A Validation Testsuite for OpenACC 1.0

Cheng Wang*, Rengan Xu*, Sunita Chandrasekaran*, Barbara Chapman* and Oscar Hernandez†

*HPCTools Group, Department of Computer Science,
University of Houston, Houston, TX, 77004, USA
† Computer Science and Mathematics Division, Oak Ridge National Laboratory

in Workshop on Multithreaded Architectures and Applications,
in conjunction with IPDPS 2014, Phoenix, Arizona

May 23, 2014

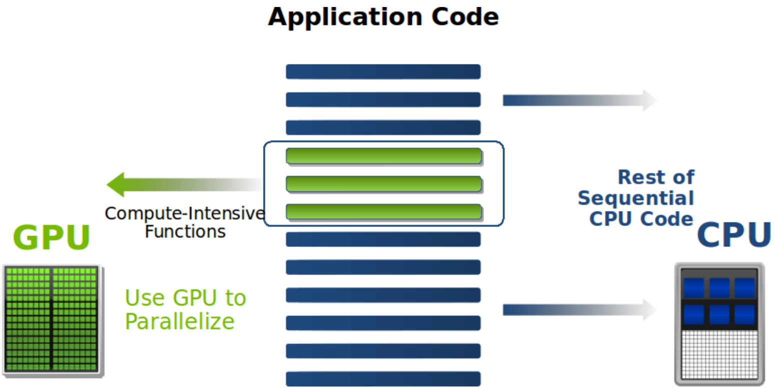
[1] The first two authors contribute equally to this work

Outline

1. Introduction to GPU computing
2. Introduction to OpenACC
3. OpenACC Validation Testsuite Design and Implementation
4. Results and Discussion
5. Future work

Why GPU Computing?

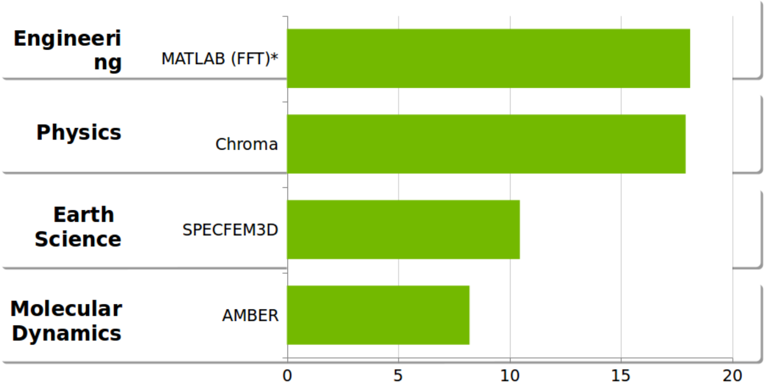
Add GPUs: Accelerate Science Applications



[1] Slide based on a talk from Mark Ebersole of NVIDIA, https://developer.nvidia.com

Why GPU Computing?

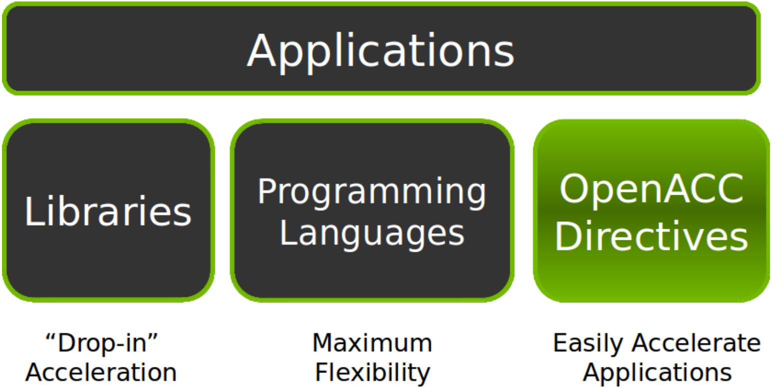
Tesla K20X Speed-Up over Sandy Bridge CPUs



[1] Slide based on a talk from Mark Ebersole of NVIDIA, https://developer.nvidia.com

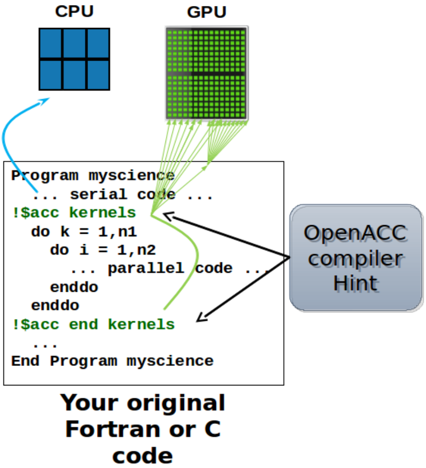
[2] CPU results: Dual socket E5-2687w, 3.10 GHz, GPU results: Dual socket E5-2687w + 2 Tesla K20X GPUs

