Performance Assessment of OpenMP Compilers Targeting NVIDIA V100 GPUs

Joshua H. Davis[1], Christopher Daley, Swaroop Pophale, Thomas Huber, Sunita Chandrasekaran, Nicholas J. Wright

[1]jhdavis@udel.edu
@jhdavis_josh

Background and Motivation

- OpenMP is implemented by a growing number of compilers targeting accelerators
  - LLVM, GNU, IBM, Cray/HPE, AMD, Intel, NVIDIA
- ECP SOLLVE Verification and Validation Suite offers correctness status
- **What is the performance status of OpenMP offloading compilers?**
  - *What are the underlying causes of performance differences across compilers?*
  - *How should compiler and application developers tackle observed performance differences in compilers?*

jhdavis@udel.edu
@jhdavis_josh

Proxy App and Benchmark Suite

Selected real-world codes with potential to expose performance differences

- **`su3`**: complex number matrix-matrix multiply, from **MILC**[1]
- **`BabelStream`**: device memory bandwidth benchmark
- **`laplace`**: iterative Jacobi method solver for Laplace equation
- **`gpp`**: generalized plasmon-pole model, from **BerkeleyGW**[1]
- **`ToyPush`** (Fortran): electron sub-cycling, from **XGC1**[1]

[1]ECP Applications

jhdavis@udel.edu
@jhdavis_josh

Systems and Compilers

	Cori-GPU	Summit
Node architecture	Cray CS-Storm 500NX	IBM AC922
Node CPUs	2 × Intel Skylake	2 × IBM Power 9
Available cores per CPU	20 @ 2.40 GHz	21 @ 3.07 GHz
Node GPUs	8 x 16 GB NVIDIA V100	6 x 16 GB NVIDIA V100
CPU-GPU interconnect	PCIe 3.0 switch	NVLink 2.0

Baselines

Compiler	GPU offload	Cori-GPU version	Summit version
NVCC	CUDA	10.2.89	-
NVIDIA/PGI	OpenACC	20.4	-
Cray CCE	OpenMP	10.0.0 (LLVM version)	-
Cray CCE	OpenMP	9.0.0 (Classic version)	-
IBM XL	OpenMP	-	16.1.1-5
LLVM/Clang	OpenMP	11.0.0-git (#17d8334)	11.0.0-git (#17d8334)
GNU/GCC	OpenMP	-	9.1.0

jhdavis@udel.edu
@jhdavis_josh

Compiler and App Compatibility

Compiler	su3	babelStr.	laplace	gpp	ToyPush
NVCC (CUDA)	✓	✓	NI	NI	NI
PGI (OpenACC)	✓	✓	✓	✓	✓
Cray-llvm	✓	✓	✓	✓	✓
Cray-classic	✓	✓	✓	✓	✓
XL	✓	✓	✓	✓	✓
Clang	✓	✓	✓	✓	-
GCC	✓	✓	✓	✓	RE

No Implementation

Runtime Error

jhdavis@udel.edu
@jhdavis_josh

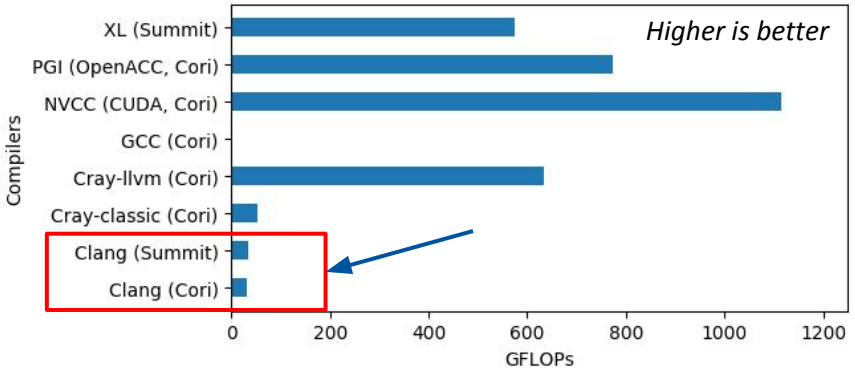
Summary of Recommendations (R)

