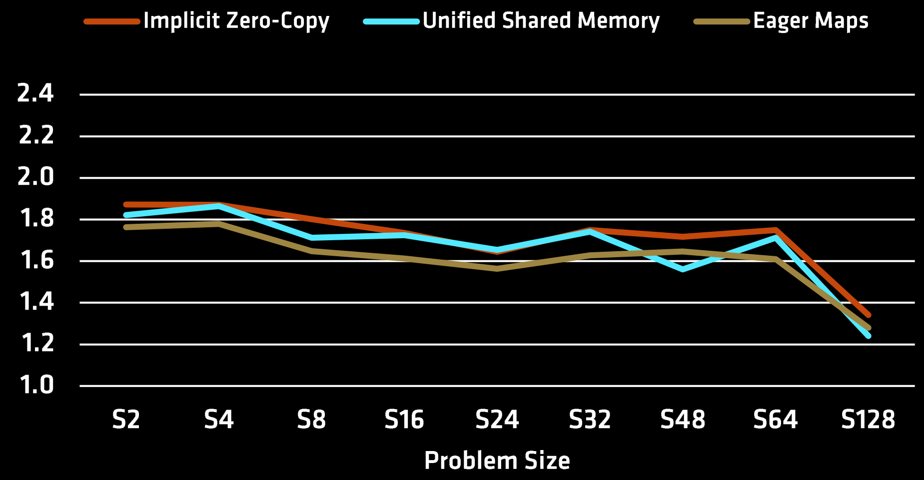
1 OpenMP Thread



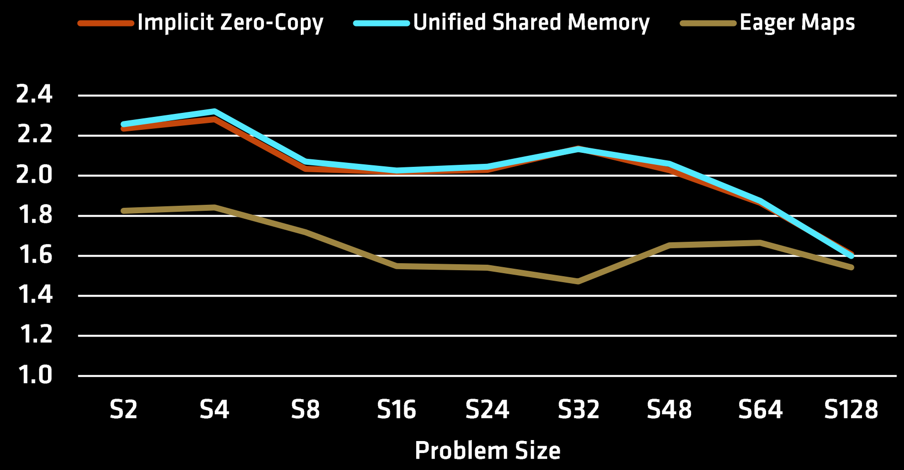
2 OpenMP Threads



4 OpenMP Threads



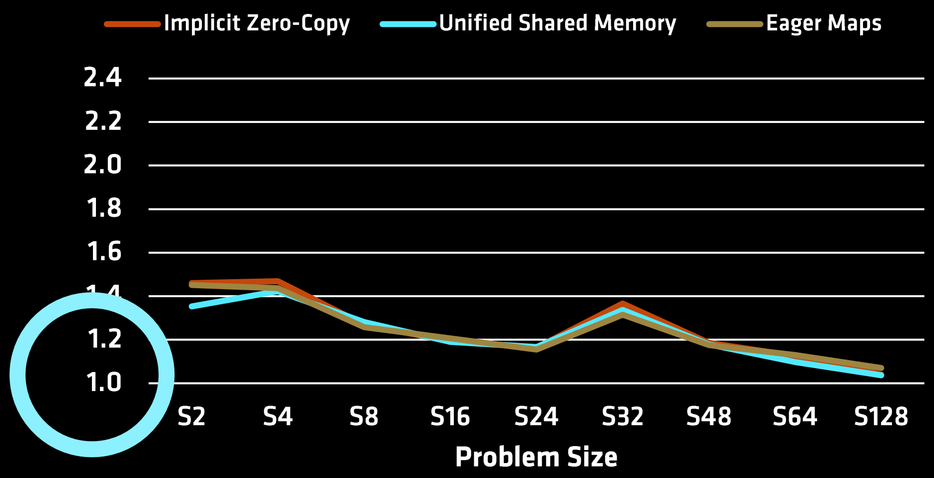
8 OpenMP Threads



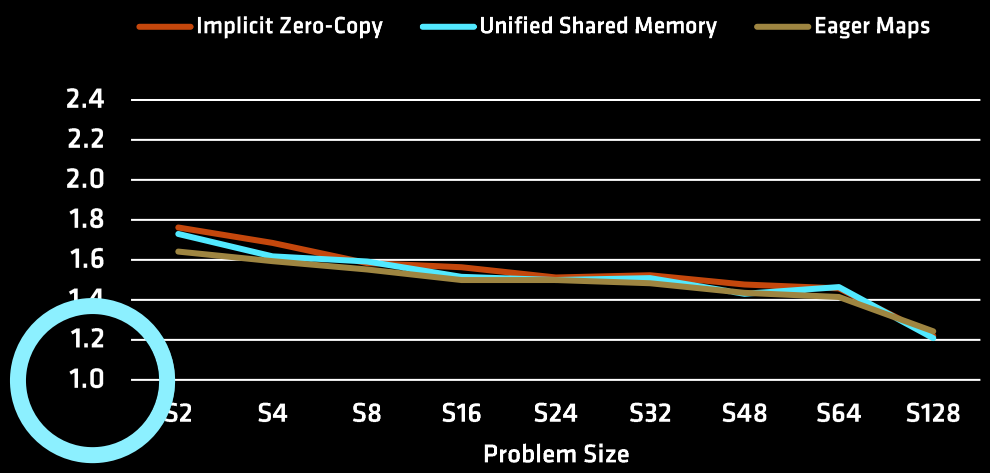
QMCPack Problem Size Scaling

Ratio between Copy/Zero-Copy configuratoins

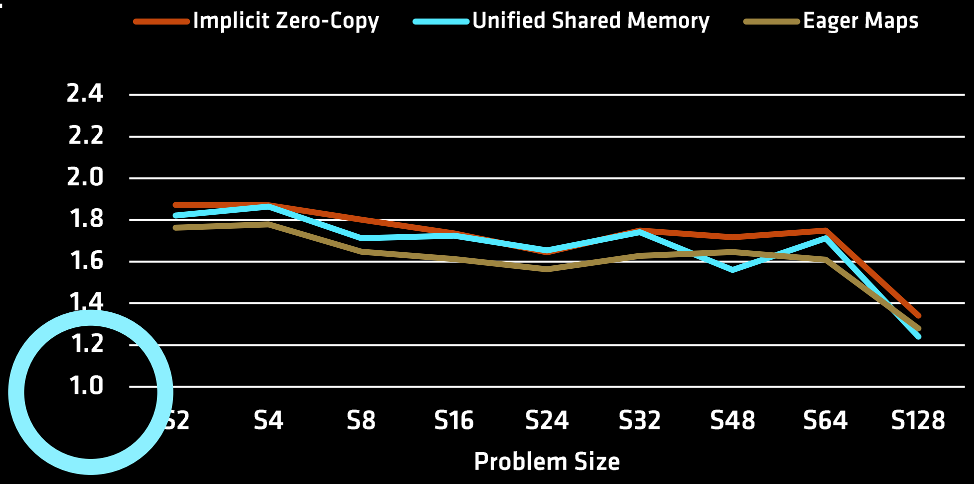
1 OpenMP Thread



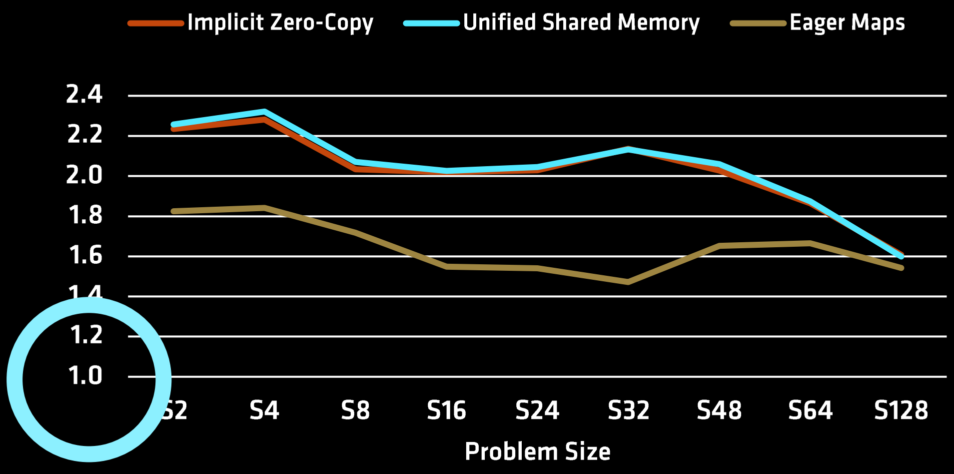
2 OpenMP Threads



4 OpenMP Threads



8 OpenMP Threads



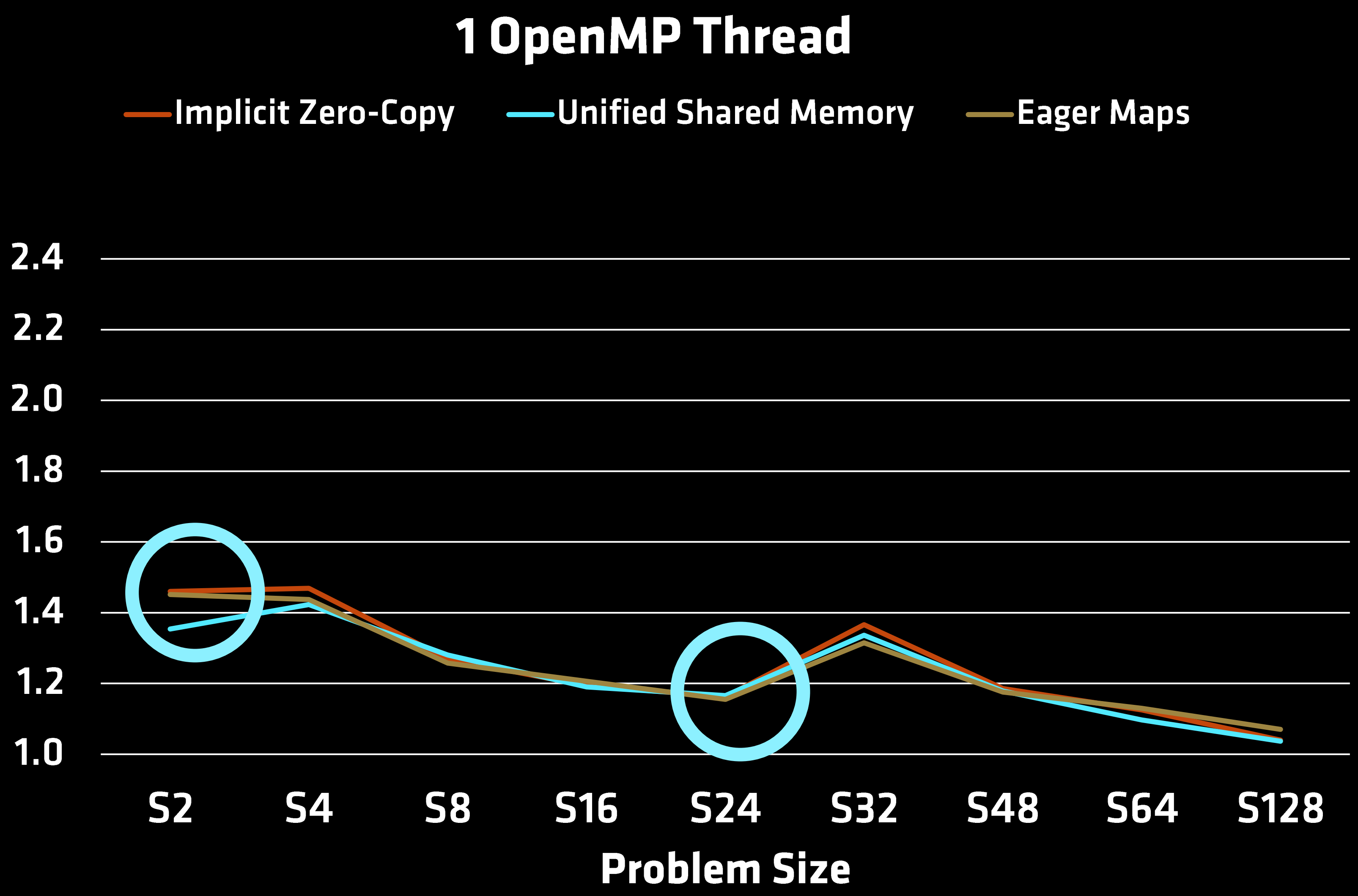
AMD
together we advance_

Why is Zero-Copy Winning?

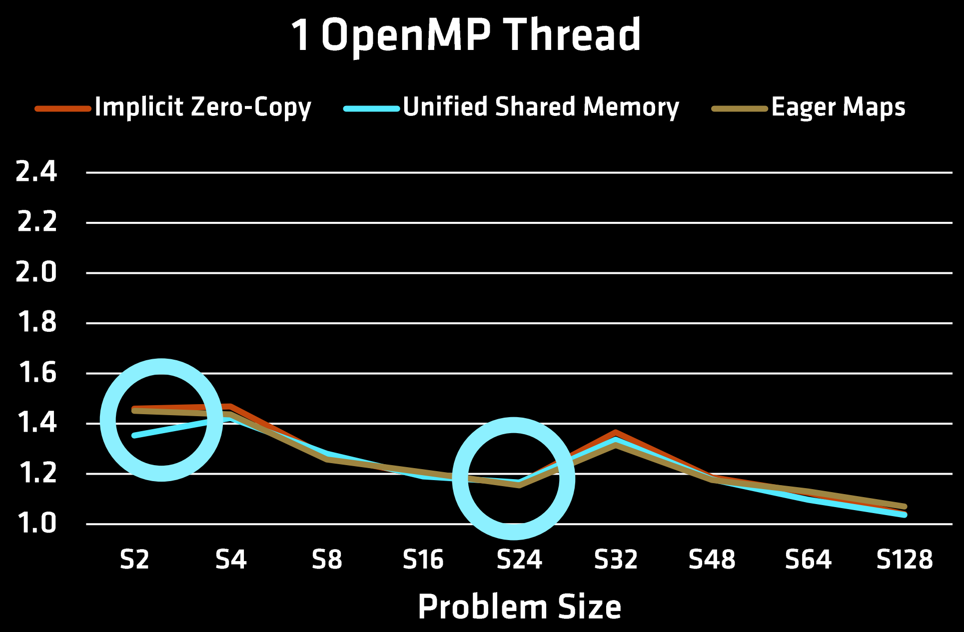
HSA™/ ROCr call	Use	Copy	Implicit Zero-Copy	Copy/Implicit Z-C
