National Aeronautics and Space Administration

Is OpenMP 4.5 Target Off-load Ready for Real Life? A Case Study of Three Benchmark Kernels

Jose M. Monsalve Diaz (UDEL), Gabriele Jost (NASA), Sunita Chandrasekaran (UDEL) and Sergio Pino(ex-UDEL)

November 13, 2018

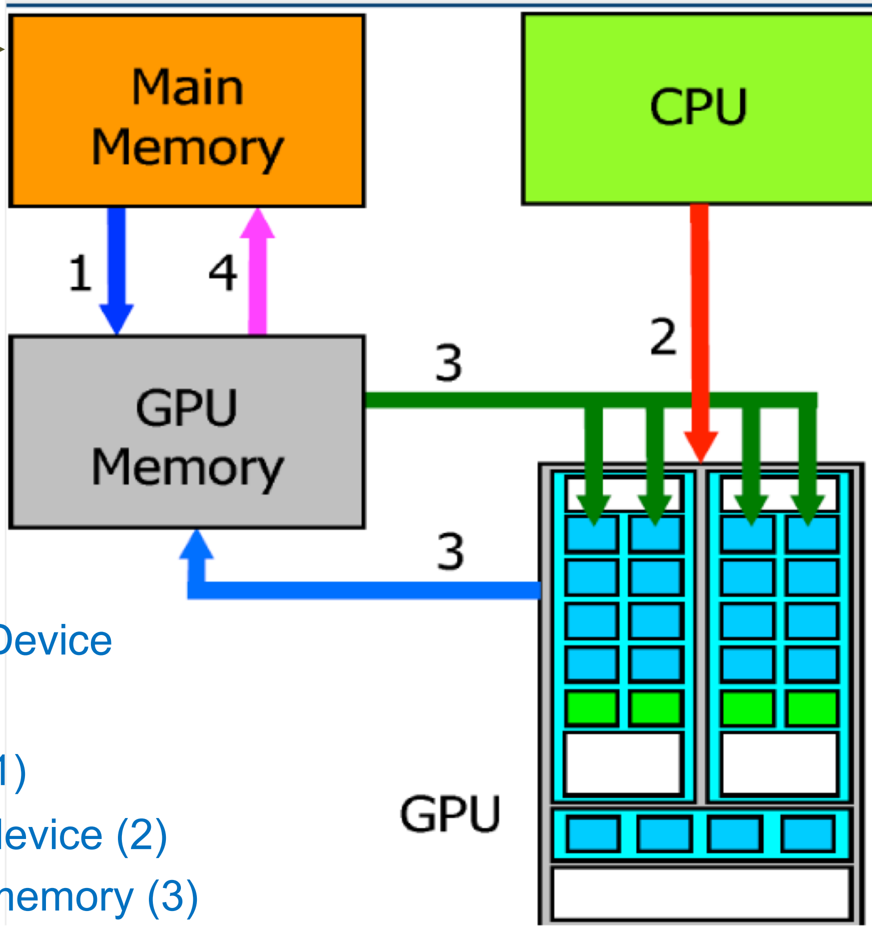
SC18 Booth Talk

Outline

- Introduction
  - Device Accelerated Programming
  - The OpenMP Target Concept
- Benchmark descriptions
- Porting OpenACC (2011) to OpenMP 4.0/4.5 (2013)
- Timings and Performance Analysis:
  - Evaluation Environment
  - Comparing compilers
  - Comparing OpenMP 4.5 to OpenACC
- Summary and Conclusions
- Discussion

Device Accelerated Computing

Example:
Device is a GPU



- Device Accelerated Programming
  - Identify and off-load compute kernels
  - Express parallelism within the kernel
  - Manage data transfer between CPU and Device
- Execution flow
  - Data copy from main to device memory (1)
  - CPU initiates kernel for execution on the device (2)
  - Device executes the kernel using device memory (3)
  - Data copy from device to main memory (4)

OpenMP and device offloading

- Essential tasks:
  - Identify compute kernels and offload to the device
  - Describe parallelism in the compute kernel
  - Manage data transfer between host and device