	Recommendation	Compilers Affected
R1	Prefer combined constructs: interleaving code between teams and parallel harms performance	Clang 11.0.0-git (#17de334)
R2	Tune the kernel launch configuration: compiler-selected values are not always performant	Clang 11.0.0-git Cray-classic 9.0.0
R3	Avoid reductions where possible: they are inefficient for multiple compilers	Clang 11.0.0-git Cray-llvm 10.0.0
R4	When reductions are necessary, try using local variables to sequentialize some of the work	Cray-classic 9.0.0
R5	Ensure that the GPU is being sent enough work: OpenMP runtime overhead can be significant for some compilers	GCC 9.1.0 Clang 11.0.0-git
R6	Use GCC mainly for correctness at the current time	GCC 9.1.0

jhdavis@udel.edu
@jhdavis_josh

R1: Prefer combined constructs

- `su3`: **~20x more** DRAM write transactions in **Clang** than **CUDA**
- Restructuring directives gives an **18x speedup**



GFLOPs per compilers for `su3`, unoptimized (V0)

jhdavis@udel.edu
@jhdavis_josh

R1: Prefer combined constructs

- Removing code between **`teams`** and **`parallel`** constructs give similar benefits to combined constructs
  - 18x speedup in **`su3`** with **Clang**
- Fork-join model maps poorly to GPU
  - Excess memory flushes
  - Help the compiler by creating parallelism upfront, allow for SPMD transformation

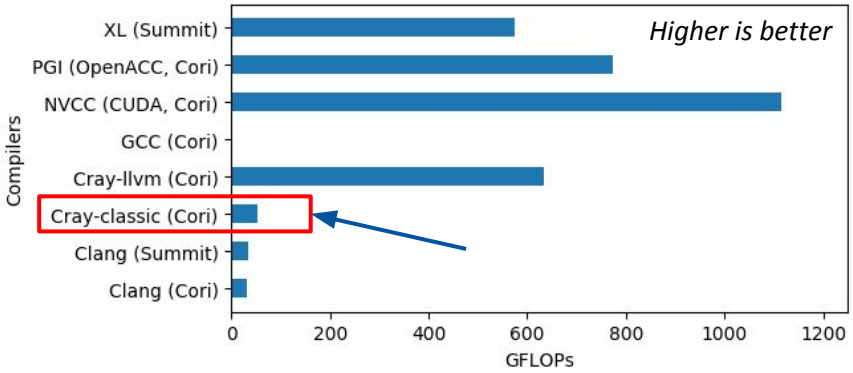
```
#pragma omp target teams distribute
    for (int i = 0; i < total_sites; ++i) {
#pragma omp parallel for collapse(3)
        // 3 for loops ...
```

```
#pragma omp target teams
#pragma omp parallel
{
    // compute istart, iend for each team ...
    for (int i = istart; i < iend; ++i) {
#pragma omp for collapse(3)
        // 3 for loops ...
```


jhdavis@udel.edu
@jhdavis_josh

R2: Tune the runtime configuration

- **2.03x** speedup in **Cray-classic** after tuning **`num_teams`** in **`su3`**
  - Change default 81920 to 10000
- **4.45x** speedup in **Clang** after tuning **`num_teams`** and **`num_threads`** (after directive restructuring)
  - Change defaults <128, 128> to <1600, 64>



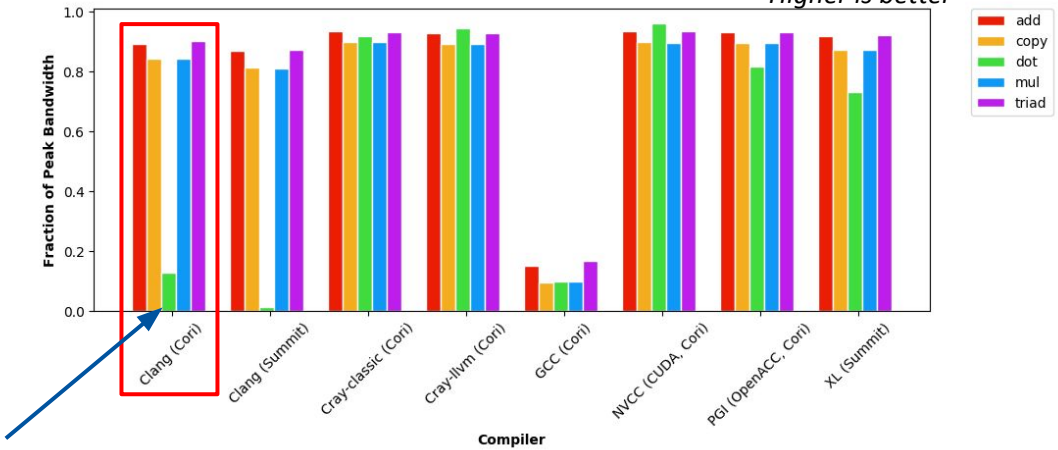
*GFLOPs per compilers for **`su3`**, unoptimized (V0)*