1 OpenMP host thread		#calls	#calls	ratio
signal wait scacquire	Kernel completion	351,653	99,627	3.53
memory pool allocate	Allocate device memory	23,277	19	1.23×10^3
memory async copy	Memory copy	307,607	3	1.03×10^5
signal async handler		194,848	0	N/A

AMD
together we advance_

Why Increasing Problem Size Hurts Zero-Copy?



Why Increasing Problem Size Hurts Zero-Copy?



Number of (HSA) runtime calls

- Copy: 5X
- Implicit Zero-Copy: 10X
- Copy call latency >> Implicit Zero-Copy

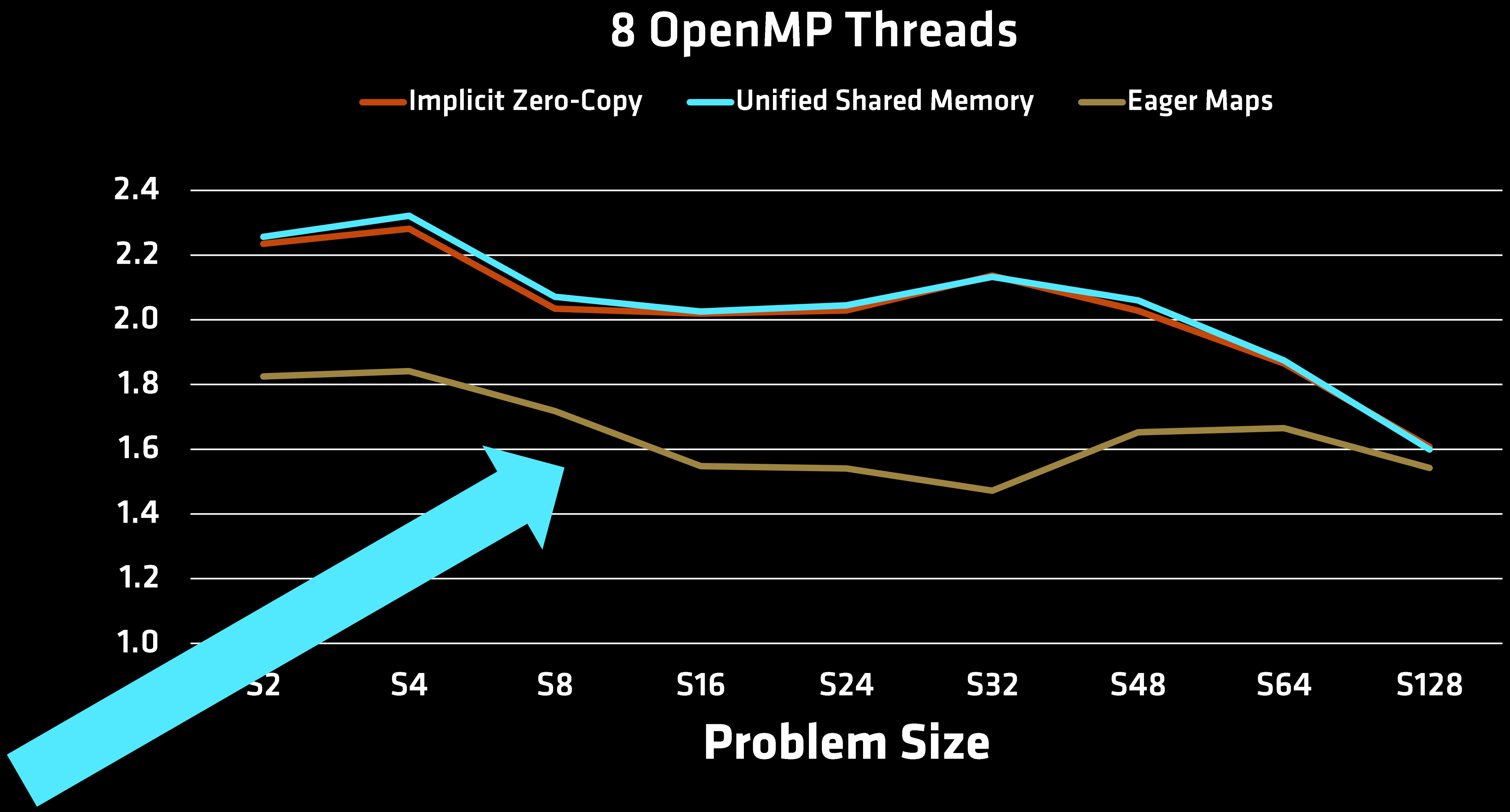
Larger problem size means:

- Larger data structures
- Overhead does not increase
- More time spent in kernels