OpenMP 4.0 (since 2013)

target [data]
declare target
target update
target teams distribute [simd]
target teams distribute parallel for [simd]
… and other API Calls …

OpenMP 4.5	OpenMP 5.0
target enter data target exit data target simd (combined construct) … and other API Calls …	target declare mapper target parallel loop target teams loop omp_get_device_number … and other API Calls …

Important OpenMP Constructs and Clauses

#pragma omp target or $!omp target

- Offloads code region to the device and creates a data environment on the device

#pragma teams

Re-use of existing constructs

- Start kernel on the device threads

#pragma teams distribute, parallel for, simd

- 3 Levels of parallelism
- Distribute the work across the teams and threads within each team

#pragma omp target map(*map-type: list*)

- Map a variable to/from the device data environment

NPB Benchmark Descriptions

- **FT** = Discrete 3D Fast Fourier Transform
  - Requires all-to-all data data transfers
  - Compiler Challenges:
    - Usage of complex data structures required manually handling real and imaginary parts separately; function calls in inner loops benefit from manual inline of function calls

Class A: 256x256x128
Class C: 512x512x512

- **LU-HP** = Lower-Upper Gauss Seidel Solver using a hyperplane method
    - A pipelined algorithm requires explicit thread-to-thread synchronization, which is not suitable for device execution
  - Compiler Challenges:
    - Data layout is not optimal for device execution; shared array data structures increase data transfer

Class A: 64x64x64
Class C:162x162x162

- **MG** = Multi-Grid Solvers on a sequence of meshes
  - Requires long and short distance data transfers between grids
  - Memory intensive
  - Compiler Challenges:
    - 3D data structures required manual linearization

Class A: 256x256x256
Class C: 512x512x512

- NPB benchmark offers different classes (Problem size) – S thru E; we used sizes A and C for our study

General Implementation Strategy: Translating OpenACC to OpenMP 4.5

- Start out with the existing NPB 2.5 OpenACC Implementation developed in 2014 by Xu et al. (see Ref 1.)
- Translate OpenACC to OpenMP 4.5 matching constructs if available
- OpenACC:
  - A parallel programming model originally targeted toward device accelerated programming

Open ACC 2.5	OpenMP 4.5
parallel	target
{enter, exit} data	target {enter,exit} data
parallel present (a1,a2,…)	target map (alloc: a1, a2,..
loop [gang/worker/vector]	distribute / parallel for /simd
device_ptr	is_device_pointer
kernels	----

Tell compiler that data is already present on the device

Compiler will detect if data is already present on the device

Not available in OpenMP!
Places burden of dependence analysis on the compiler

FT Implementation

3D partial differential equation using an Fast Fourier Transform (FFT)

- Complex data:
  - Treat real and imaginary parts separately as in OpenACC
- Many function calls in inner loops
  - Manually inline function calls as in OpenACC

```
#pragma acc parallel num_gangs(d3) vector_length(128) \
        present(gty1_real,gty1_imag,gty2_real,gty2_imag,\
                u1_real,u1_imag,u_real,u_imag
#pragma omp target map ( alloc: u1_real, u1_imag, u_real, u_imag, \
                gty1_real, gty1_imag, gty2_real, gty2_imag)
{
#pragma acc loop gang independent
#pragma omp teams distribute collapse(2)
    for (k = 0; k < d3; k++) {}
#pragma acc loop vector independent
       for(l = 1; l <= logd1; l += 2){
#pragma omp parallel for collapse(2) private(i11, i12, i21, i22, uu1_real, uu1_imag,
x11_real, x11_imag, x21_real, x21_imag, temp_real, temp_imag)
          for (i1 = 0; i1 <= li - 1; i1++) {
            for (k1 = 0; k1 <= lk - 1; k1++) {
                …
                gty2_real[k][i21+k1][j] = x11_real + x21_real;
                …
                temp_real = x11_real - x21_real;
                gty2_real[k][i22+k1][j] = (uu1_real)*(temp_real) -
                                           (uu1_imag)*(temp_imag);
```