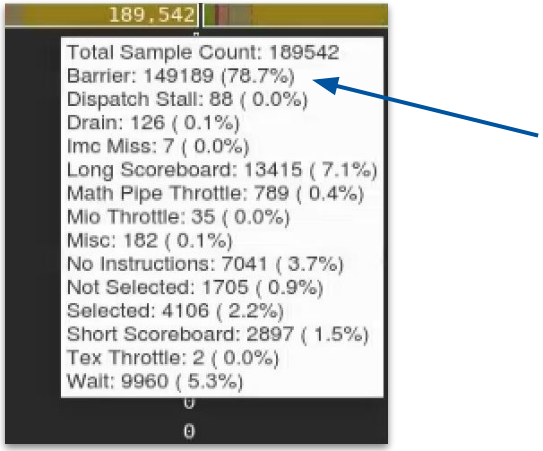
jhdavis@udel.edu
@jhdavis_josh

R3: Reductions can be slow

- **Clang**: dot product kernel in **`BabelStream`** is slow due to use of **+** reduction
  - Introduces barrier latency

Higher is better



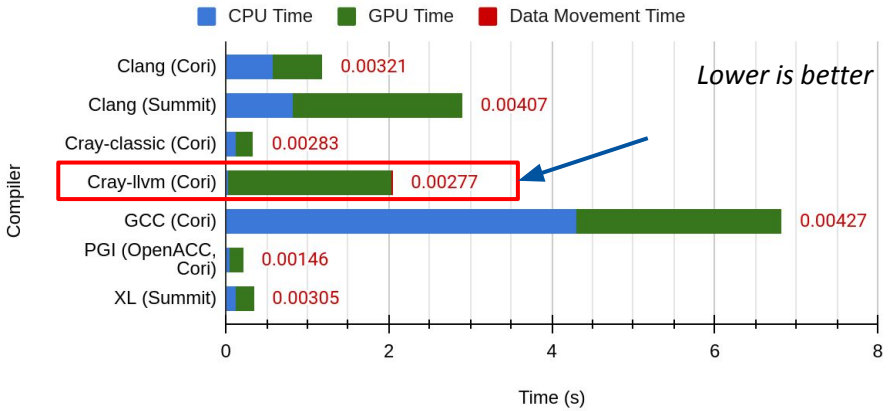


Fraction of peak memory bandwidth per compiler, kernel

jhdavis@udel.edu
@jhdavis_josh

R3: Reductions can be slow

- **Cray-llvm:** latency issues caused by **`max`** reduction in **`laplace`**
  - "Long Scoreboard" samples: waiting on L1 cache
- Changing the reduction type to **+** instead of **`max`** gives an **8.3x speedup**
  - **`max`** reduction version has **~2745x more** atomic and L2 atomic transactions compared to **+**

