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)

September 25, 2018

Talk at OpenMPCon, 2018

Outline

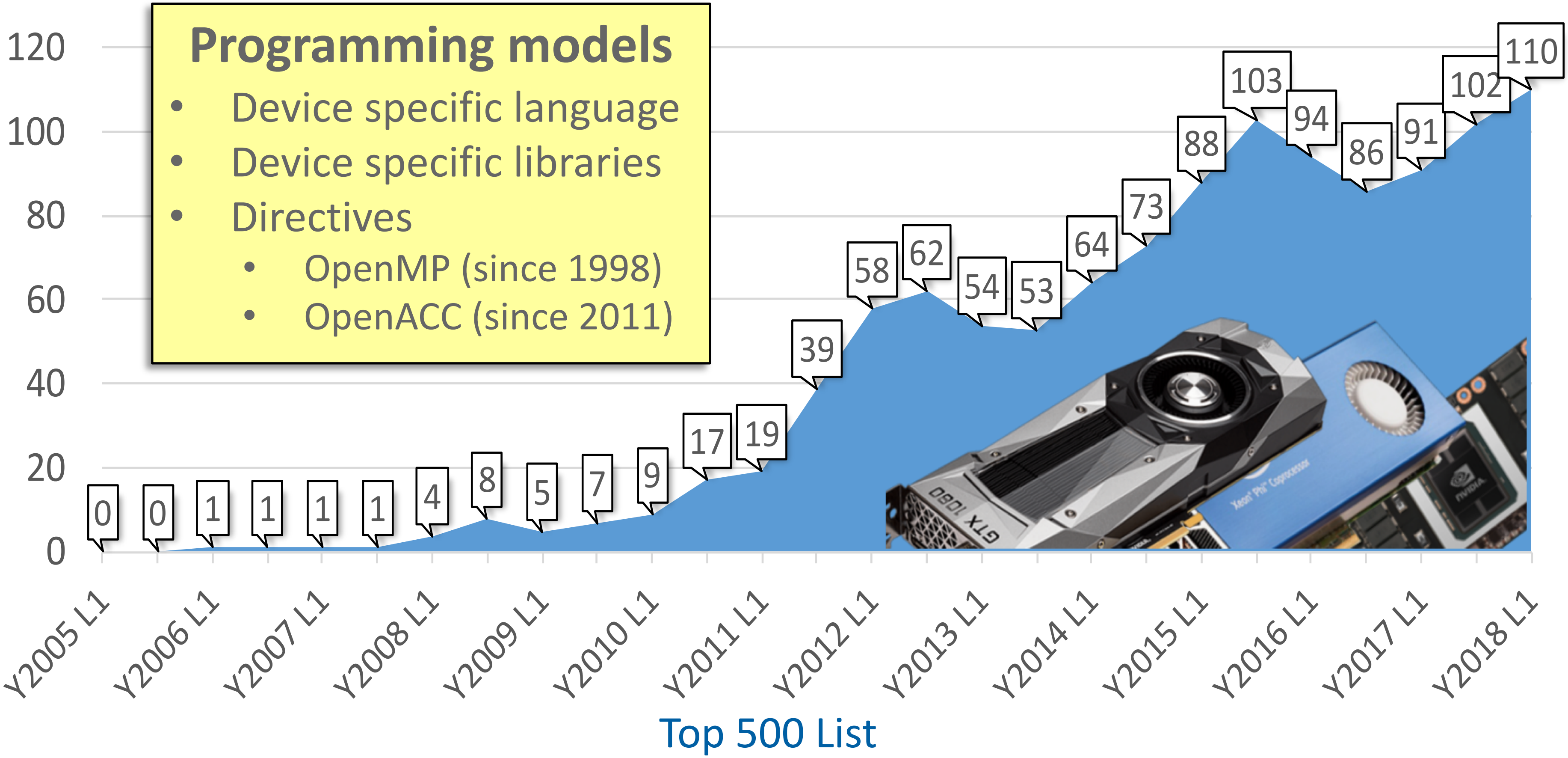
- Introduction
  - The OpenMP Target Concept
- Benchmark implementations
  - Benchmark descriptions
  - Porting OpenACC to OpenMP 4.5
- Performance Analysis:
  - Comparing compilers
  - Comparing hardware
  - Comparing OpenMP 4.5 to OpenACC
- Summary and Conclusions
- Discussion