MG Implementation

Multi-Grid Solvers on a sequence of meshes

- Long and short distance data transfers between grids; memory bandwidth intensive
- Compiler Challenges:
  - 3D data structures required manual linearization

```
#define I3D(array,n1,n2,i3,i2,i1) (array[(i3)*n2*n1 + (i2)*n1 + (i1)])

r1 = (double*)acc_malloc(n3*n2*n1*sizeof(double))
r1 = (double*)omp_target_alloc(n3*n2*n1*sizeof(double), omp_get_default_device());
…
#pragma acc data deviceptr(u1,u2), present(ou[0:n3*n2*n1]),
        present(ov[0:n3*n2*n1], or[0:n3*n2*n1])nt n3)
#pragma acc parallel num_gangs(n3-2) num_workers(8) vector_length(128)
#pragma omp target map(tofrom: ou[0:n3*n2*n1]) map(tofrom: ov[0:n3*n2*n1])
map(tofrom: or[0:n3*n2*n1]) is_device_ptr(u1, u2)
#pragma acc loop gang independent
#pragma omp teams distribute
 for (i3 = 1; i3 < n3-1; i3++) {
#pragma acc loop worker independent
#pragma omp parallel for collapse(2)
    for (i2 = 1; i2 < n2-1; i2++) {
#pragma acc loop vector independent
      for (i1 = 0; i1 < n1; i1++) {
        I3D(u1, n1, n2, i3, i2, i1) = I3D(ou, n1, n2, i3, i2-1, i1)
                                    + I3D(ou, n1, n2, i3, i2+1, i1)
                                    + I3D(ou, n1, n2, i3-1, i2, i1)
                                    + I3D(ou, n1, n2, i3+1, i2, i1); }}}
```