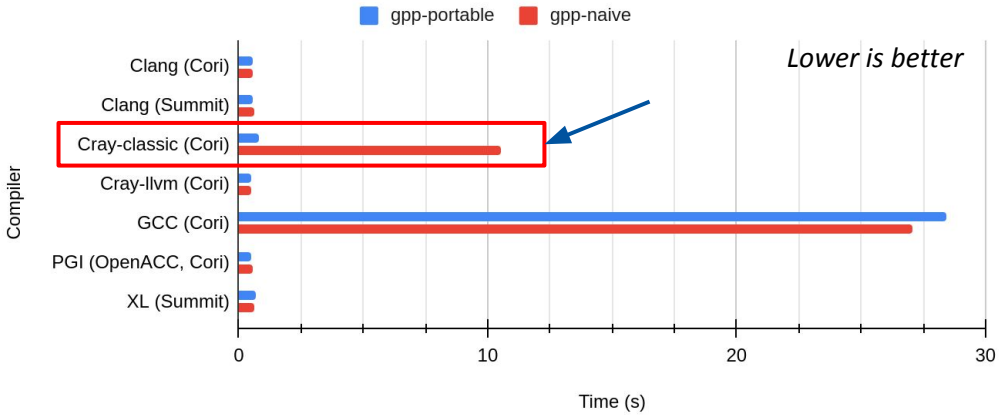


Composition of `laplace` kernel runtime

jhdavis@udel.edu
@jhdavis_josh

R4: Mitigate reduction slowdowns in **Cray-classic**

- **`gpp`** uses an **`+`** reduction for an important kernel
- Developers mitigated some reduction performance issues
- **`gpp-portable`** performs some reduction work sequentially in the innermost loop on a local variable, and then reduces on that local variable



`gpp-portable` *vs* ***`gpp-naive`***
execution time (s)

jhdavis@udel.edu
@jhdavis_josh

R5: Runtime overheads can be high

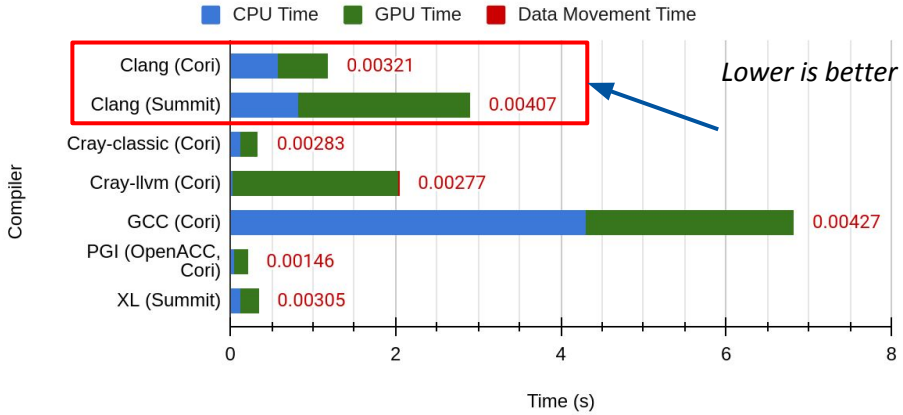
NVTX Range Time = GPU Time + CPU Time + Data Movement Time

```
id1 = nvtxRangeStartA("launch");
#pragma omp target teams distribute parallel for
for (i = 1; i <= height; ++i) {
    // stencil update ...
}
nvtxRangeEnd(id1);
```

- **`laplace`** and **`ToyPush`** launch many small kernels
  - More sensitive to OpenMP runtime overhead
- Nvprof with NVTX range enables separating out CPU, GPU, data movement time
  - CPU time corresponds to runtime overhead

jhdavis@udel.edu
@jhdavis_josh

R5: Runtime overheads can be high



Composition of `laplace` kernel runtime

- **GCC**, **Clang** perform poorly with `laplace`
- High CPU time to blame, i.e., high OpenMP runtime overhead
  - High GPU time in Clang on Summit due to further elevated barrier latency on reduction

jhdavis@udel.edu
@jhdavis_josh

Summary of Recommendations (R)

	Recommendation	Compilers Affected
R1	Prefer combined constructs: interleaving code between teams and parallel harms performance	Clang 11.0.0-git (#17de334)
R2	Tune the kernel launch configuration: compiler-selected values are not always performant	Clang 11.0.0-git Cray-classic 9.0.0
R3	Avoid reductions where possible: they are inefficient for multiple compilers	Clang 11.0.0-git Cray-llvm 10.0.0
R4	When reductions are necessary, try using local variables to sequentialize some of the work	Cray-classic 9.0.0
R5	Ensure that the GPU is being sent enough work: OpenMP runtime overhead can be significant for some compilers	GCC 9.1.0 Clang 11.0.0-git
R6	Use GCC mainly for correctness at the current time	GCC 9.1.0

jhdavis@udel.edu
@jhdavis_josh

Conclusions and Future Work

- There is room for improvement in OpenMP implementations
  - Compiler developers should prioritize improving reductions and reducing overheads
  - Applications likely benefit more from an improved OpenMP compiler than the next GPU generation
- Roofline analysis can be misleading — app can appear to be near memory roofline due to excess data movement introduced by compiler
- Profilers, including nvprof and Nsight Compute, can be used to reveal which directives are inefficient for a compiler and why

jhdavis@udel.edu
@jhdavis_josh

Future Work

- Evaluate on other GPUs or accelerators than the V100
  - AMD, Intel Xe
- Evaluate on other compilers, like AMD AOMP, Intel ICC
  - Future NVIDIA HPC SDK OpenMP support
- Something like the V&V suite: public suite of diverse real-world mini-apps