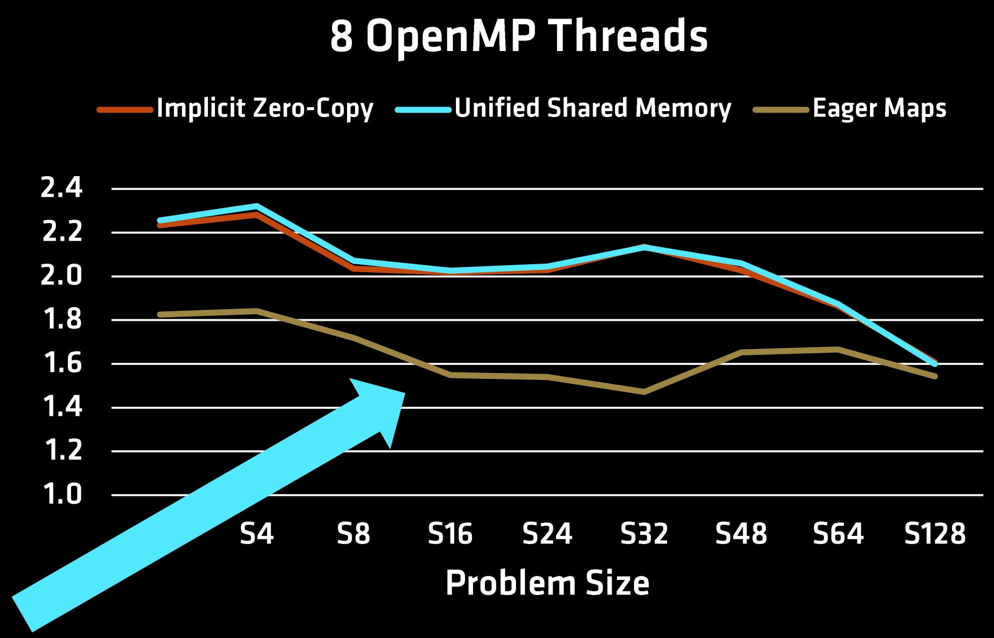
Data prefetching and data streaming

- Amortize extra memory copies

Why Eager Maps Suffers at 8 OpenMP Host Threads?



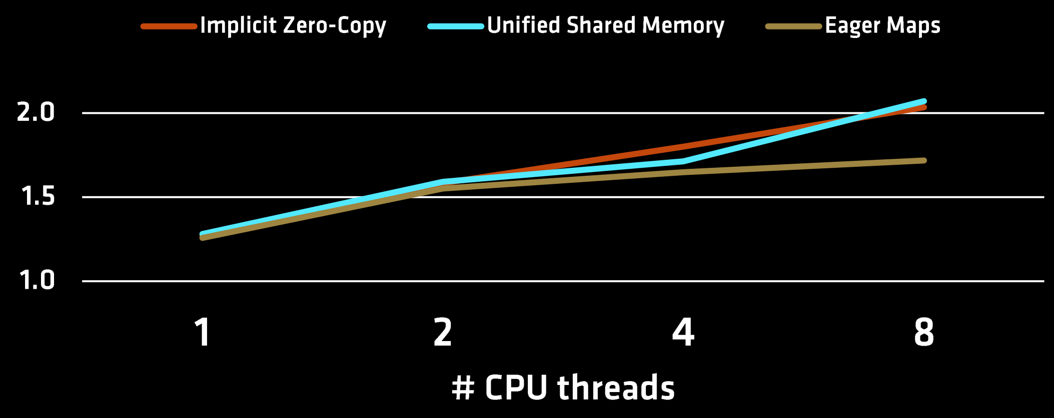
Why Eager Maps Suffers at 8 OpenMP Host Threads?



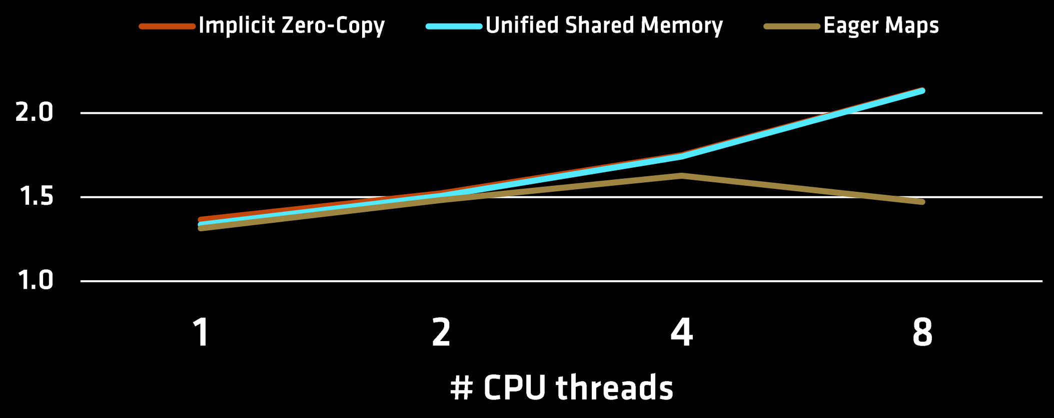
- 8 threads asking the driver to prefault memory
  - Synchronous call
  - Contention on same driver
- Not visible when most of the time is spend in kernel (S128)