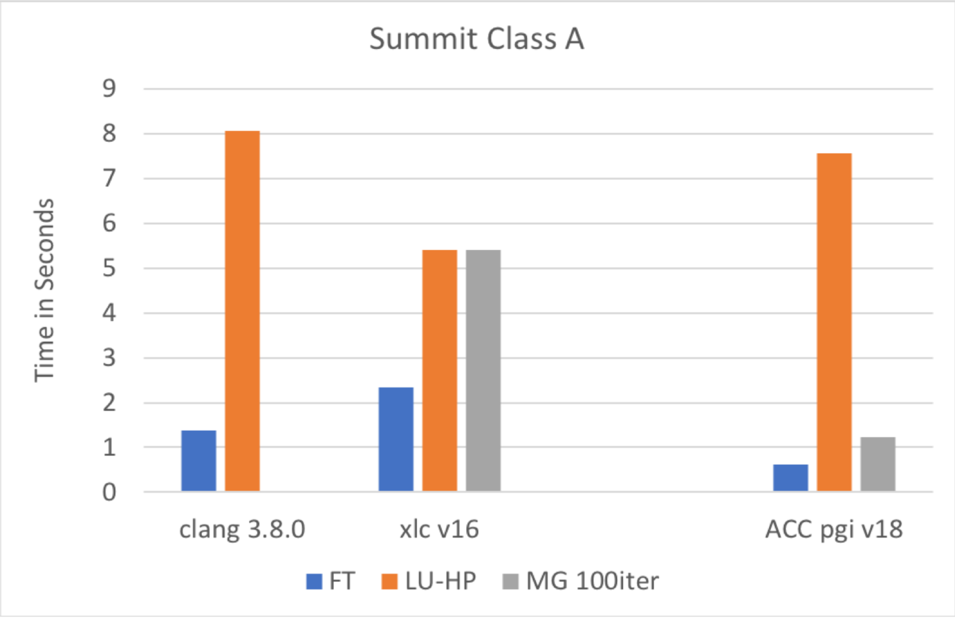
Evaluation Environment

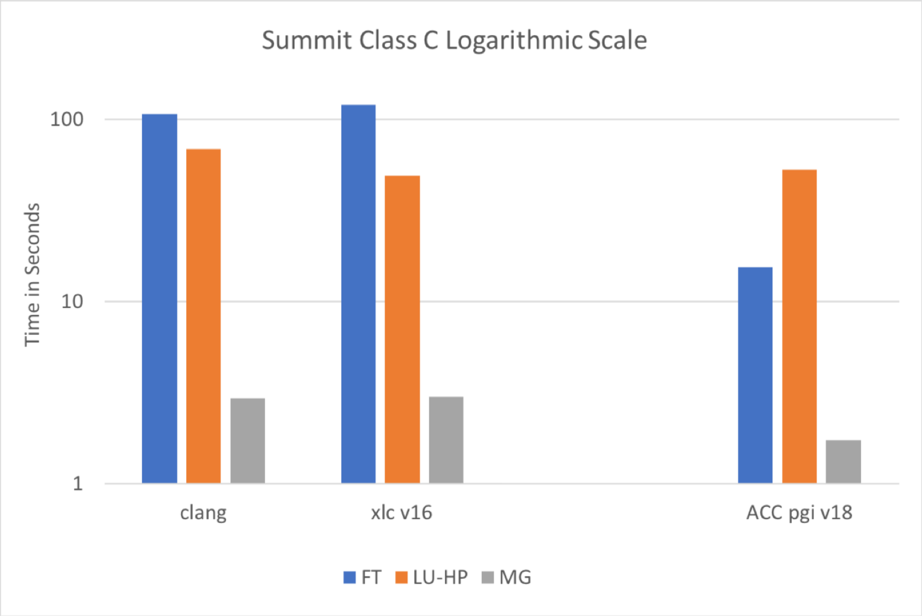
	Titan	Summit	Summitdev
System	Cray XK7	IBM AC922	IBM S822LC
Nodes	6274	9216	54
CPU	16 cores AMD Opteron 6274	22 Cores IBM POWER9	20 cores IBM POWER8
Accelerators	1 NVIDIA K20X	6 NVIDIA V100	4 NVIDIA P100

	Titan	Summit	Summitdev
GCC	-	-	7.1.1
PGI	18.5	18.3	18.4
CCE	8.7.3	-	-
CLANG/LLVM	-	CORAL 3.8.0	CORAL 3.8.0
XLC	-	16.1.0	13.1.0

- What do we compare?
  - Selection of systems with overlapping compilers….on the same system….??? We tried to do a non-empty intersection set for each test