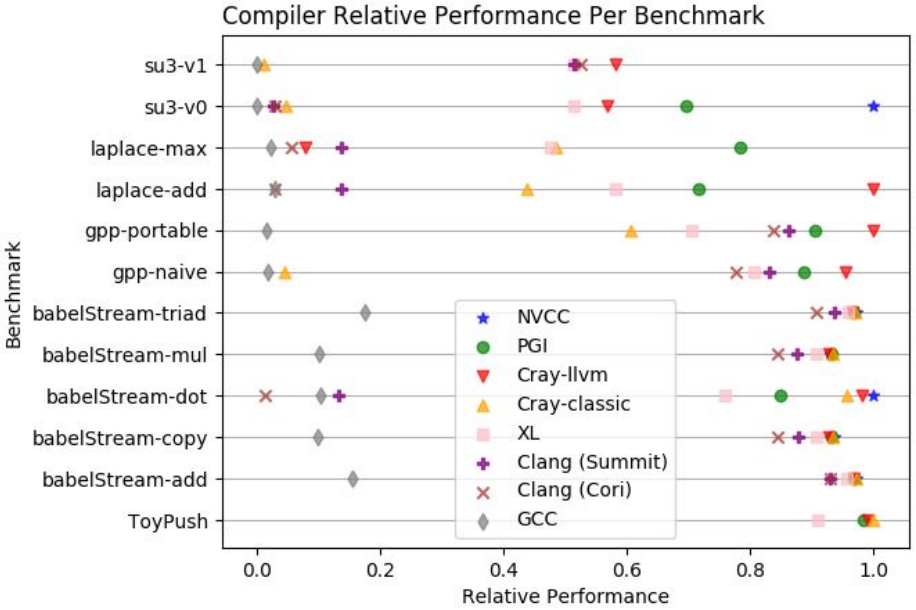
jhdavis@udel.edu
@jhdavis_josh

Acknowledgements

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. This research also used resources of the Oak Ridge Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. The authors would like to thank Doug Doerfler and Rahul Gayatri for helpful discussion about the su3 benchmark and useful research directions for this project.

jhdavis@udel.edu
@jhdavis_josh

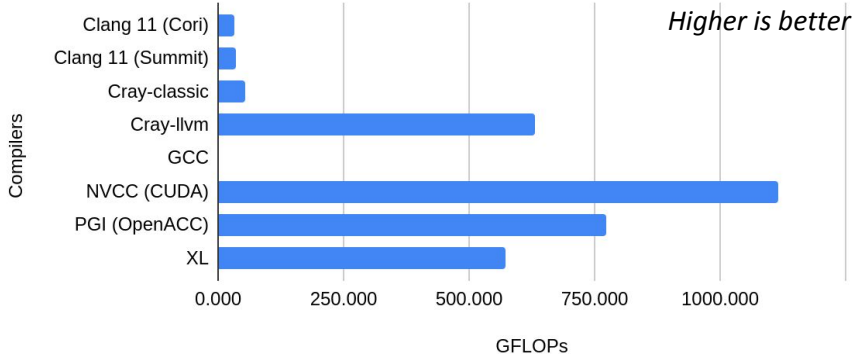
Performance Summary



jhdavis@udel.edu
@jhdavis_josh

su3 Results

- **`su3`** is a complex number matrix-matrix multiply proxy app
- Cray-classic performance improves after tuning **`num_teams`**
  - Tuned to 10000, greater than Clang-tuned value (1200), less than default (81920)
- Clang uses **~20x more** DRAM write transactions than CUDA