Introduction

Number of systems with Accelerator devices in the Top500 list

Programming models

- Device specific language
- Device specific libraries
- Directives
  - OpenMP (since 1998)
  - OpenACC (since 2011)



OpenMP and device offloading

#pragma omp…

OpenMP 4.0

target [data]
declare target
target update
simd
declare simd
loop simd
parallel loop simd
teams

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

OpenMP 4.5

taskloop
taskloop simd
target enter data
target exit data
target simd
… Other API Calls …

OpenMP 5.0

allocate
declare mapper
Memory spaces
parallel loop
teams loop
… Other API Calls …

Important OpenMP Constructs and Clauses

#pragma omp target or **$!omp target**

- Create data environment and execute code region on the device

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

- Map a variable to/from the device data environment

#pragma teams

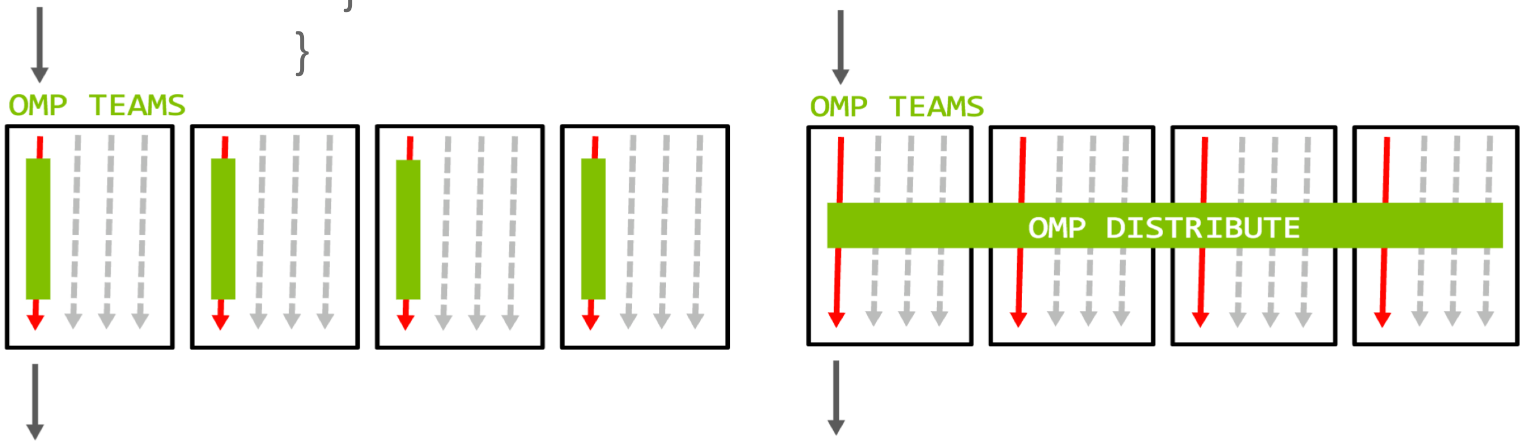
- Start kernel on the GPUs

#pragma teams distribute, parallel for, simd

- Distribute the work across the teams and threads within each team

Laplace Kernel Example

```
#pragma omp target teams distribute
  for( int j = 1; j < n-1; j++) {
  #pragma parallel for reduction(max:error)
    for( int i = 1; i < m-1; i++ ) {
      Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] +
                            A[j-1][i] + A[j+1][i]);
      error = fmax( error, fabs(Anew[j][i] - A[j][i]));
    }
  }
```



For more details check out the presentation by Jeff Larkin, Nvidia:
http://on-demand.gputechconf.com/gtc/2016/presentation/s6510-jeff-larkin-targeting-gpus-openmp.pdf

