Automatic Testing of OpenACC Applications

Khalid Ahmad
School of Computing/University of Utah

Michael Wolfe
NVIDIA/PGI

November 13th, 2017

Why Test?

When optimizing or porting

Validate the optimization or the port

Identify where the computations start to diverge

Use Cases

General

- Validate a new machine (x86, ARM, OpenPower)
- Validate a different version of the compiler
- Validate a new compiler optimization
- Validate modifications / new algorithms

GPU / OpenACC

- Find where computations start to diverge
- Programmer error: missing data movement
- Hardware differences: different FMA, rounding, intrinsics, accumulation order
- Compiler bugs

Code Example

```
1)  void vectorSinGPU(double *A, double *C, uint32_t N)
2)  {
3)    // Ensure the data is available on the device
4)    #pragma acc data copyin(A[0:N]) copyout(C[0:N])
5)    {
6)     // Compute construct
7)     #pragma acc kernels loop independent present(A[0:N],C[0:N])
8)      for (int i = 0; i < N; i++) {
9)        C[i] = fsin(A[i]);
10)     }
11)   }
12) }
```

1) General Compare

- The user may specify several parameters using environment variables
- User passes pointer to data, datatype, size
- User creates golden data file with known correct settings / program
- User reruns program to compare with golden data file

General Compare Code Example

```
1)    void vectorSinGPU(double *A, double * C, uint32_t N){
2)        #pragma acc enter data copyin(A[0:N])
3)        #pragma acc enter data create(C[0:N])
4)        #pragma acc kernels loop present(A[0:N],C[0:N]) independent
5)          for (int i = 0; i < N; i++) {
6)              C[i] = sin(A[i]);
7)          }
8)        //Copy output data from the CUDA device to the host memory
9)         #pragma acc exit data copyout(C[0:N])
10)        #pragma acc exit data delete(A[0:N])
11)        pgi_compare(C,"double",N,__FILE__,__LINE__);
12)        pgi_compare(A,"double",N,__FILE__,__LINE__);
13) }
```

How to use the general compare

1) *export PGI_COMPARE=FILE=TRIAL,CREATE*

2) Run program with function calls

3) *export PGI_COMPARE=FILE=TRIAL,rel=5,COMPARE*

4) Rerun program with function calls

PGI_Compare Environment Variable

Option	Description
abs=r	Use 10^{-r} as an absolute tolerance
rel=r	Use 10^{-r} as a relative tolerance
report=n	Report first n differences
skip=n	Skip the first n differences
patch	Patch erroneous values with correct values
stop	Stop after report= differences
summary	Print a summary of the comparisons and differences found at program exit

OpenACC Background

- OpenACC runtime manages two copies of the data, host and device, and identified by the present table.

