OpenACC 2.5 Validation Testsuite targeting multiple architectures

Kyle Friedline, Sunita Chandrasekaran, University of Delaware, USA

Graham Lopez, Oscar Hernandez, Oak Ridge National Lab, USA

June 22, 2017

P3MA Workshop @ ISC

https://github.com/OpenACCUserGroup/OpenACCV-V

OpenACC- a directive-based parallel programming model

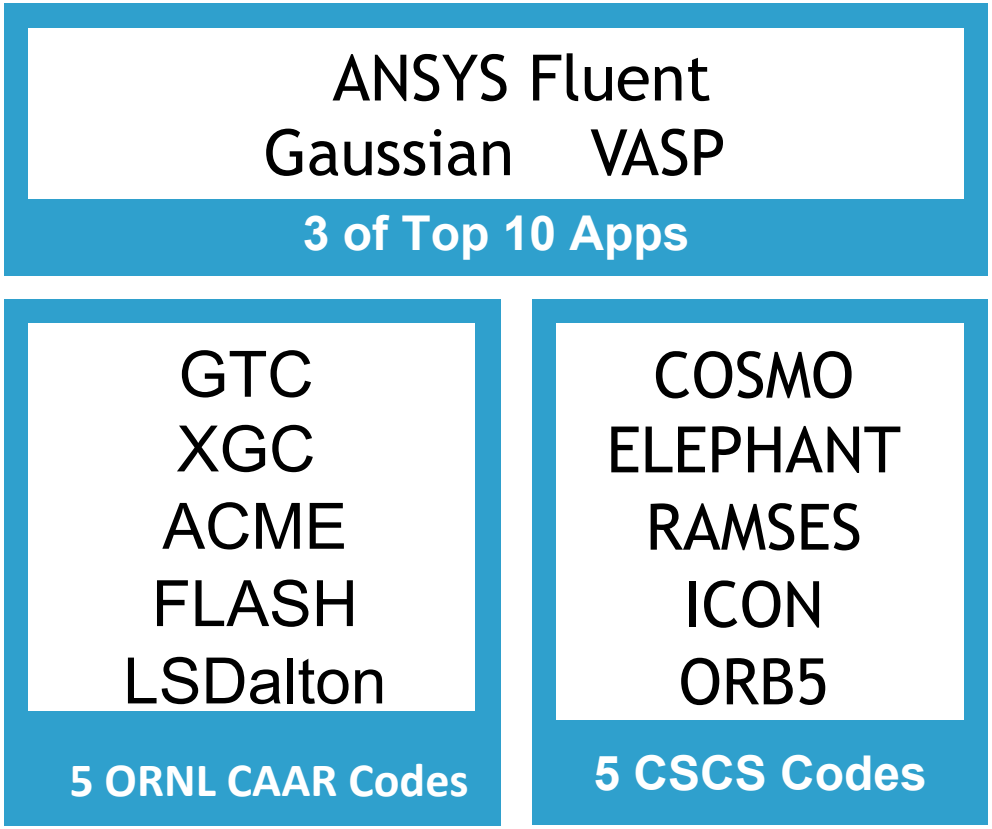
Add Simple Compiler Directive

```
main()
 {
  <serial code>
  #pragma acc kernels
  {
    <parallel code>
  }
}
```

Motivation

- To help developers find and fix compiler bugs
- To assist users to know where the compilers stand in terms of OpenACC functionality support
- To assist users with fundamental understanding of spec features
- To discern ambiguities in the OpenACC specifications
- To discuss misinterpretations of feature definitions

Top HPC Apps adopting OpenACC



OpenACC

- Execution:
  - The `parallel` directive for explicit parallelization mapping
  - The `kernels` directive for implicit parallelization mapping
  - The `loop` construct for parallelization specification
- Memory:
  - The `data` directive for scoped, explicit-lifetime data elements
  - The `enter/exit` data directives for executable data transfer

OpenACC Execution Model

- Parallel
  - Best when given explicit parallelization dimensions.
- Kernels
  - Best when compiler is allowed to implement its determined best parallelization dimensions
  - Potential for multiple compute kernels allowing multiple, varying-size compute-resource allocations.
- Loop
  - Describes parallelization across for-loops with the addition of `gang`, `worker`, or `vector` clauses.
  - Describes data dependencies both in code structure and in loop iterations through the `async`, independent, and `seq` clauses.

OpenACC Memory Model

- Enter/Exit data directives
  - Executes memory allocation, deallocation, and transfer
  - Allows for cross-scope data lifetimes
  - Requires more vigilance on the part of the programmer to manage data lifetimes, especially with dynamic execution patterns
- Data region
  - Gives data a specified beginning and ending on device
  - Allows compiler to predetermine data presence via scopes and call trees.
  - Can lead to data remaining allocated on device longer than necessary to complete scope
- Data management clauses
  - Private
  - Firstprivate
  - Reduction
  - Cache
  - Deviceptr
  - Present

Multi-Paradigm Accelerating Model

- Multiple Accelerator Support
  - acc_get_num_devices
  - acc_set_device_type
  - acc_set_device_num