QMCPack OpenMP Host Thread Scaling

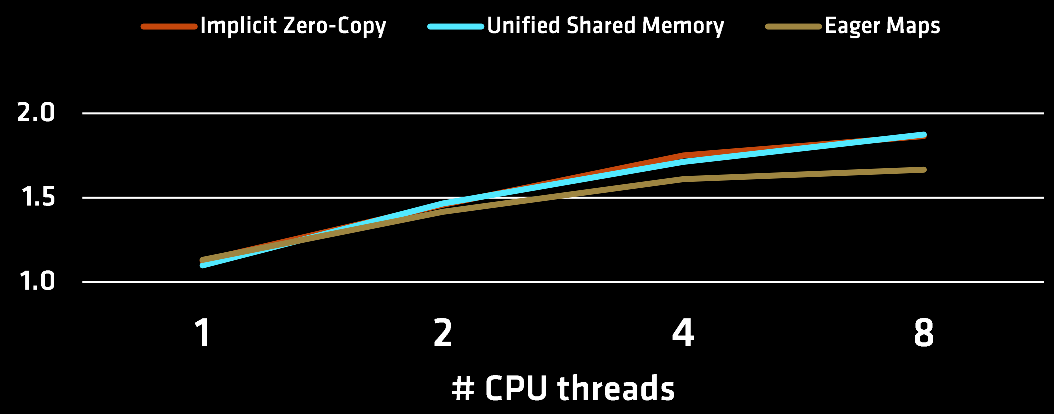
S8 Problem Size



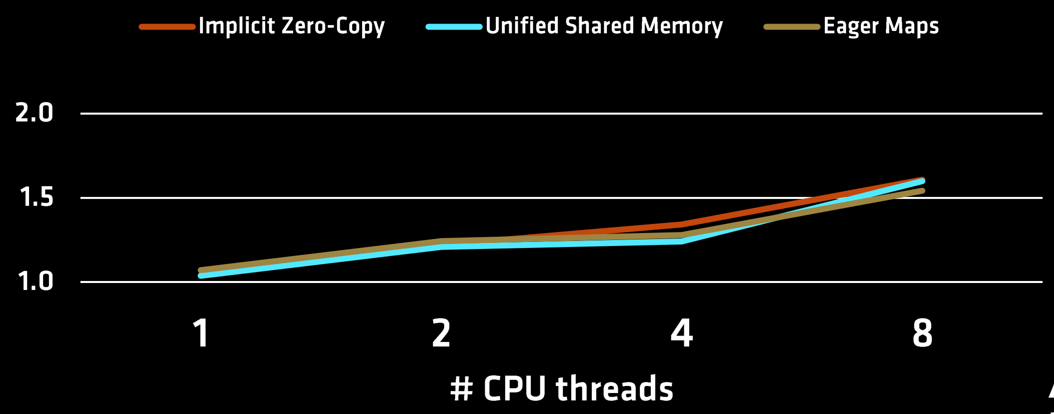
S32 Problem Size



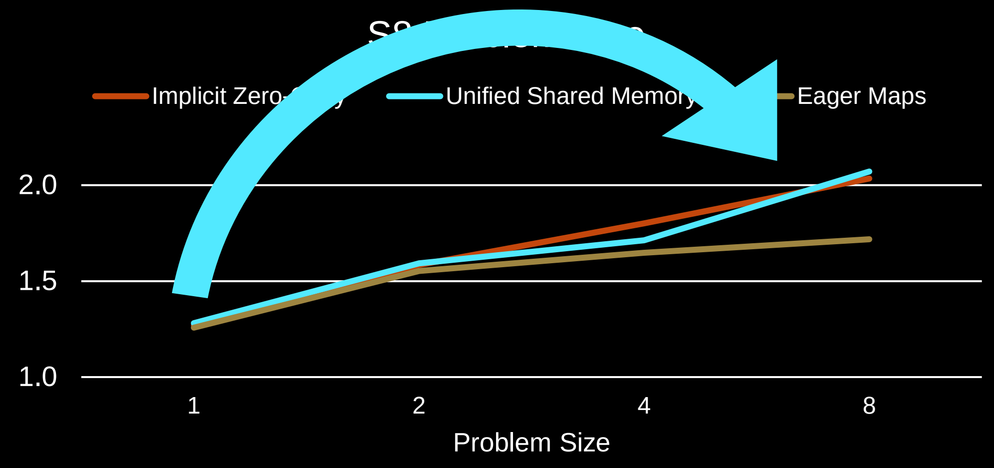
S64 Problem Size

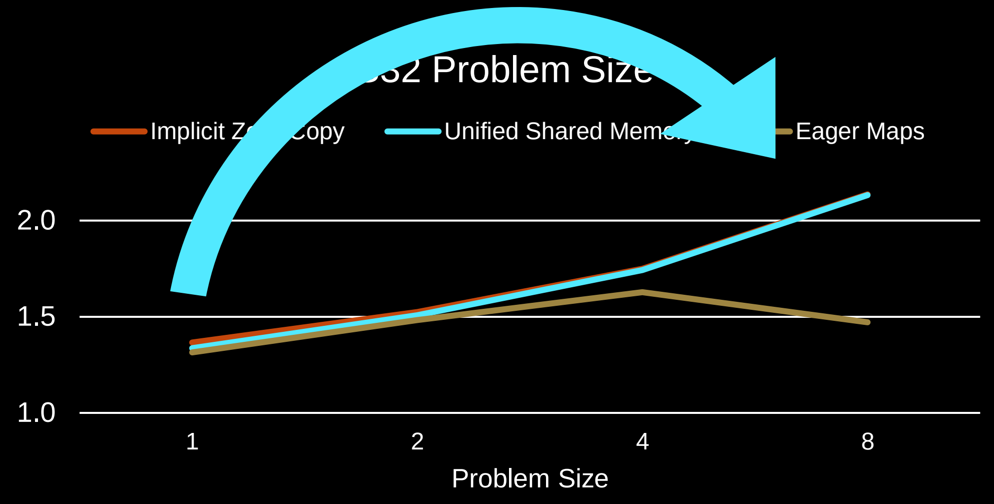


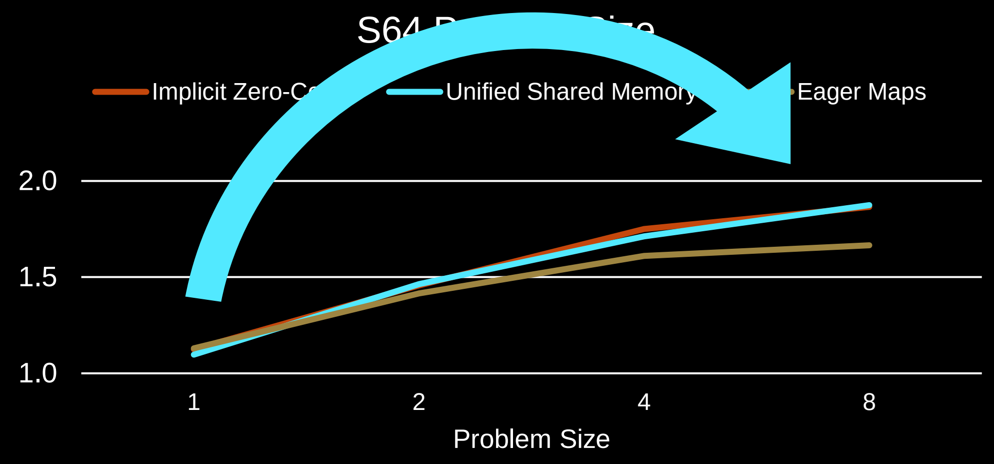
S128 Problem Size

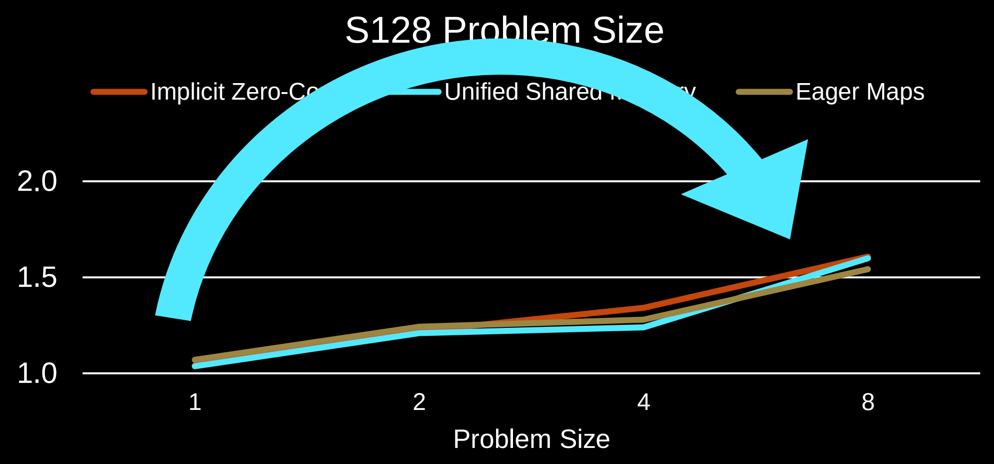


Why More OpenMP Host Threads Helps Zero-Copy?





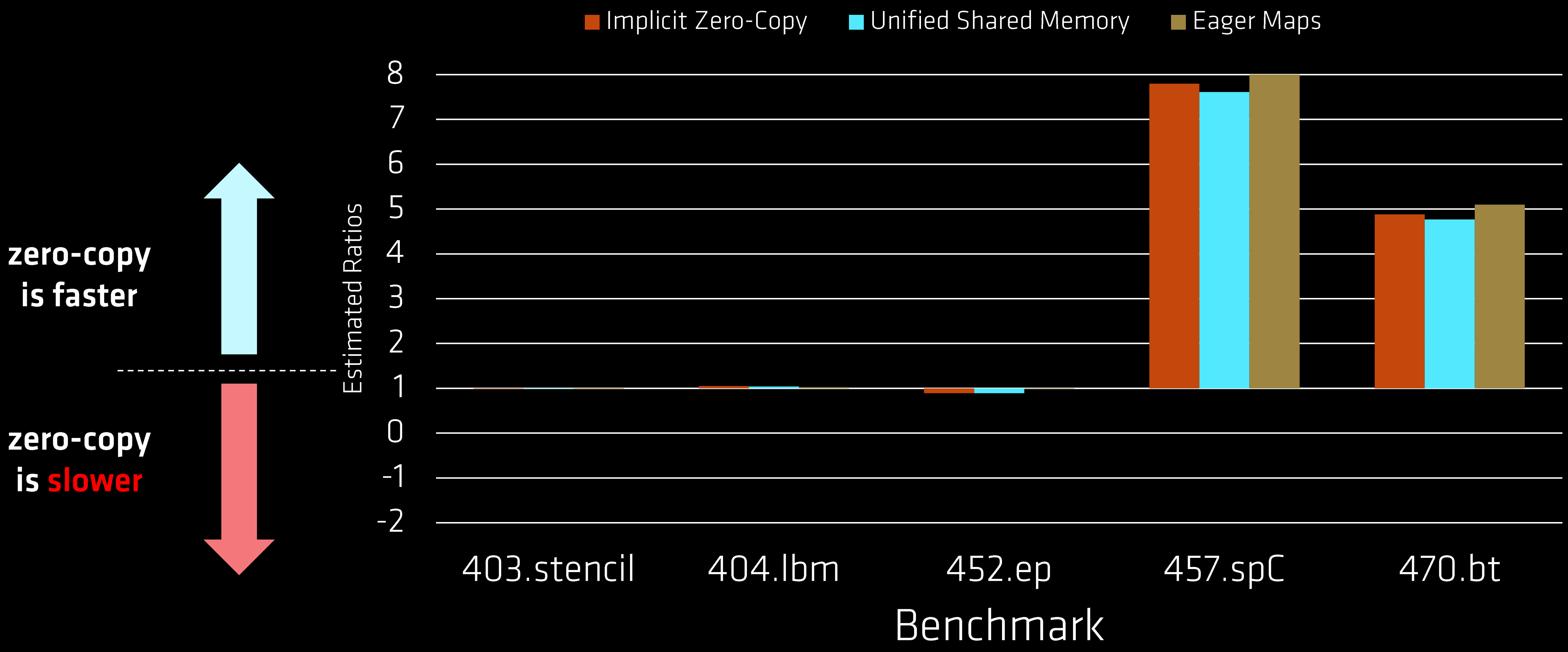




Why More OpenMP Host Threads Helps Zero-Copy?

ROCr call	1 OpenMP host thread			8 OpenMP Host Threads		
	Copy	Zero-Copy	Copy/Z-C	Copy	Zero-Copy	Copy/Z-C
	#calls	#calls	ratio	#calls	#calls	ratio
signal wait scacquire	351,653	99,627	3.53	1,360,088	738,483	1.84
memory pool allocate	23,277	19	1.23×10^3	20,848	90	231.64
memory async copy	307,607	3	1.03×10^5	1,124,258	3	3.75×10^5
signal async handler	194,848	0	N/A	491,492	0	N/A