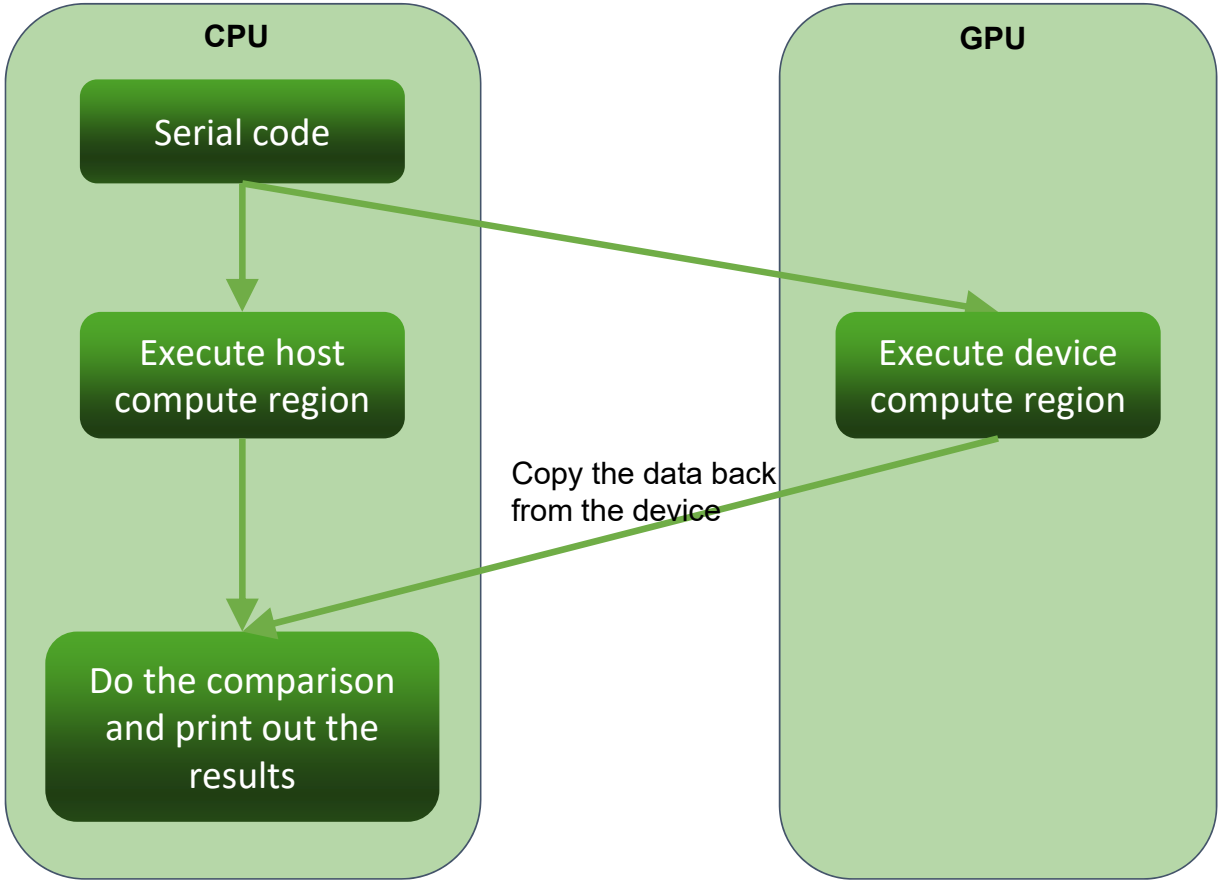
- Present table is indexed by the host address, contains device address, data size, data type

2) Host Device Compare*

- User passes pointer to host resident data and size of data
- Function locates the relevant device data pointer in the present table
- Using the present table we can also know the data type being used
- Then we perform a data type based comparison

* The autocompare will be exposed with a command line option, when it gets released in an upcoming PGI version sometime hopefully in early 2018

Auto-compare flow chart



Host Device Compare Code Example

```
1)    void vectorSinGPU(double *A, double * C, uint32_t N){
2)       #pragma acc enter data copyin(A[0:N])
3)       #pragma acc enter data create(C[0:N])
4)       #pragma acc kernels loop present(A[0:N],C[0:N]) independent
5)       for (int i = 0; i < N; i++) {
6)             C[i] = sin(A[i]);
7)       }
8)       acc_compare(C,N);
9)       //Copy output data from the CUDA device to the host memory
10)      #pragma acc exit data copyout(C[0:N])
11)      #pragma acc exit data delete(A[0:N])
12) }
```