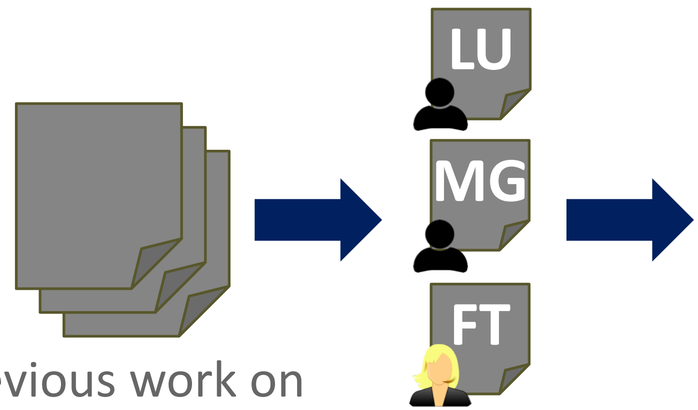
Image courtesy of Nvidia

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
- **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
- **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
- NPB benchmark offers different classes (Problem size) – S thru E

Development methodology

Initial steps



Previous work on NPB OpenACC Benchmark

Workload distribution among developers

OpenMP offloading development cycle

- Select offloading sections
- Detect possible pitfalls
- Identify data movements
- Create target regions
  No parallelism
- Check correctness
- Obtain base performance
- Apply constructs for parallelism
- Obtain performance metrics
- Find optimization opportunities

General Implementation Strategy: Translating OpenACC to OpenMP

- 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

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)\
                   map(from: 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);
```

LU-HP Implementation

Lower-Upper Gauss Seidel Solver using a hyperplane method

- Compiler Challenges:
  - Array privatization
  - Change data layout to enable memory coalescing
  - Manual loop unrolling