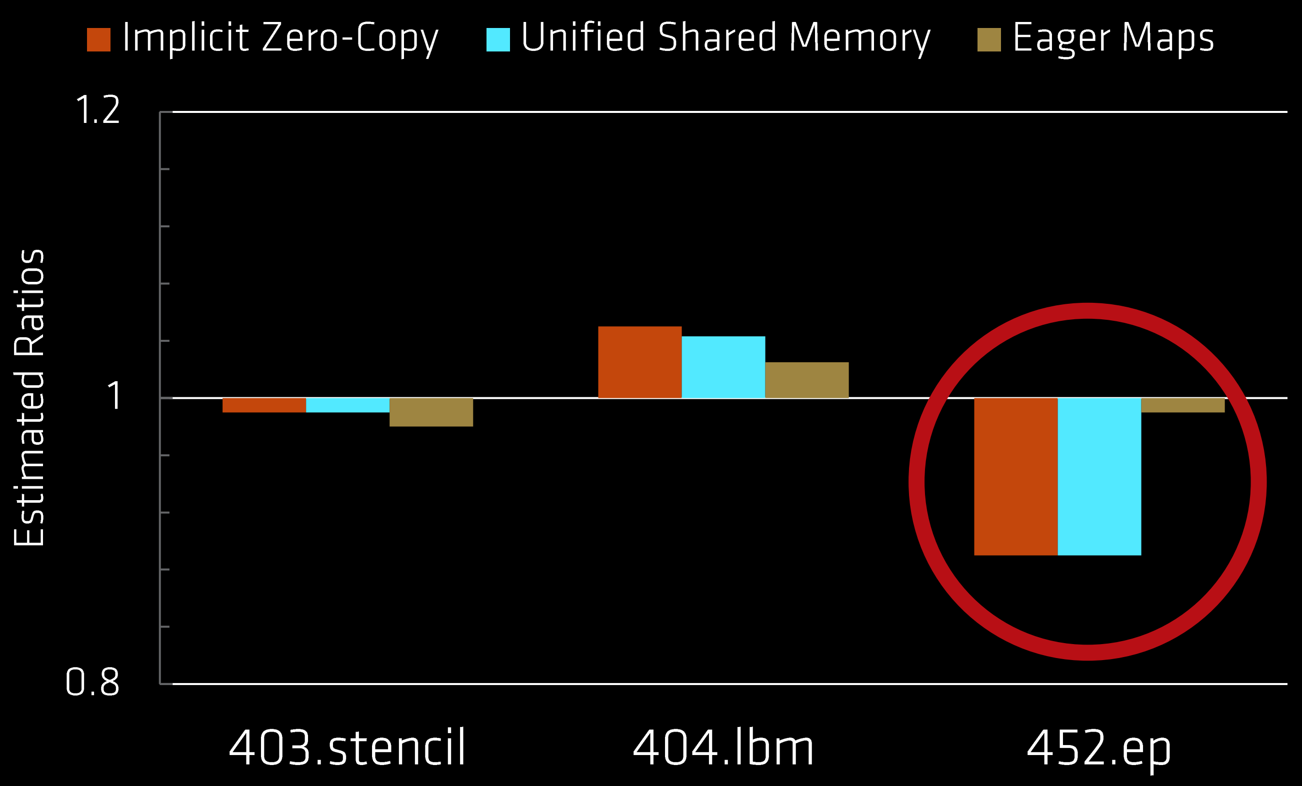
SPECaccel® 2023 Estimates*: Ratio Copy/Zero-Copy

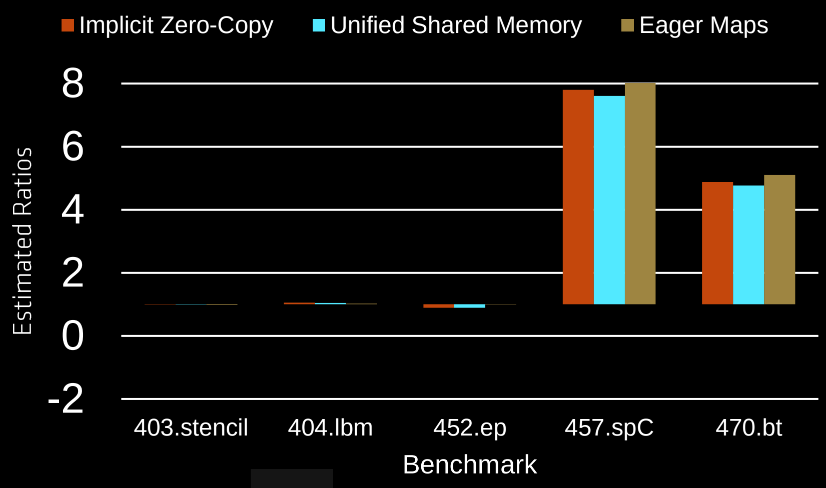


* According to SPEC rules of disclosure, our results are labeled as estimates because we ran the C/C++ subsets of the benchmarks.

[Public]

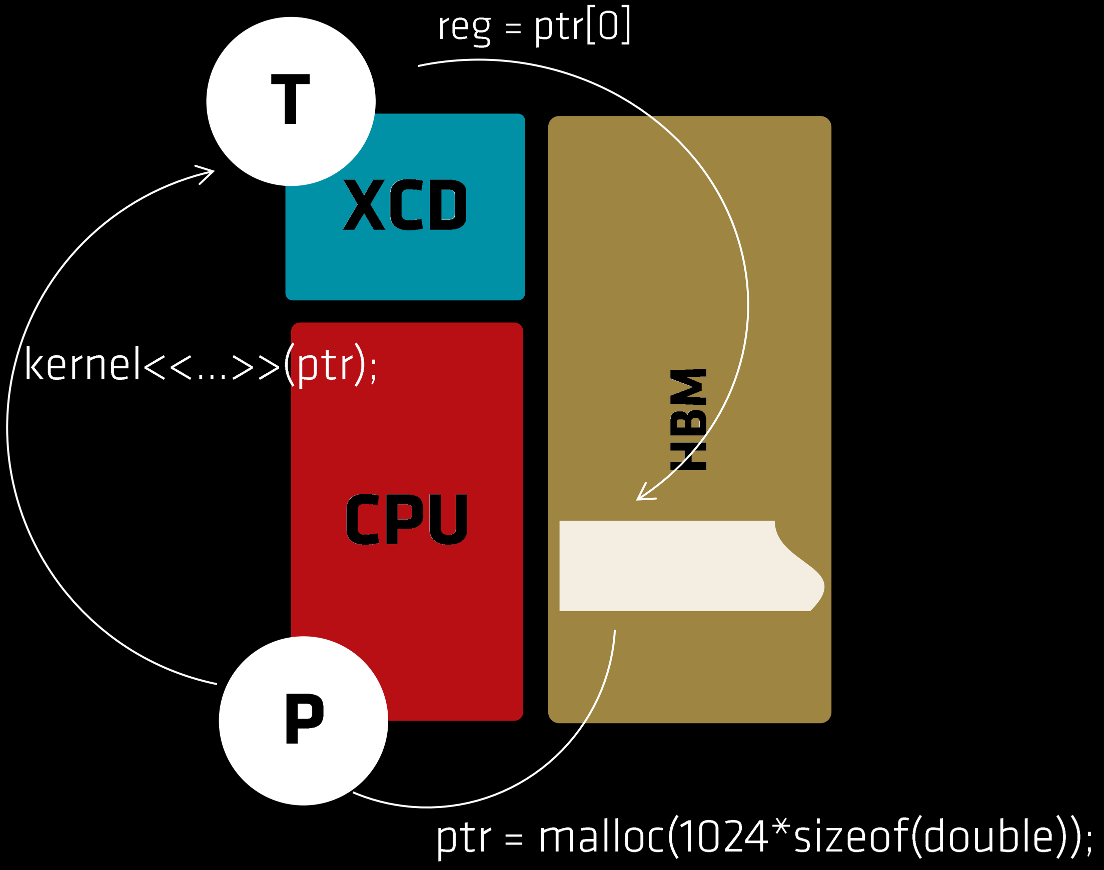
SPECaccel 2023 Estimates*: Zero-Copy Slow Downs