3) Host Device Compare All*

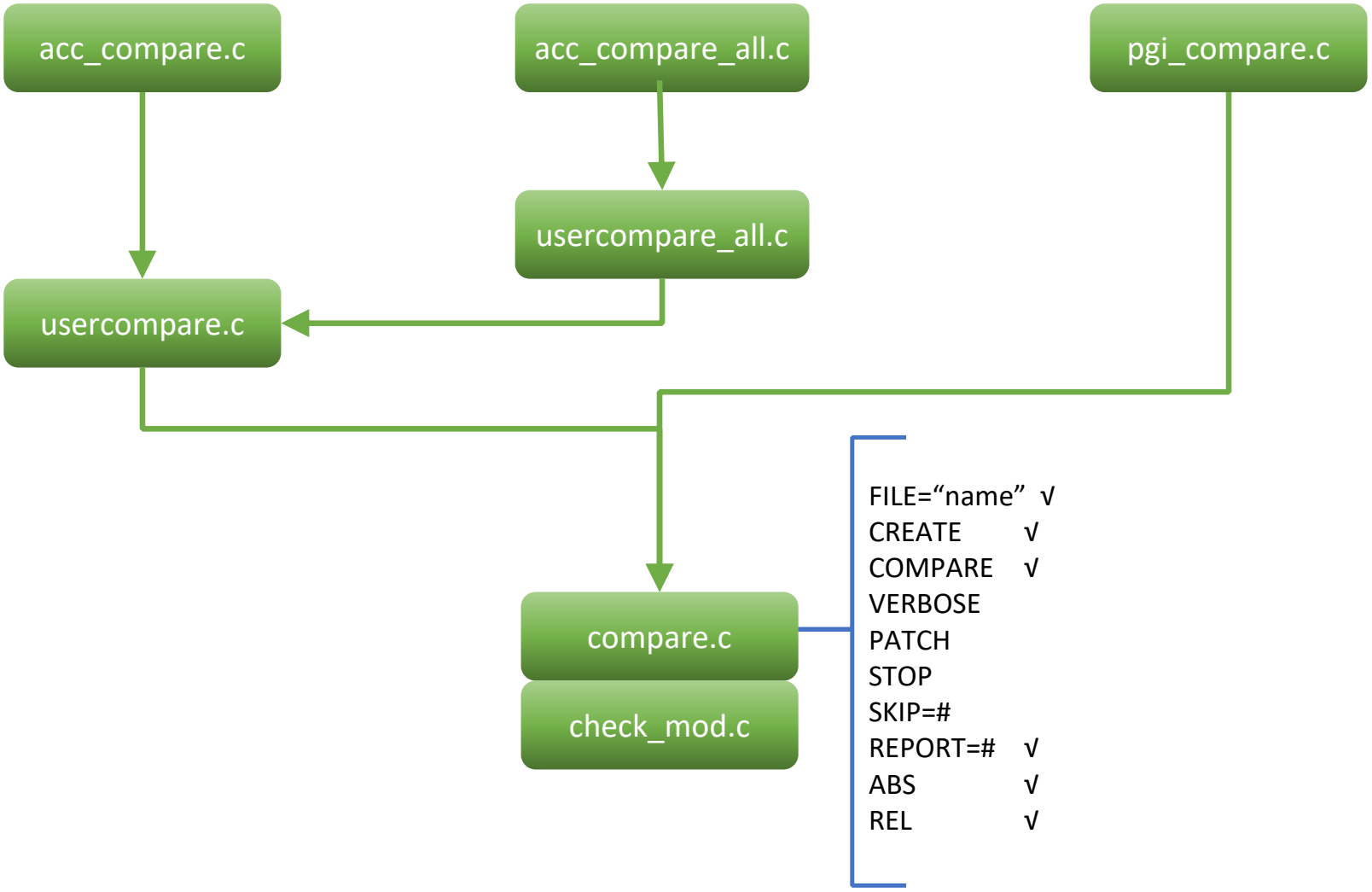
- No parameters to pass, data type is stored in the present table, so the compares are type-aware even though the user doesn't identify the data types
- The function traverses the present table
- And calls the compare function on each entry in the present table

* The autocompare will be exposed with a command line option, when it gets released in an upcoming PGI version sometime hopefully in early 2018