```
#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 teams map (alloc: a, b, c, d, u, indxp, jndxp, rho_i, qs) \
        num_teams((npl+127)/128)
  {
#pragma omp distribute parallel for private( tmp1, tmp2, i, j, k)
for (n = 1; n <= npl; n++) {
      j = jndxp[l][n];
      i = indxp[l][n];
      k = l - i - j;
      tmp1 = rho_i[k][j][i];
      tmp2 = tmp1 * tmp1;
      d[0][0][n] =  1.0 + dt * 2.0 * ( tx1 * dx1 + ty1 * dy1 + tz1 * dz1 );
      d[0][1][n] = -dt * 2.0
          * ( tx1 * r43 + ty1 + tz1 ) * c34 * tmp2 * u[1][k][j][i];
      d[1][1][n] =  1.0
          + dt * 2.0 * c34 * tmp1 * ( tx1 * r43 + ty1 + tz1 )
          + dt * 2.0 * ( tx1 * dx2 + ty1 * dy2 + tz1 * dz2 );
```

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	4 NVIDIA P100	6 NVIDIA V100

Challenge for our study:

- Different set of compilers available on different platforms
  - Each behaving differently (correctness is not always portable)
- What do we compare?
  - Selection of systems with overlapping compilers
    - Support for OpenMP Offloading in HPC system is still low despite compilers support
    - Clang trunk bug with *math.h* and host specific asm code

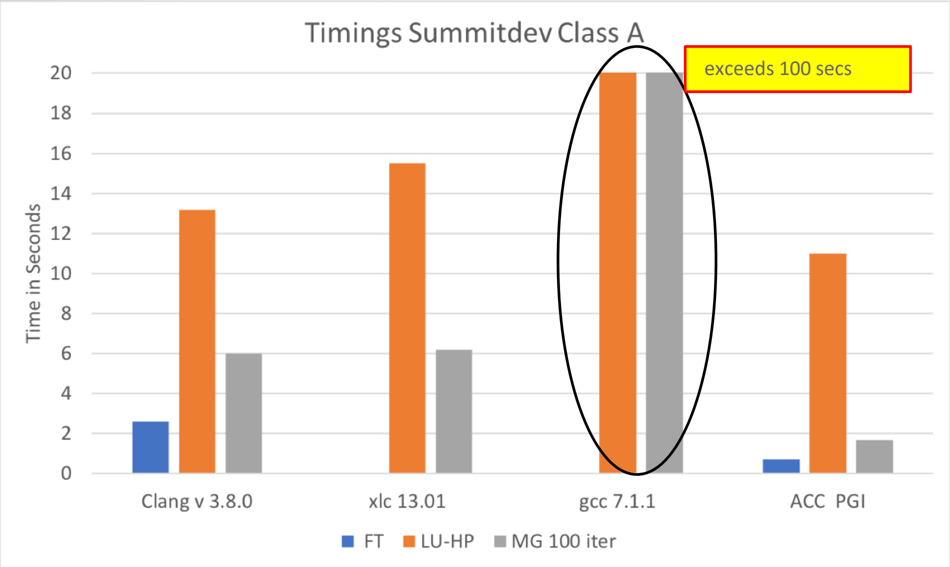
Compilers

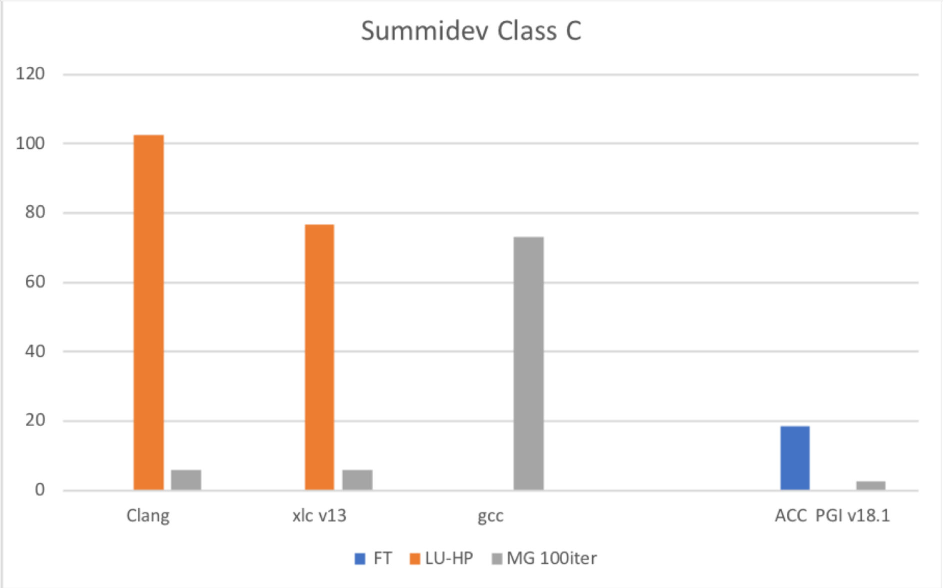
	Titan	Summit	Summitdev
GCC	-	-	7.1.1
PGI	18.5	18.3	18.4
CCE	8.7.3	-	-
CLANG/LLVM	-	CORAL 3.8.0	CORAL 3.8.0