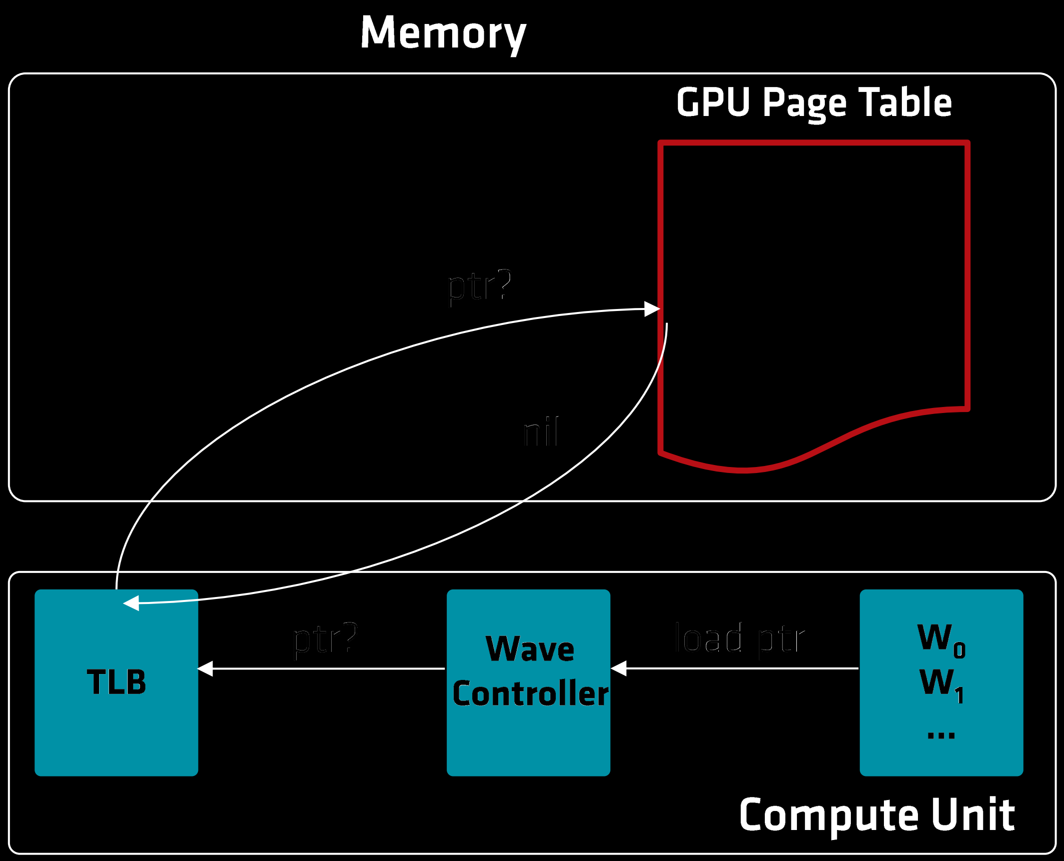




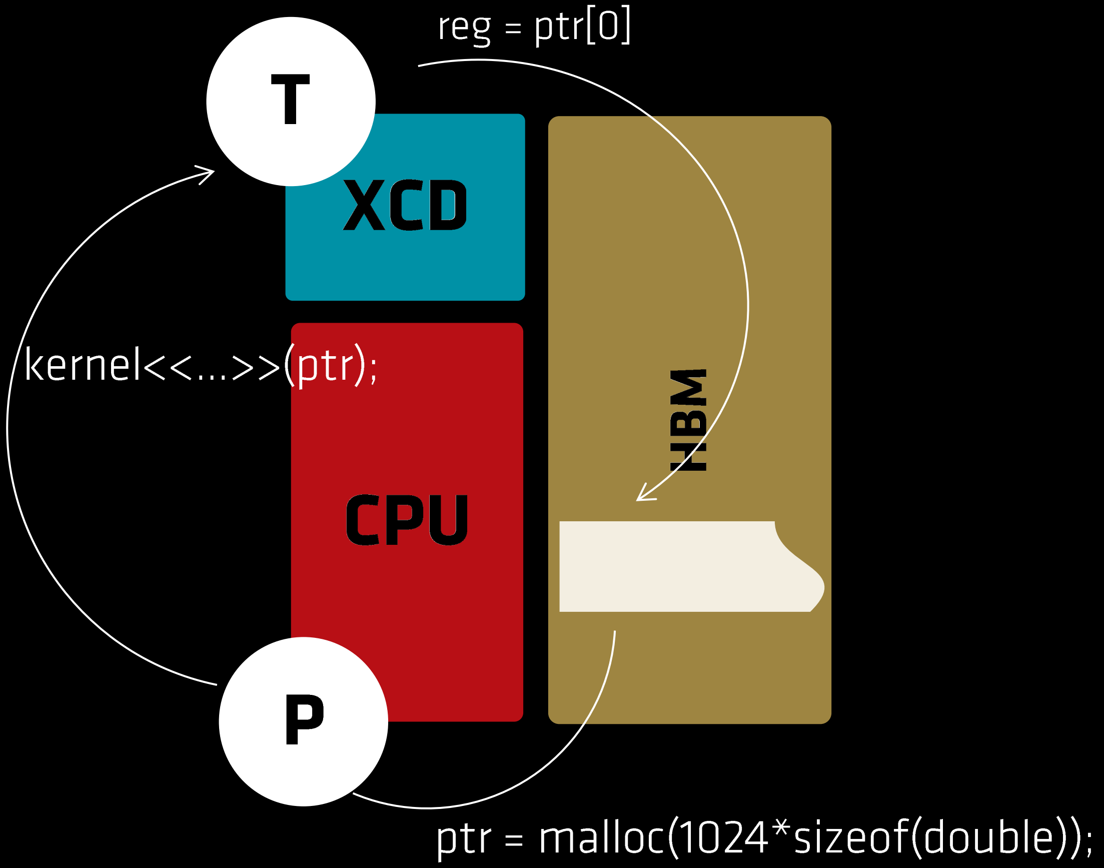
AMD
together we advance_

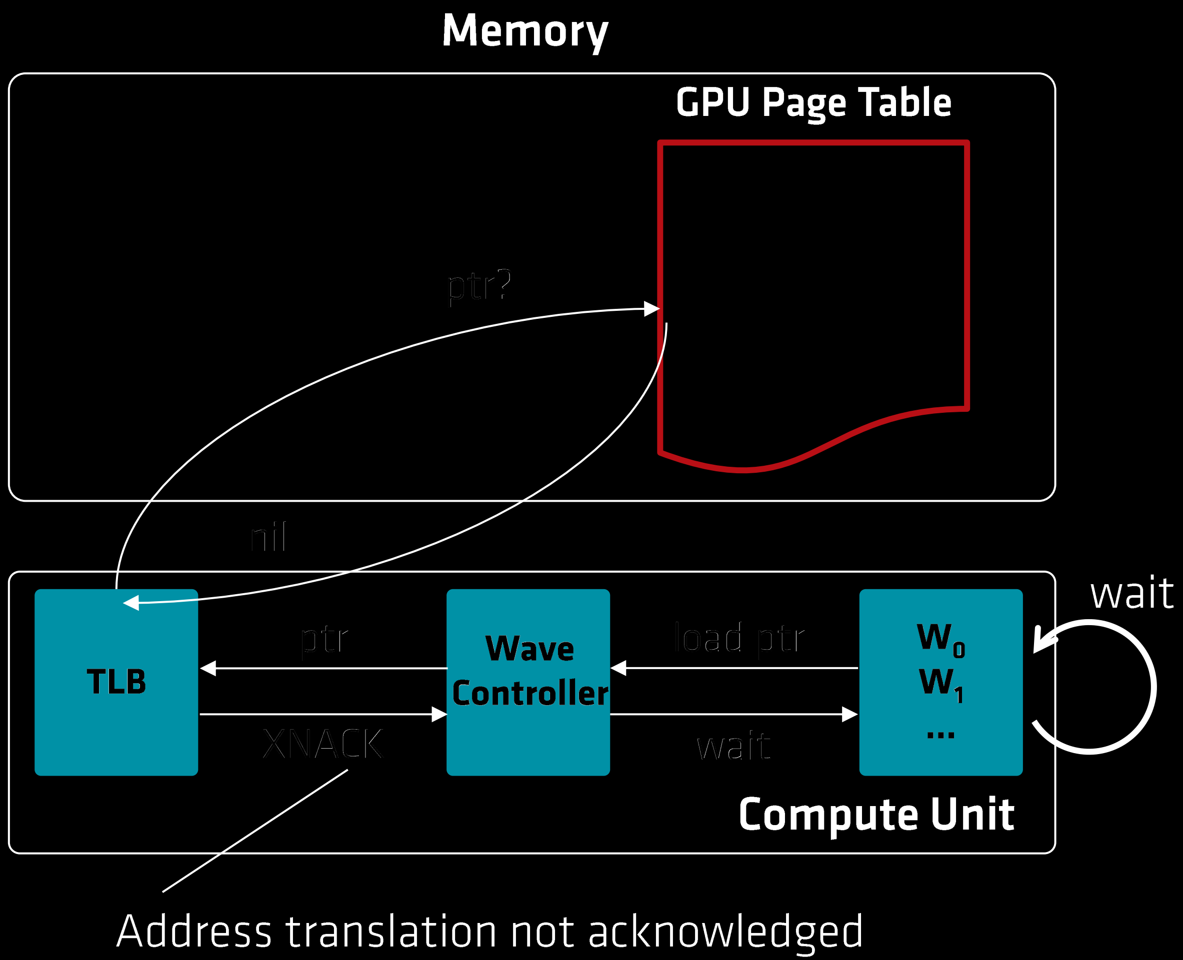
How to access CPU-allocated Memory on the GPU? XNACK