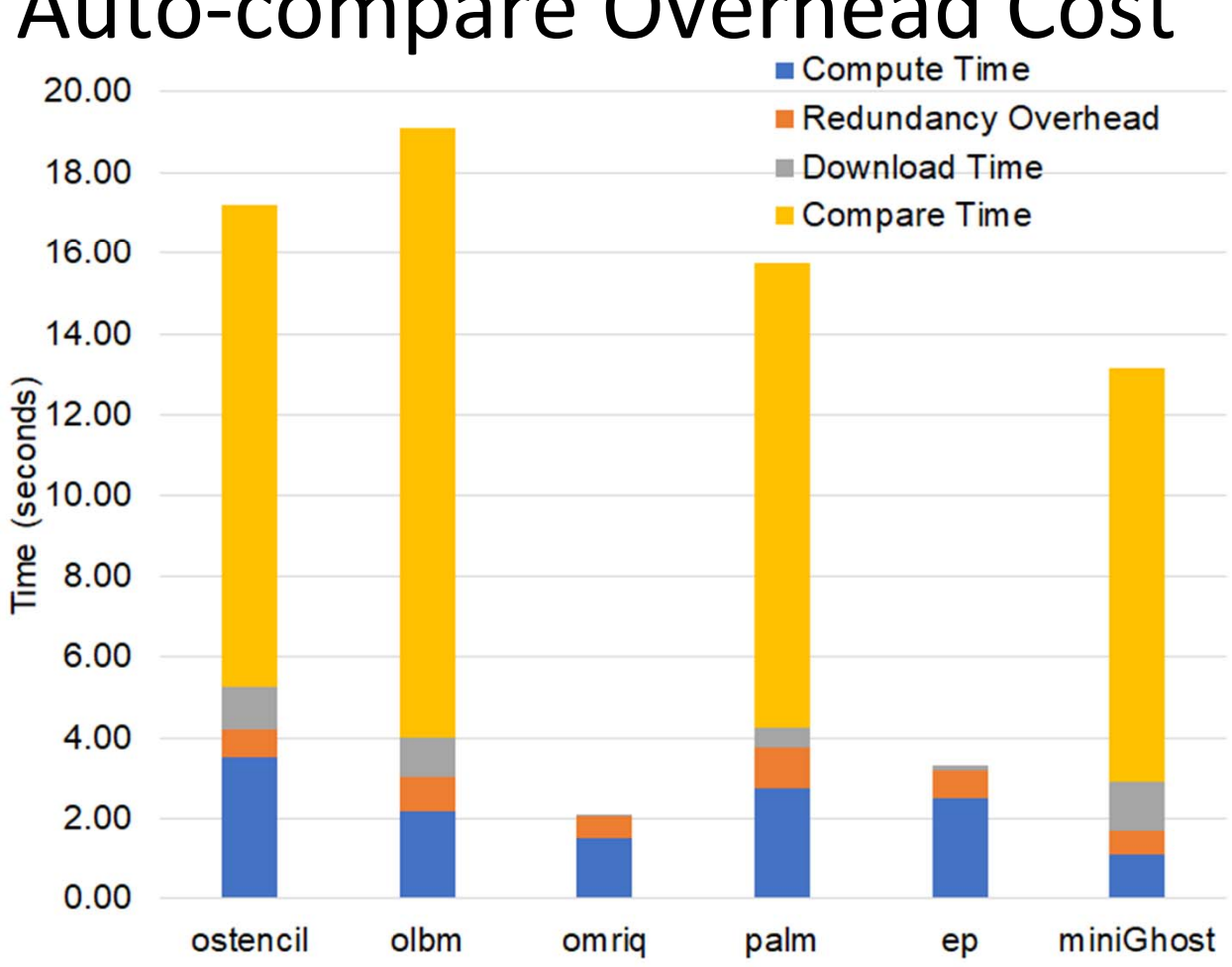
Host Device Compare All Code Example

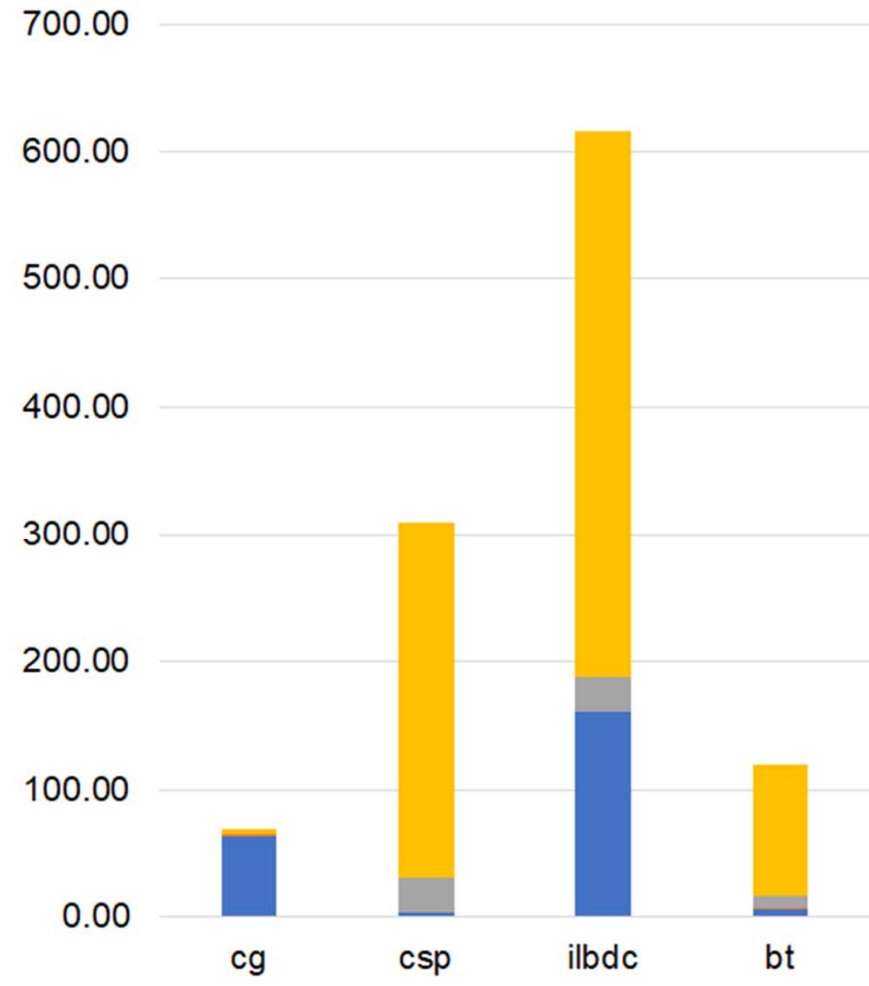
```
1)    void vectorSinGPU(double *A, double * C, uint32_t N){
2)       #pragma acc enter data copyin(A[0:N])
3)        #pragma acc enter data create(C[0:N])
4)        #pragma acc kernels loop present(A[0:N],C[0:N]) independent
5)        for (int i = 0; i < N; i++) {
6)              C[i] = sin(A[i]);
7)        }
8)         acc_compare_all();
9)       //Copy output data from the CUDA device to the host memory
10)       #pragma acc exit data copyout(C[0:N])
11)       #pragma acc exit data delete(A[0:N])
12)  }
```