Compilers supporting OpenACC

- PGI
  - NVIDIA GPU, X86 CPU, X86 Xeon Phi, POWER
- GNU
  - NVIDIA GPU
- Cray
  - NVIDIA GPU
- Sunway OpenACC*
- Omni (PEZY-SC) from University of Tsukaba
- OpenARC from ORNL, OpenUH from UH and SBU

https://www.openacc.org/tools

The Validation Test Suite

- The suite of 177 tests includes 86 Fortran tests and 91 C tests covering:
  - All of the parallel construct and its clauses, but missing some fringe functionality and clause combinations
  - Most of kernels construct and its clauses, but missing all fringe functionality and clause combinations
  - Data directives (data, enter data, exit data) with primary clauses tested, but missing all fringe functionality and most clause combinations
- Tests build upon each other; initial tests test core functionality, while later tests built upon previously tested functionality

Test Suite Structure

- Primary test types
  - Unit tests
  - Incremental Integration Tests
  - Use Cases
- Test Design Issues
  - Backwards-compatibility
  - Compute complexity
  - Platform-independent test compatibility

Test Design

```
double * a = (double *)malloc(n * sizeof(double));
double * b = (double *)malloc(n * sizeof(double));
double * a_copy = (double *)malloc(n *sizeof(double));
int * devtest = (int *)malloc(sizeof(int));
int err = 0;
devtest[0] = 1;
#pragma acc enter data copyin(devtest[0:1])
#pragma acc parallel present(devtest[0:1])
{
  devtest[0] = 0;
}
for (int x = 0; x < n; ++x){
  a[x] = rand();
  a_copy[x] = a[x];
}
if (devtest[0] == 1 && run_probabilistic == 1){
  #pragma acc enter data create(a[0:n])
  #pragma acc exit data copyout(a[0:n])
  err = 1; \\Failing
  for (int x = 0; x < n; ++x){
    if (fabs(a[x] - a_copy[x]) > PRECISION){
      err = 0;
    }
  }
}
for (int x = 0; x < n; ++x){
  a[x] = rand();
  b[x] = 0.0;
}
#pragma acc enter data copyin(a[0:n]) create(b[0:n])
#pragma acc parallel present(a[0:n], b[0:n])
{
  for (int x = 0; x < n; ++x){
    b[x] = a[x];
  }
}
#pragma acc exit data copyout(b[0:n]) delete(a[0:n])
for (int x = 0; x < n; ++x){
  if (fabs(a[x] - b[x]) > PRECISION){
    err += 1;
    break;
  }
}
```

- Design issues addressed
  - Data clauses are called as 2.5 specifies
  - Compute complexity is kept minimal
  - Internal testing determines memory type for some platform dependent testing

Contributions of the Test Suite

- Check for correctness of implementations of OpenACC features
- Reported bugs to both PGI and GCC
- Check and report ambiguities or misinterpretation in the OpenACC specification

Open-sourcing V&V Suite

- Validation and Verification Suite will soon be made available to everyone
  - Target deadline late August 2017
- https://github.com/OpenACCUserGroup/OpenACCV-V
- Feedback on the suite is welcome
- Contributions to the suite welcome
- BSD 3-Clause license

Tested Platforms

	K20	K80	P100	Ivy Bridge	Bulldozer	Power8+	Knights Landing
PGI	V16.10	V**16.10***	V17.1	V**16.10**** /17.3	V16.10	V17.1	V17.1***
GNU	V6.3-20170303	V6.0.0-20160415		V6.0.0-20160415	V6.3-20170303		

*K80 was run against many versions of PGI which will be seen in a following slide.
**Version 16.10 reported in preference of 17.3 since 16.10 was released as the community edition. Freely available at no cost for download
***Knights Landing is not officially supported; testing was performed by using –ta=haswell flag.

Results from PGI and GNU compilers

Architecture	PGI Pass Rate	GNU Pass Rate
K20	175/177	112/177
K80	175/177	113/177
Ivy Bridge	171/177	154/177
Bulldozer	172/177	157/177

Cross-Platform Performance of PGI

Architecture	Pass Rate
K20	175/177
K80	175/177
P100	175/177
Ivy Bridge	171/177
Bulldozer	172/177
Power8+	165/177
Knights Landing	167/177

PGI Improvement Over Versions

Compiler Version	Fortran Pass Rate	C Pass Rate	Fortran % Passed	C % Passed
14.10	60/86	67/91	69.8	73.6
15.1	64/86	80/91	74.4	87.9
15.5	65/86	80/91	95.6	87.9
15.10	68/86	84/91	79.1	92.3
16.1	69/86	84/91	80.2	92.3
16.4	82/86	84/91	95.3	92.3
16.7	85/86	90/91	98.8	98.9
16.10	85/86	90/91	98.8	98.9
17.1	85/86	90/91	98.8	98.9
17.3	85/86	90/91	98.8	98.9

These results are from running against K80

Future Work

- Near future work – open source V&V suite
- Improve documentation
  - Forward and backward referencing
- Addition of use-cases
- Build a more comprehensive suite (cross, orphan test cases)
- Keep up with the growing feature set of the specification
- Brainstorming building an infrastructure common to both OpenMP and OpenACC V&V