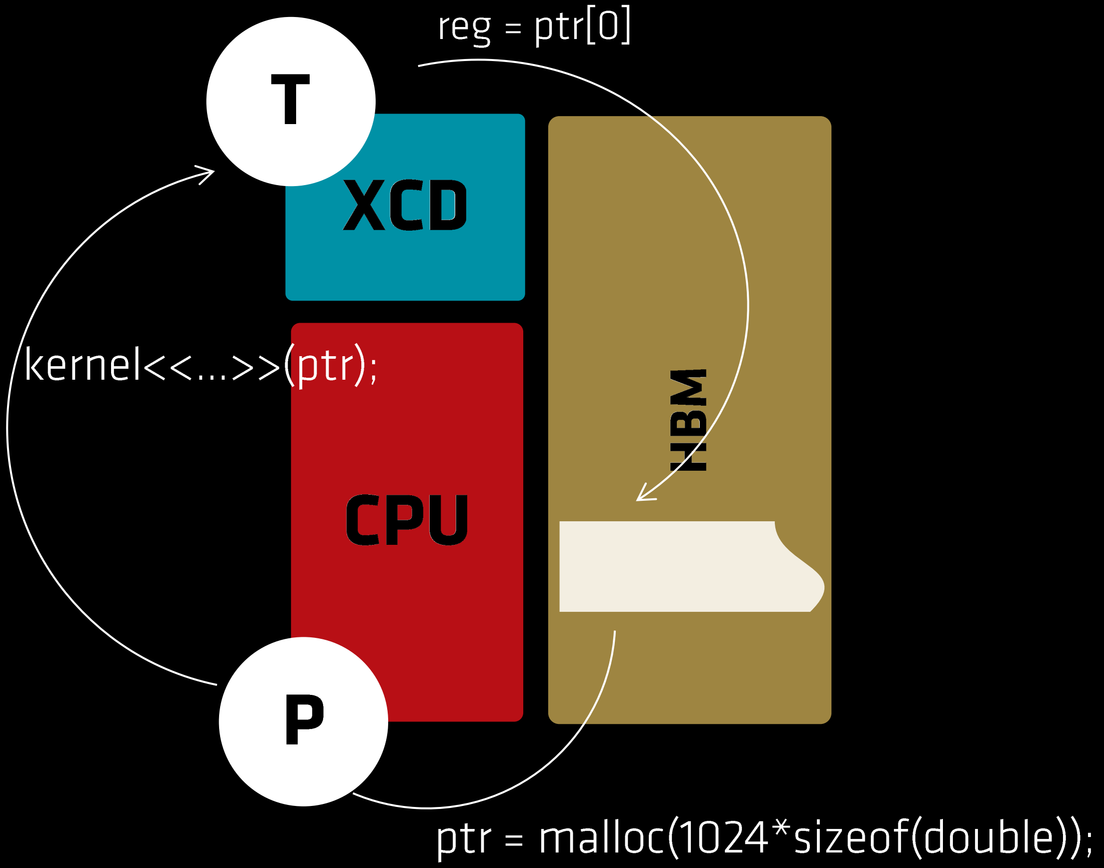


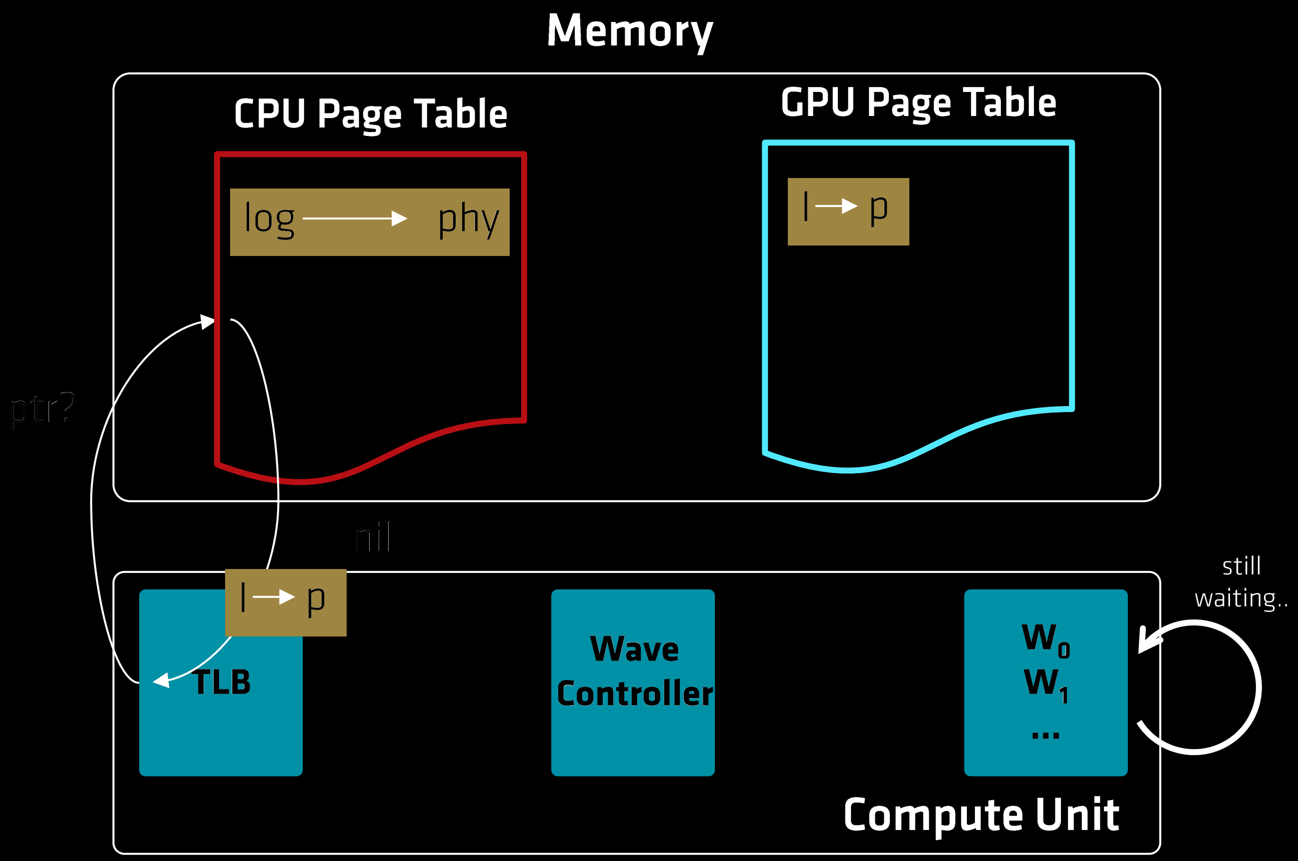
How to access CPU-allocated Memory on the GPU? XNACK



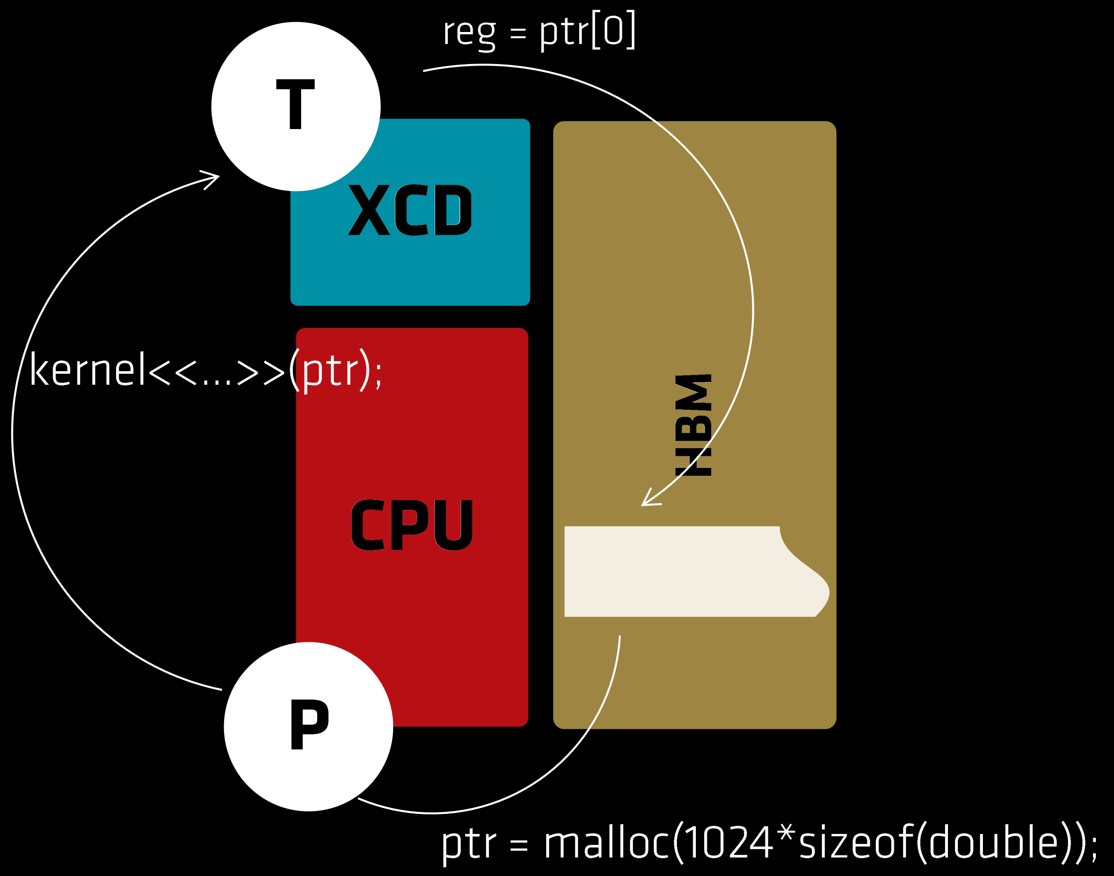


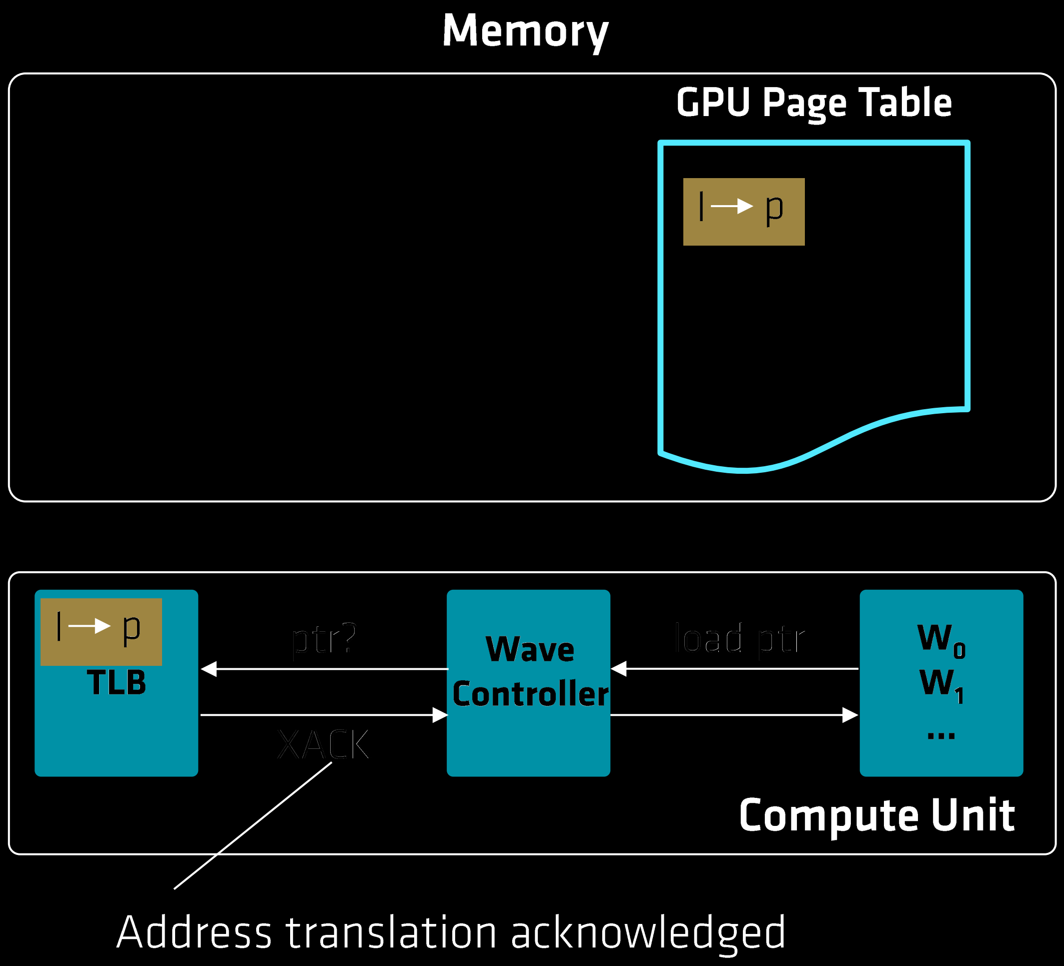
How to access CPU-allocated Memory on the GPU? XNACK