Implementation



Auto-compare Overhead Cost





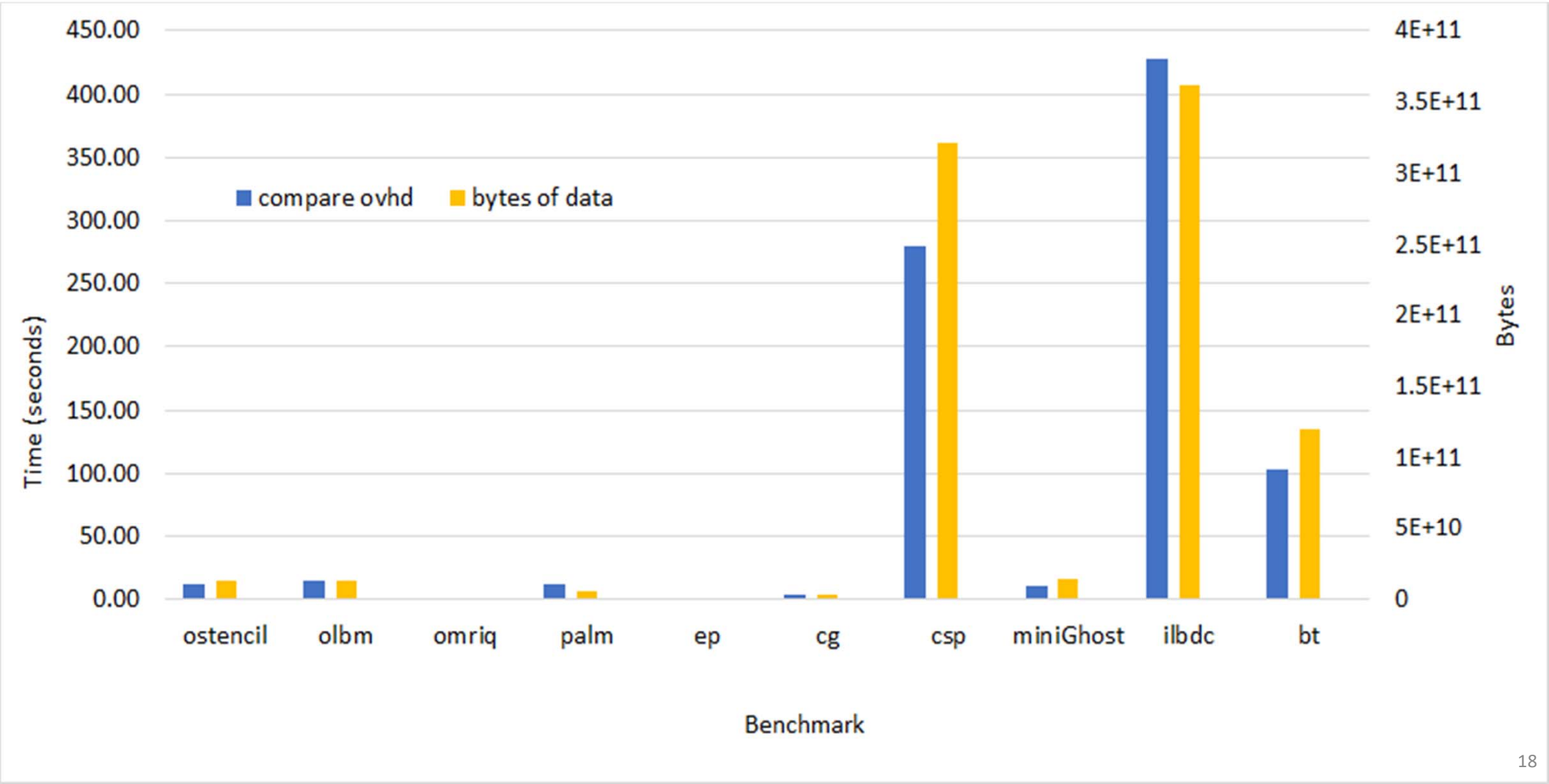
Benchmark

Single core Intel Haswell
Nvidia Pascal P100

Benchmark Statistics

Benchmark	Variables and arrays compared	Values compared	Variables and arrays with differences	Differences tolerated
ostencil	202	3,388,997,632	0	0
olbm	61	586,800,000	59	520,634,266
omriq	3	68,608	2	53,240
palm	31,244	1,532,482,935	14,784	374,679,922
ep	4	13	2	2
cg	186	621,600,195	168	4,858,272
csp	4,057	40,132,155,677	3,897	5,693,059
miniGhost	2,506	1,844,059,545	175	175
ilbdc	3,001	53,818,895,200	2,000	35,305,830,600
bt	5,036	15,041,440,200	4,798	38,931,891

Comparing Byte Count vs Compare Time



Related Work

1) OpenARC compiler framework
   - Similar to our auto-compare feature
   - User specifies the desired compute region to test
   - The rest of the program is run sequentially including other compute regions

1) Cray Comparative Debugger (CCDB) allows the programmer to
   - Launch two versions of a program
   - Add breakpoints
   - Does not support automatic testing

Future Work

- General
  - Implementing more options such as skip, patch, stop, bits ...
  - Implement a pragma version
  - Adding support for nested data structures and derived types
  - Optimize the speed of the comparison
    - Option that runs the comparisons in parallel
    - Reduce the number of values being compared
    - Compare only specific compute constructs to reduce the overall cost

- Auto compare
  - Running the host code in parallel
  - Running the compare on the GPU

Summary

- Tool that automatically detect numerical differences and help identify bugs
- Overhead of the redundant execution dominated by the slower execution unit
- Debuggers and correctness checkers always introduce some overhead, which is fine and in most cases still a lot faster than a manual investigation