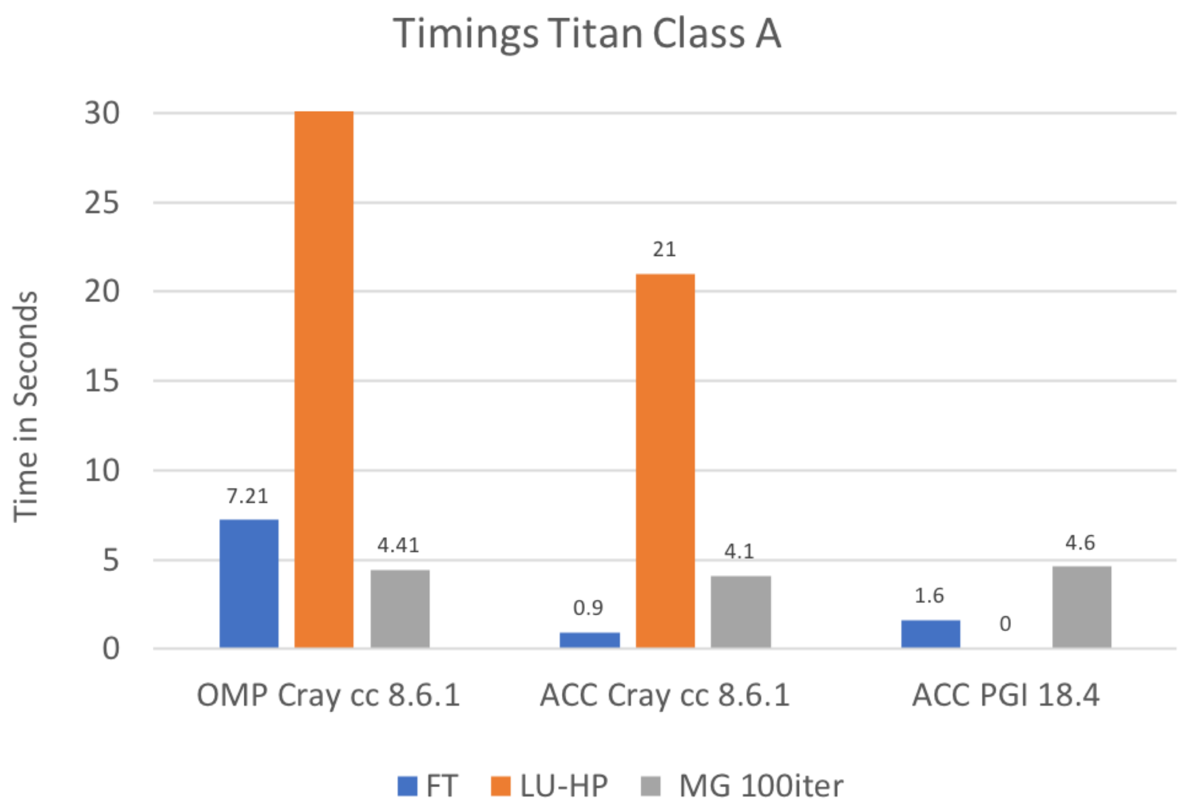
Comparing Compilers on Summit





- PGI OpenACC outperforms xlc OpenMP 4.5 for FT and MG, xlc holds up with PGI for LU-HP
- OpenACC “acc kernels” vs “acc loops”
  - The performance of “*acc kernels*” and “*acc loop*” was the same for all benchmarks
- The performance differences due to the quality of the compiler not a lack of functionality in OpenMP 4.5

Comparing OpenACC to OpenMP on Titan



- For LU-HP and FT OpenACC significantly outperforms OpenMP 4.5
- Only for MG can OpenMP 4.5 keep up with OpenACC
- Performance differences due to compiler support, not to a lack of functionality in OpenMP 4.5

Comparing OpenMP 4.5 vs OpenACC Performance FT on Titan

OpenMP 4.5 + Cray cc

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	24.12%	2.20986s	6	368.31ms	368.29ms	368.36ms	cffts1_neg
	7.94%	727.67ms	58	12.546ms	1.3440us	131.51ms	[CUDA memcpy DtoH]
	3.40%	311.86ms	56	5.5689ms	928ns	41.782ms	[CUDA memcpy HtD

OpenACC + Cray cc

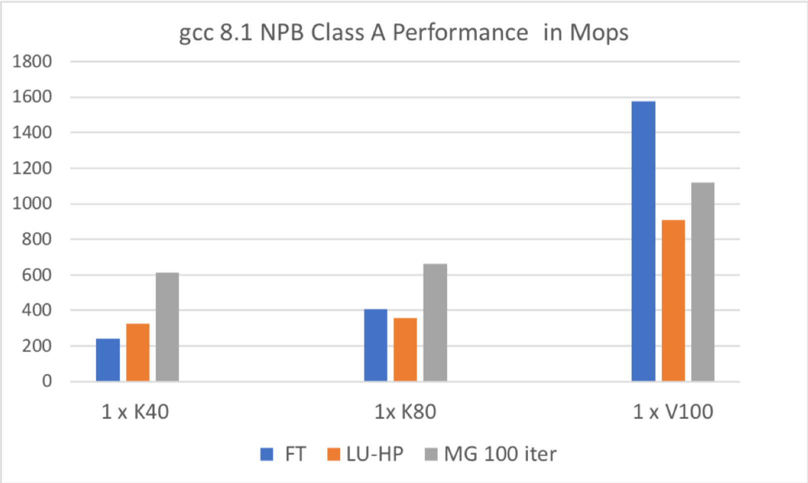
Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	32.83%	258.24ms	6	43.040ms	42.819ms	43.168ms	cffts1_neg
	13.09%	102.99ms	6	17.165ms	928ns	25.769ms	[CUDA memcpy HtoD]
	0.00%	9.9200us	6	1.6530us	1.4720us	1.9200us	[CUDA memcpy DtoH]