XLC	-	16.1.0	13.1.0

Challenge for our study:

- Different set of compilers available on different platforms
  - Each behaving differently (correctness is not always portable)
- What do we compare?
  - Selection of systems with overlapping compilers
    - Support for OpenMP Offloading in HPC system is still low despite compilers support
    - Clang trunk bug with *math.h* and host specific asm code

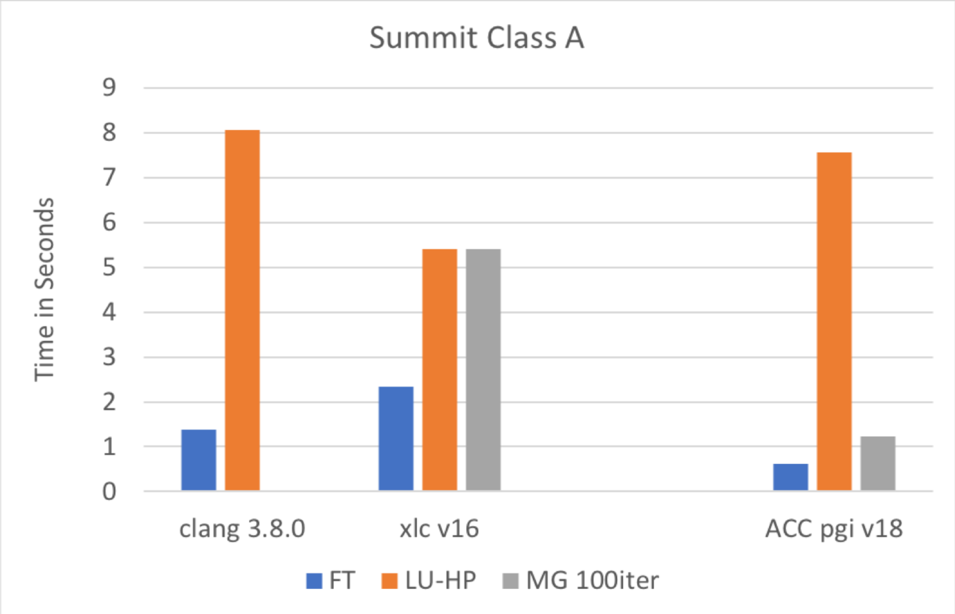
Run time on Summitdev

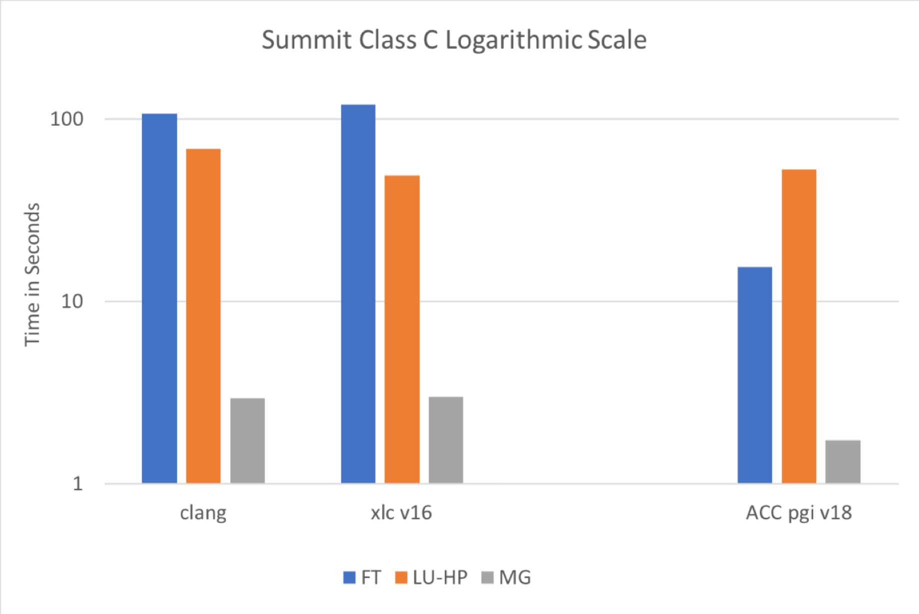




- Observations:
  - Runtimes of XL and Clang is quite similar
  - xlc V13 failed verification for FT
  - GCC 7.1.1 low performance
  - PGI-OpenACC 18.1 shows relatively better performance