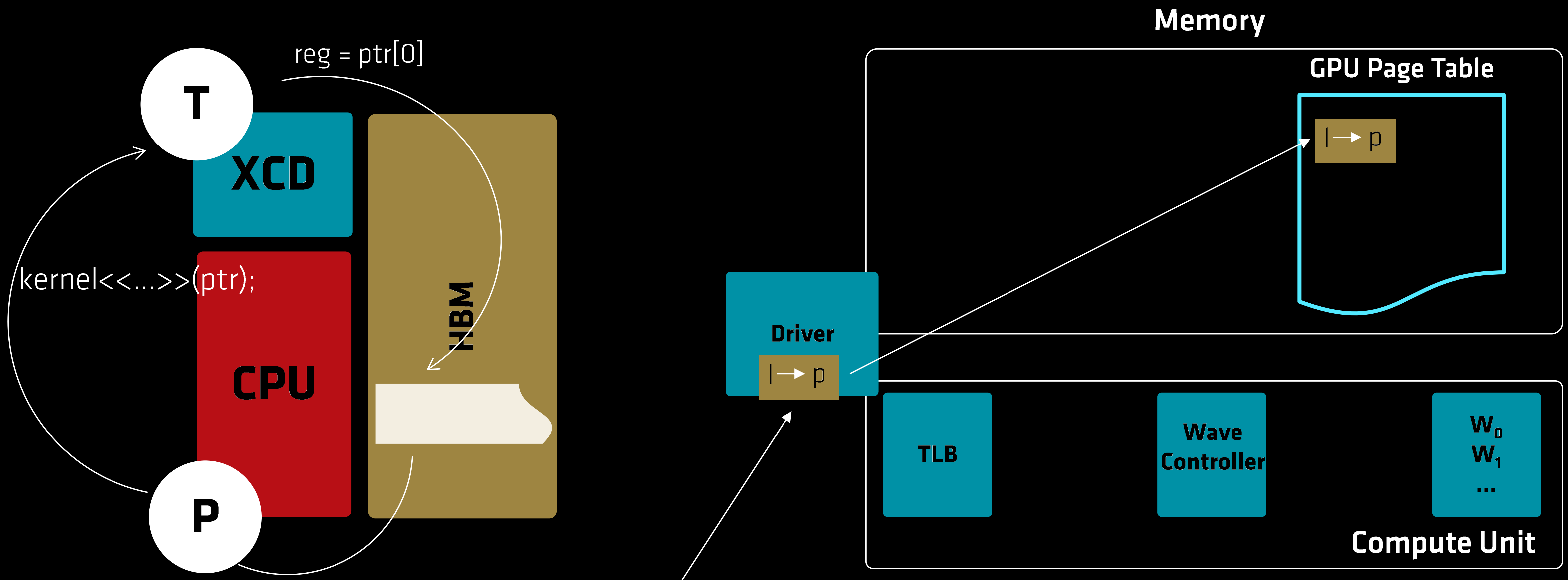
How to access CPU-allocated Memory on the GPU? XNACK





AMD
together we advance_

How to access CPU-allocated Memory on the GPU? Prefaulting




```
ptr = malloc(1024*sizeof(double));
gpu_page_table_prefault(ptr, 1024*sizeof(double));
```


AMD together we advance_

Unified Memory Overheads

- XNACK
  - First time a page is touched on the GPU
    - XNACK-replay cost
  - Page-by-page faulting
  - Typically shows up in a few of the first kernel executions of your applications

- Prefaulting the GPU page table
  - Done ahead of touching
  - Costs syscall + CPU page table walk + driver to copy page table entries to GPU page table
  - Whole array is prefaulted – not page-by-page

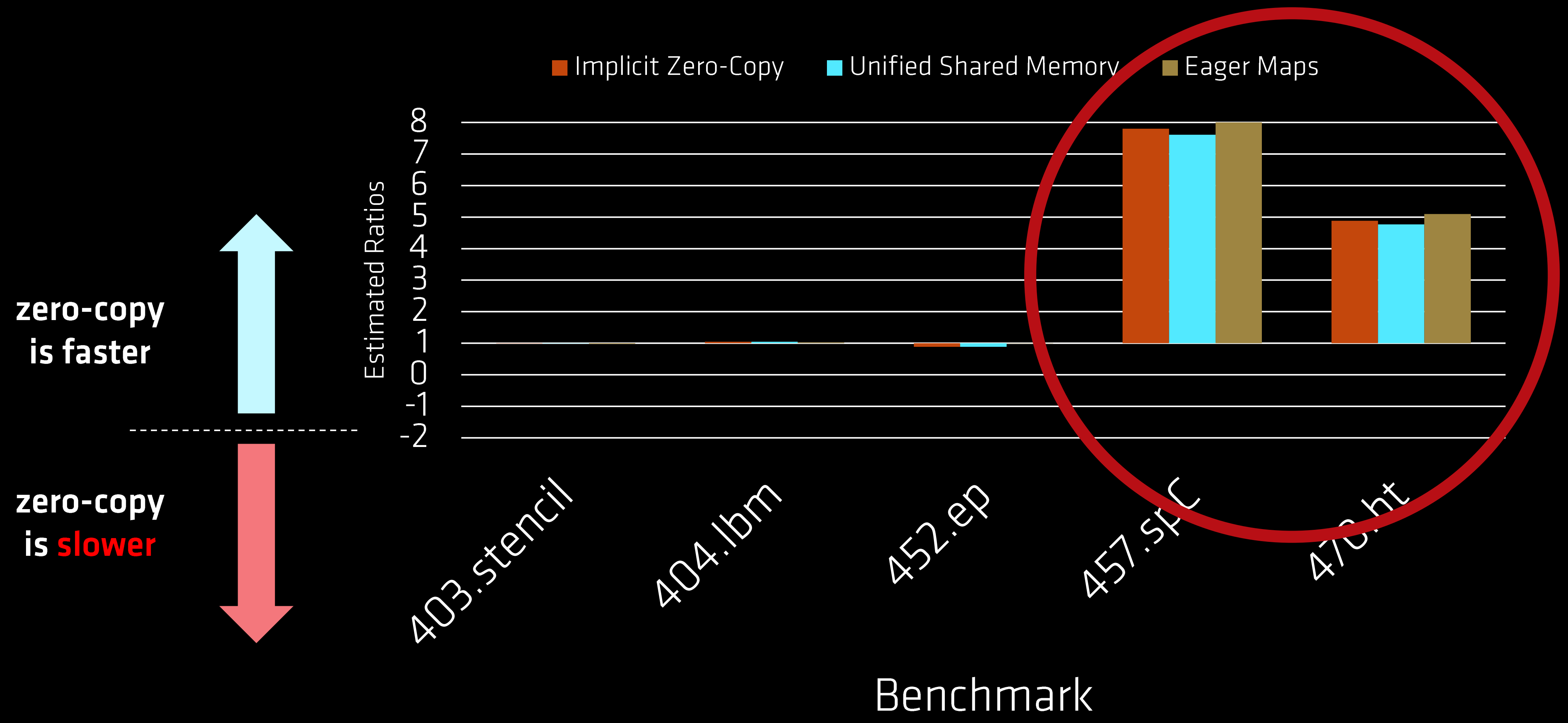
Overhead of First-Touch on GPU: 403.stencil, 452.ep

- Memory Copy: Sum of all ROCr calls to allocate and copy GPU-specific memory
- First Touch: Cost of running XNACK-replay

	Stencil		EP	
Overheads	**Memory Copy**	**First Touch**	**Memory Copy**	**First Touch**
Copy	$O(10^5)$	0	$O(10^5)$	0
Zero-Copy	0	$O(10^6)$	0	$O(10^6)$
Eager Maps	$O(10^4)$	0	$O(10^5)$	0

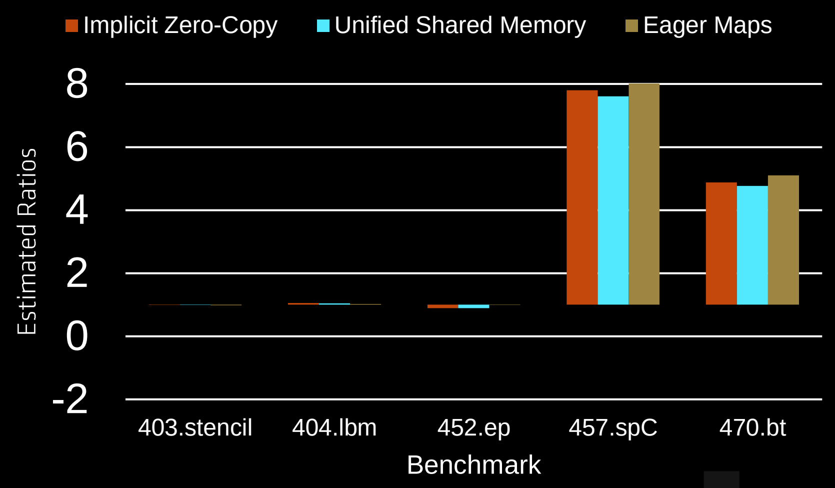
- Memory is initialized on the GPU
- No H2D memory copy needed
- First touch overhead only for zero-copy

Big Wins for Zero-Copy: 457.spC and 470.bt



AMD
together we advance_

Big Wins for Zero-Copy: 457.spC and 470.bt



Program stack for GPU arrays

- Three functions using program stack
- Copy: allocate+H2D/D2H copy at every function invocation
- Zero-Copy: pass stack pointer to target region

Zero-Copy does not pay for first touch overhead at every function invocation

- Same physical pages used across successive function calls
- Even though different data is stored on program stack
- This is more common than thought

```
void foo() {
  double A[N][M][K], B[M][N][K];
  #pragma omp target teams loop ..
    ..
    A[i][j][k] = B[j][i][k];
}

void bar() {
  double D[K][M][N];
  #pragma omp target teams loop ..
}
```

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

SPEC ACCEL is a trademark of the Standard Performance Evaluation Corporation.
See www.spec.org for more information about SPEC® benchmarks.

AMD together we advance_

AMD