*GFLOPs per compilers for **`su3`**, unoptimized (V0)*

`su3`: Removing interleaving code

- Directive restructuring led to an **18x speedup** in Clang by reducing DRAM data movement
- Removing code between **`teams`** and **`parallel`** constructs

```
#pragma omp target teams distribute
for(int i = 0; i < total_sites; ++i) {
    #pragma omp parallel for collapse(3)
    // for loops...
```

```
#pragma omp target teams {
    #pragma omp parallel {
        // compute istart, iend...
        for (int i = istart; i < iend; ++i) {
            #pragma omp for collapse(3)
            // for loops...
```

su3 Results: Runtime configuration

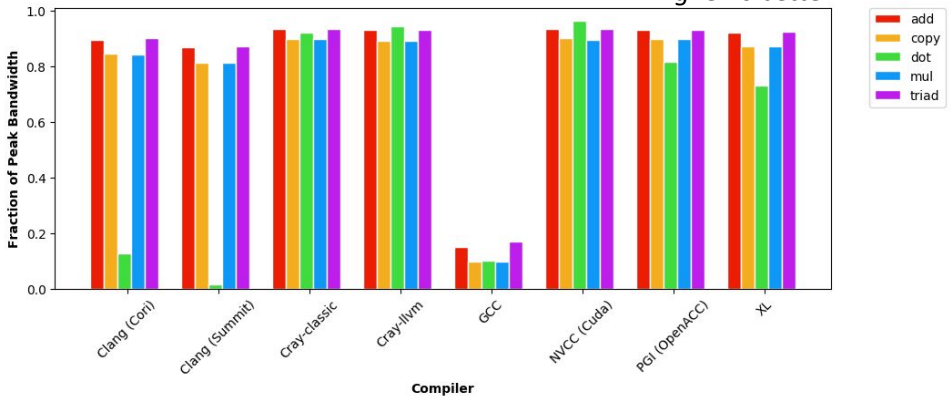
- The tuning of runtime configuration is a major factor for Clang after restructuring the directives
- Clang defaults to 128 teams with 128 threads per team, while our tuned values of 1600 teams with 64 threads each is **4.45x faster**.

		Number of Threads per Team				
		32	64	96	128	160
Number of Teams	128		156.035		**154.44**	
	400		376.596			
	800		646.489			
	1200		701.546			
	1600	582.164	**720.579**	592.331	502.128	503.574
	2000		551.603			

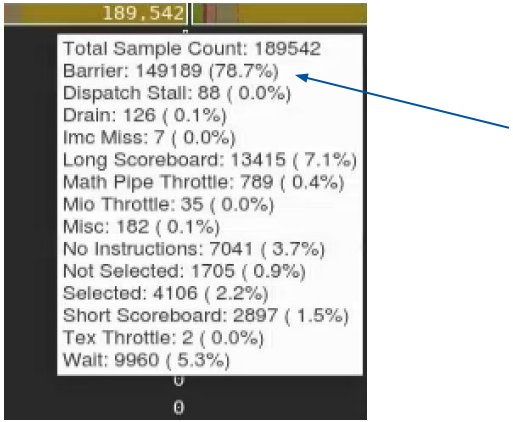
babelStream Results

- babelStream is a memory bandwidth benchmark
- The dot product kernel uses an + reduction, and shows poor performance on Clang
- Using Nsight source view, we find the the reduction induces barrier latency

Higher is better

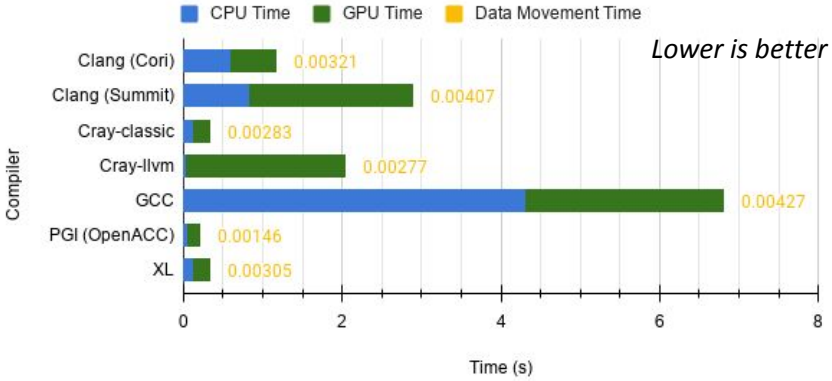


Fraction of peak memory bandwidth per compiler, kernel



`laplace` Results

- **`laplace`** launches many small kernels, so it is more sensitive to OpenMP runtime overhead. It also has an **`max`** reduction.
- Used an NVTX range to separate out GPU, CPU, and data movement time