- Observations:
  - Kernel execution and data transfer to device is greatly reduced for OpenACC
  - Leaving all data on the device between kernel executions led to excessively high execution time in for Cray cc + OpenMP 4.5
  - We introduced manual data transfer, which lowered overall execution time, but potentially increased the time for data transfers; it can probably be optimized further
  - Once again we did no identify a lack of OpenMP 4.5 functionality impacting the perforance

Summary

- We described our experiences porting 3 NPB benchmarks to OpenMP 4.5 w/offloading
- We tested our implementations on 3 different systems at OLCF
- We compared compilers and programming models
- Conclusions:
  - For our study, OpenMP 4.5 target offload **did not lack a feature/functionality** when compared with OpenACC
  - OpenMP 4.5 employs existing functionality for accelerator execution, if possible, e. g. “parallel for”, and “simd”
  - Compiler support for OpenMP would definitely benefit from further improvement

- Bright spot on the horizon: gcc 8.1+
  - OpenMP offload support is getting increasingly stable 😎



References

1. Xu, Rengan & Tian, Xiaonan & Chandrasekaran, Sunita & Yan, Yonghong & Chapman, Barbara. (2014).” OpenACC Parallelization and Optimization of NAS Parallel Benchmarks”. 10.13140/RG.2.2.23914.41921
2. https://www.openmp.org/specifications/ OpenMP 4.5 Specification
3. R.v.d. Pas et. al., Using OpenMP – The next Step”, MIT Press, Oct. 2017, ISBN: 9780262534789
4. S. Chandrasekaran and G. Juckeland, “OpenACC for Programmmers”, Addison-Wesley Professional, September 20, 2017; ISBN-13: 978-0134694283
5. https://www.olcf.ornl.gov/ Oak Ridge Computing Leadership Facility
6. Jose Monsalve Diaz et. al, “Is OpenMP 4.5 Target Off-load Ready for the Real World ?”, https://openmpcon.org/wp-content/uploads/2018_Session1_Diaz.pdf

/

Backup

Comparing OpenMP 4.5 vs OpenACC Performance FT on Titan

OpenMP 4.5 + Cray cc

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	24.12%	2.20986s	6	368.31ms	368.29ms	368.36ms	cffts1_neg
	7.94%	727.67ms	58	12.546ms	1.3440us	131.51ms	[CUDA memcpy DtoH]
	3.40%	311.86ms	56	5.5689ms	928ns	41.782ms	[CUDA memcpy HtD

OpenACC + Cray cc

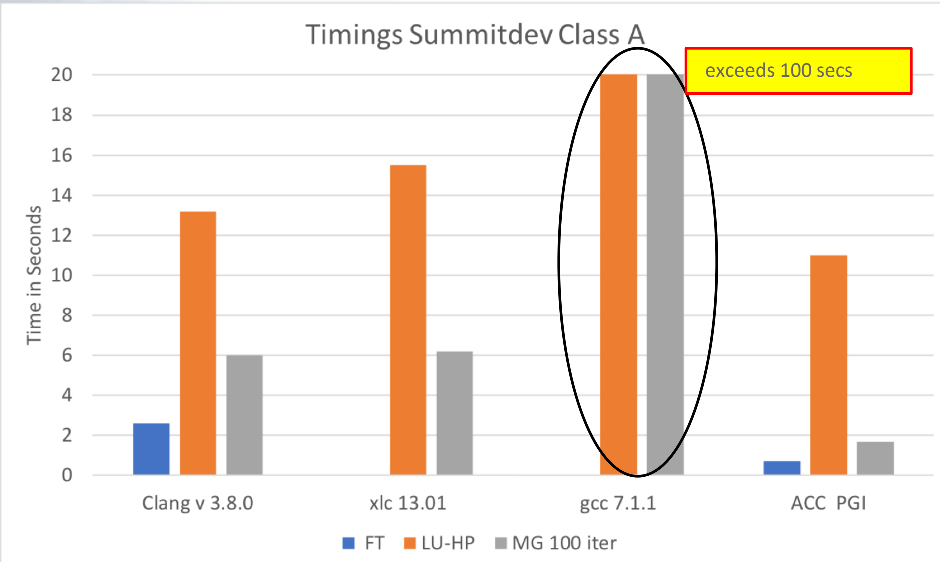
Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	32.83%	258.24ms	6	43.040ms	42.819ms	43.168ms	cffts1_neg
	13.09%	102.99ms	6	17.165ms	928ns	25.769ms	[CUDA memcpy HtoD]
	0.00%	9.9200us	6	1.6530us	1.4720us	1.9200us	[CUDA memcpy DtoH]