  - PGI supports OpenMP 4.5 in their LLVM compiler, but there is no offload support yet
  - Class C FT and LU-HP fail due to memory

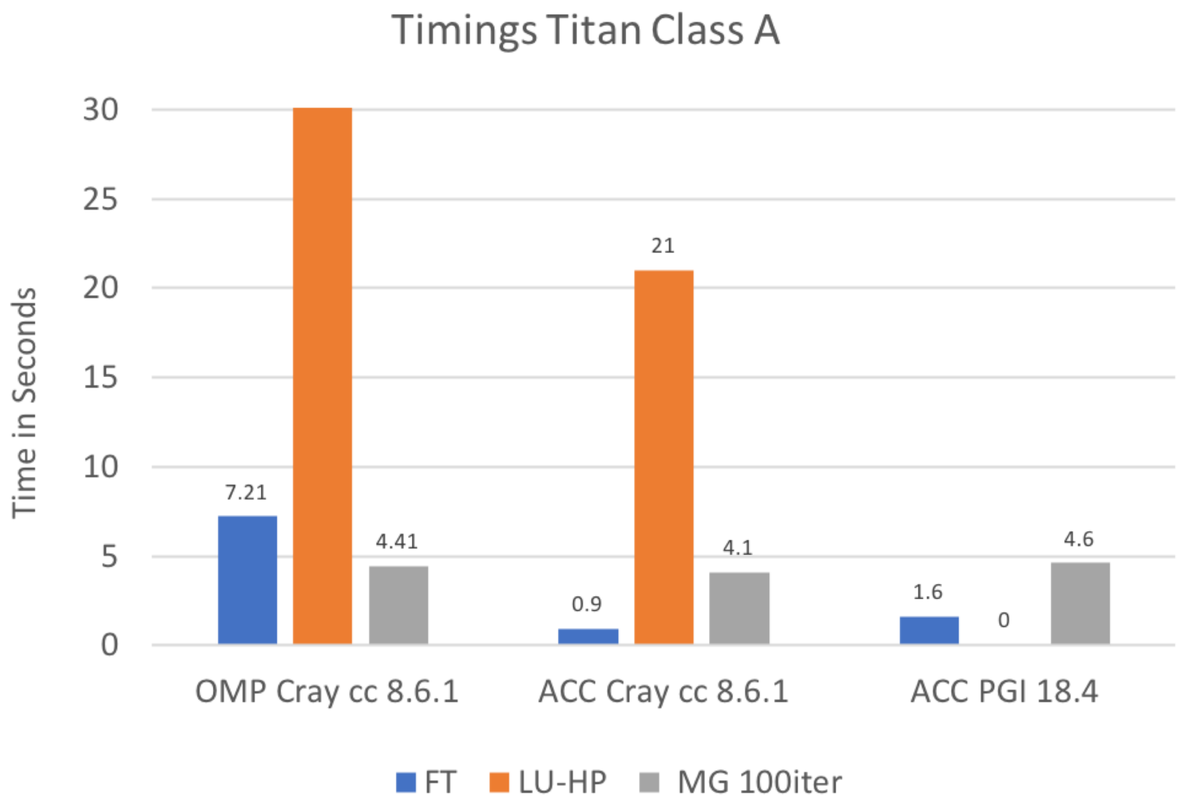
Run time on Summit





- Summit is currently unavailable due to acceptance effort …
- PGI does not support OMP Offloading yet
- Original OpenACC employs *"pragma acc kernels"* which is not available in OpenMP 4.5

Run time on Titan



- Observations:
  - For LU-HP and FT OpenACC significantly outperforms OpenMP 4.5
  - Only for MG OpenMP 4.5 can keep up with OpenACC

Comparing Compilers: MG Class A xlc and gcc on Summitdev

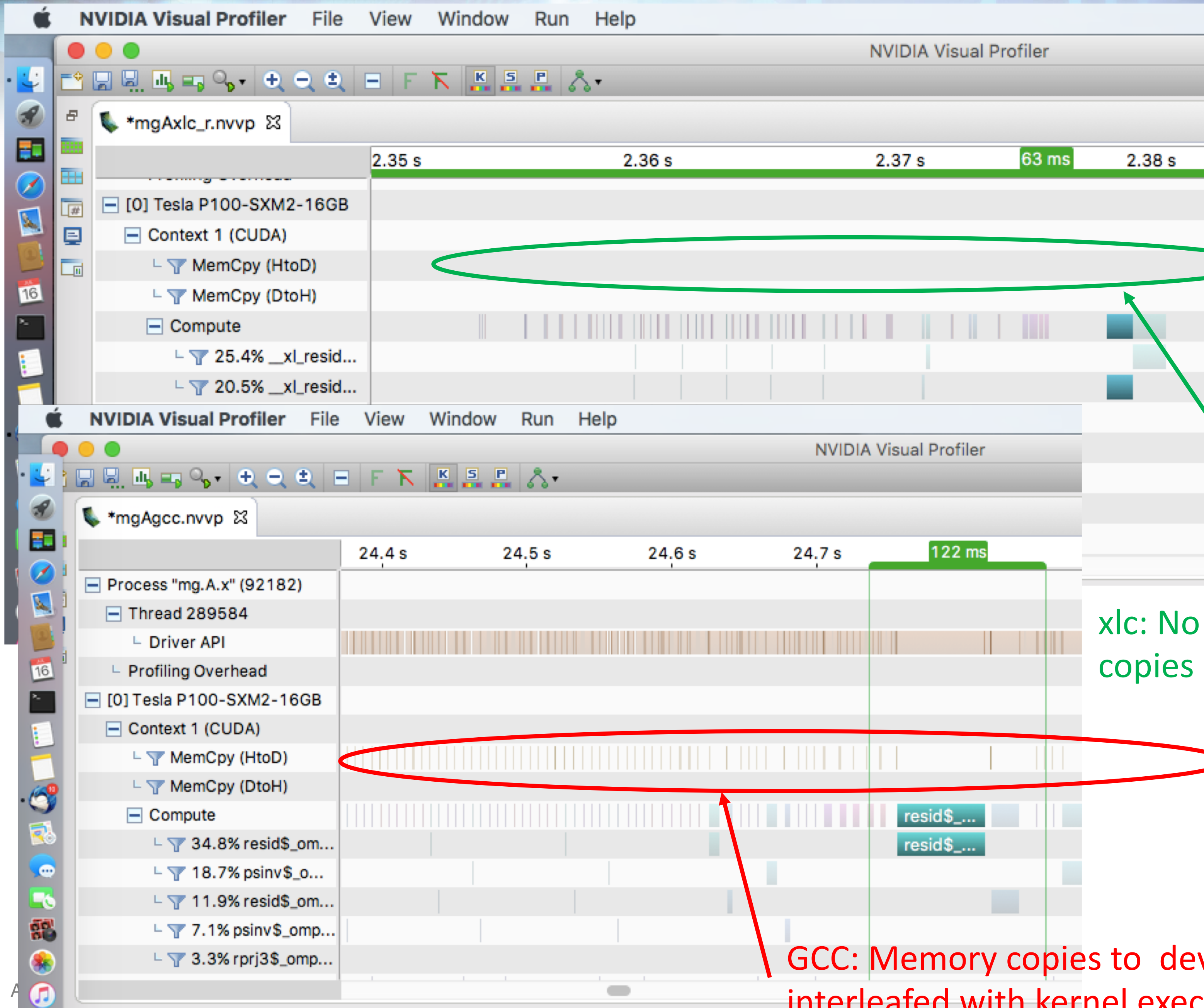
xlc v 13