*Composition of **`laplace`** kernel runtime*

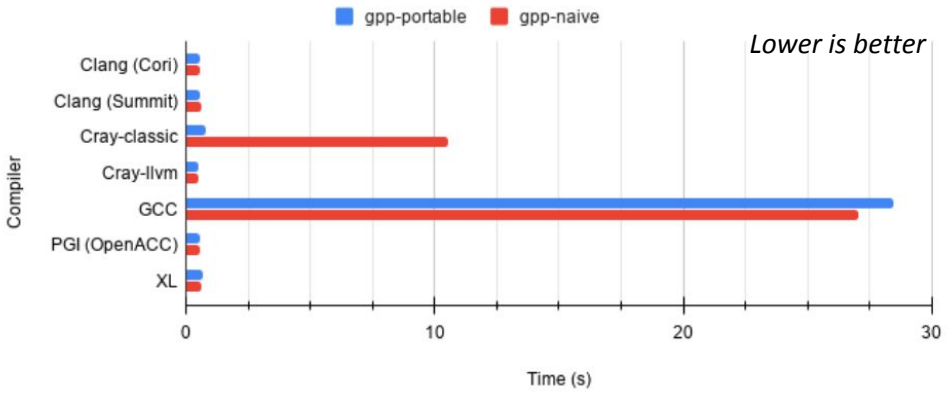
```
id1 = nvtxRangeStartA("launch");
#pragma omp target teams distribute parallel for
reduction(..) collapse(2)
for (i = 1; i <= height; ++i) {
    // stencil update...
}
nvtxRangeEnd(id1);
```

`laplace` Results: Cray-llvm

- From Nsight Compute: **`max`** reduction also poses a latency problem for Cray-llvm
- However, the latency samples are mostly “Long Scoreboard” rather than barrier.
  - Indicates warps are waiting on the L1 cache.
- Changing the reduction type to **`+`** instead of **`max`** gives an **8.3x speedup.**
- From Nsight Compute SASS analysis: Cray-llvm implements the **`max`** reduction less efficiently.
- **`max`** reduction version has **~2745x more** atomic and L2 atomic transactions compared to **`+`**.

`gpp` Results

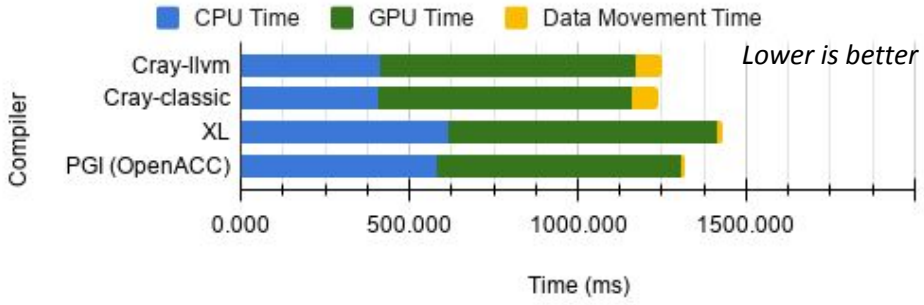
- `gpp` uses an `+` reduction for an important kernel
- Developers mitigated some reduction performance issues
- `gpp-portable` performs some reduction work sequentially in the innermost loop on a local variable, and then reduces on that local variable



`gpp-portable` vs `gpp-naive`
execution time (s)

`ToyPush` Results

- **`ToyPush`** is a larger mini-app
- Exemplifies the pattern shown in Laplace
- Large number of short-running kernels: likely sensitive to overhead
- Lower data movement in XL, PGI: optimization copies data to pinned memory in chunks before moving



***`ToyPush`** execution time (s)*

laplace Results: Kernel Launch Latency

- Using an empty kernel and NVTX markers (as used for the application itself), we observe all compilers show a launch latency of less than 2 microseconds.
- So high GPU time in Cray-llvm is likely be caused by something else

Compiler	NVTX Duration (us)	GPU Exec Time (us)	Runtime Overhead (us)
Clang	38.43	1.664	36.766
Cray-llvm	24.054	1.632	22.422
Cray-classic	14.308	1.024	13.284