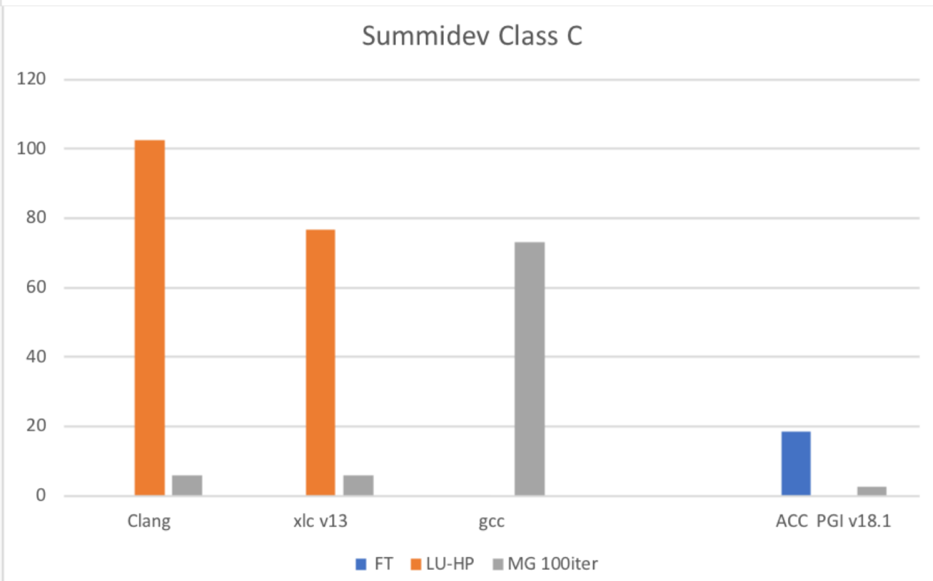
OpenACC + Cray cc no explicit manual data management

GPU activities:

25.38%	41.3997s	1	41.3997s	41.3997s	41.3997	ft$_$CFE_main_clone_
25.38%	41.3993s	1	41.3993s	41.3993s	41.3993s	fft$_$CFE__main_clone_
22.24%	36.2780s	1	36.2780s	36.2780s	36.2780s	fft$_$CFE_id_main_clone_
1.23%	2.00821s	6	334.70ms	334.56ms	334.76ms	cffts1_neg$_$CFE
0.46%	744.48ms	58	12.836ms	1.3120us	134.77ms	[CUDA memcpy DtoH]

Comparing Compilers on Summitdev





- Observations:
  - Runtimes of XL and Clang is quite similar
  - xlc v13 failed verification for FT
  - GCC 7.1.1 low performance
  - PGI-OpenACC 18.1 shows relatively better performance
    - PGI supports OpenMP 4.5 in their LLVM compiler, but there is no offload support yet
  - Class C FT:
    - Most runs fail due to memory errors
  - Results have to be taken with a grain of salt, as Summitdev was just set up for preparing for Summit