```
==67718== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:
                   25.34%  292.94ms       810  361.65us  4.4800us  1.3336ms  __xl_resid_l679_OL_4
                   20.51%  237.16ms       810  292.79us  5.2800us  1.1059ms  __xl_resid_l672_OL_3
                   14.04%  162.31ms       808  200.88us  3.1680us  1.3819ms  __xl_psinv_l551_OL_2
                   11.04%  127.61ms       808  157.93us  3.9040us  1.1059ms  __xl_psinv_l550_OL_1 1.26%
                    0.00%  12.128us        14     866ns     704ns  1.6000us  [CUDA memcpy HtoD]
```

gcc 7.1

```
==45872== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:
                   34.59%  13.3874s       810  16.528ms  532.87us  62.394ms  resid$_omp_fn$44
                   18.63%  7.21199s       808  8.9257ms  158.34us  60.259ms  psinv$_omp_fn$40
                   11.81%  4.56960s       810  5.6415ms  384.61us  20.359ms  resid$_omp_fn$46
                    7.07%  2.73749s       808  3.3880ms  129.28us  20.859ms  psinv$_omp_fn$42
                    3.30%  1.27857s       707  1.8084ms  123.84us  8.7265ms  rprj3$_omp_fn$16
                    0.33%  129.02ms    115738  1.1140us     960ns  2.5920us  [CUDA memcpy HtoD]
```

- Observations:
  - gcc shows a larger number of host-to-device data transfer
  - xlc uses just a small number of asynchronous data transfers (cuMemcpyHtoDAsync)
  - gcc and xlc employ different grid size, block size and number of registers per thread (see following 2 slides)



xlc: No memory copies

GCC: Memory copies to device interleafed with kernel execution

Comparing OpenMP 4.5 vs OpenACC Performance FT on Titan

OpenMP 4.5 + Cray cc

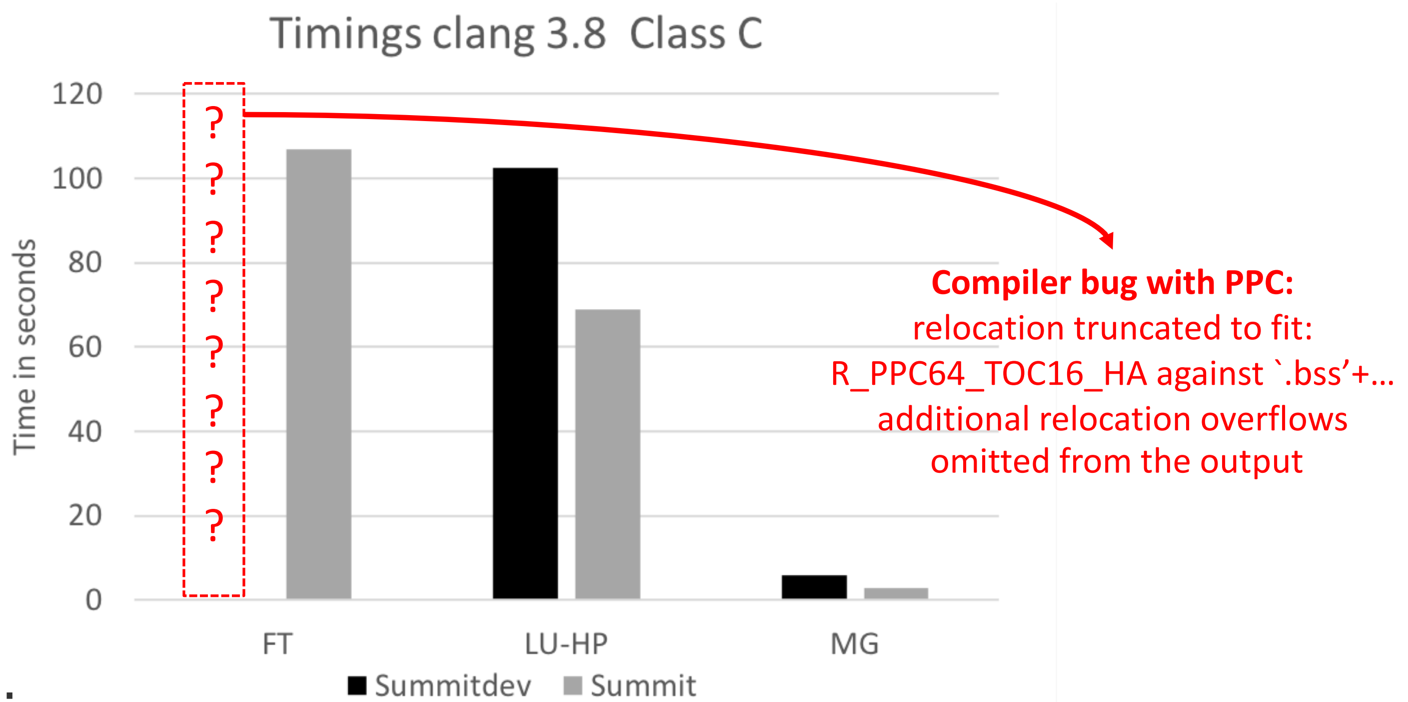
Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	24.12%	2.20986s	6	368.31ms	368.29ms	368.36ms	cffts1_neg
	7.94%	727.67ms	58	12.546ms	1.3440us	131.51ms	[CUDA memcpy DtoH]
	3.40%	311.86ms	56	5.5689ms	928ns	41.782ms	[CUDA memcpy HtD
	7.91%	45.793ms	6	7.6321ms	7.5692ms	7.7115ms	cffts1_neg
	5.36%	30.988ms	6	5.1646ms	5.0361ms	5.3973ms	cffts1_neg

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:
  - For each loop there are 3 kernels for OpenMP 4.5 vs 1 kernel for OpenACC
  - Data transfer to device is greatly reduced for OpenACC
  - Data transfer to host is very high in OpenMP 4.5:
    - We had to move some arrays back to host to ensure correct execution, not necessary for OpenACC

Comparing Hardware: Performance Summitdev vs Summit using clang 3.8



- Observations:
  - We use the same compiler version on both platforms
  - We compare impact of using 1 Nvidia P100 GPU (Summitdev) vs 1 Nvidia V100 GPU(Summit)

Invitation to my next talk

IWOMP Thursday 2:30pm session

OpenMP 4.5 Validation and verification Suite for Device offload

Jose Monsalve Diaz **Swaroop Pophale**

Oscar Hernandez David E. Bernholdt Sunita Chandrasekaran

UNIVERSITY OF DELAWARE.

ECP EXASCALE COMPUTING PROJECT

OAK RIDGE National Laboratory

Summary

- We described our experiences porting 3 NPB benchmarks to OpenMP 4.5 w/ offloading
- We tested our implementations on 3 different systems at OCLF
- We compared compilers, programming models and hardware
- Conclusions:
  - Evaluating 3 NPB benchmarks showed that 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
- User community:
  - Would you find it useful to have a public domain, full NPB OpenMP target implementation available?

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

Images

https://pixabay.com/en/user-avatar-female-blond-girl-310807/