Three ways to Accelerate Applications on GPUs



[1] Slide based on a talk from Will Ramey of NVIDIA, `https://developer.nvidia.com`

OpenACC Execution Model



1. Simple compiler hints specifying parallel regions
2. Compiler handles data between host and accelerators
3. Compiler parallelizes codes

[1] Slide based on a talk from Will Ramey of NVIDIA, https://developer.nvidia.com

High-level ... with Low-level Access

1. Compiler directives to specify parallel regions
   - Offload parallel regions
   - Portable across OSes, host CPUs, accelerators, and compilers

OpenACC
DIRECTIVES FOR ACCELERATORS

High-level ... with Low-level Access

1. Compiler directives to specify parallel regions
   - Offload parallel regions
   - Portable across OSes, host CPUs, accelerators, and compilers
2. Create high-level heterogeneous programs
   - Without explicit restructuring your code
   - Without explicit data transfers between host and accelerator

OpenACC
DIRECTIVES FOR ACCELERATORS

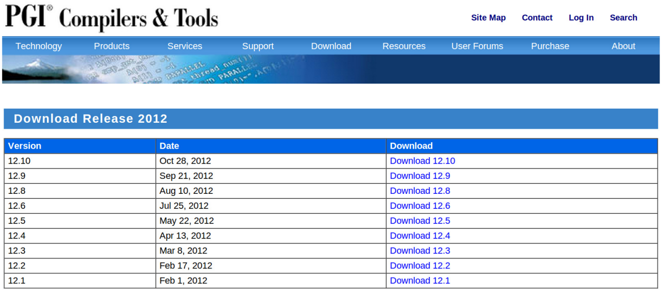
High-level ... with Low-level Access

1. Compiler directives to specify parallel regions
   - Offload parallel regions
   - Portable across OSes, host CPUs, accelerators, and compilers
2. Create high-level heterogeneous programs
   - Without explicit restructuring your code
   - Without explicit data transfers between host and accelerator
3. Compatible with other GPU languages and libraries
   - Interoperate between CUDA C/Fortran and GPU libraries
   - e.g. cuFFT, cuBLAS, cuSPARSE, etc.

OpenACC®
DIRECTIVES FOR ACCELERATORS

OpenACC Compilers

1. Developed by PGI/NVIDIA, CRAY and CAPS from early 2012
   - Evolving rapidly
   - On top of prior efforts
   - e.g. OpenHMPP, PGI Accelerator Directives, etc.


PGI® Compilers & Tools

Site Map | Contact | Log In | Search

Technology | Products | Services | Support | Download | Resources | User Forums | Purchase | About

Download Release 2012

Version	Date	Download
12.10	Oct 28, 2012	Download 12.10
12.9	Sep 21, 2012	Download 12.9
12.8	Aug 10, 2012	Download 12.8
12.6	Jul 25, 2012	Download 12.6
12.5	May 22, 2012	Download 12.5
12.4	Apr 13, 2012	Download 12.4
12.3	Mar 8, 2012	Download 12.3
12.2	Feb 17, 2012	Download 12.2
12.1	Feb 1, 2012	Download 12.1

OpenACC Compilers

1. Developed by PGI/NVIDIA, CRAY and CAPS from early 2012
   - Evolving rapidly
   - On top of prior efforts
   - e.g. OpenHMPP, PGI Accelerator Directives, etc.

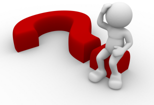
2. Different compiler vendors may interpret the new standard in different ways
   - More than one way of implementations for a given OpenACC feature
   - e.g. Mapping OpenACC gang/worker/vector clauses to CUDA grid/block/warp/threads
   - Inconsistent compiler behaviors

OpenACC Compilers

1. Developed by PGI/NVIDIA, CRAY and CAPS from early 2012
   - Evolving rapidly
   - On top of prior efforts
   - e.g. OpenHMPP, PGI Accelerator Directives, etc.

2. Different compiler vendors may interpret the new standard in different ways
   - More than one way of implementations for a given OpenACC feature
   - e.g. Mapping OpenACC gang/worker/vector clauses to CUDA grid/block/warp/threads
   - Inconsistent compiler behaviors

3. The OpenACC specification is evolving
   - Ambiguities is highly likely to happen



OpenACC Compiler Validation Suite

1. An OpenACC compiler validation suite to check for
   - Completeness
   - Correctness
   - Conformance to the standard

OpenACC Compiler Validation Suite

2. Benefit to OpenACC compiler developers
   - Implement the testing from users' perspective can find the defects and bugs, which developers are unable to detect during the code review process
   - Collaborate with OpenACC compiler vendors since the inception of the standard
   - Share experience

OpenACC Compiler Validation Suite

3. Benefit to OpenACC users
   - Verify the compiler used for their critical code
   - *Silent errors:* Compiler generates incorrect results in silence
   - Debugging is a pain when the software is big

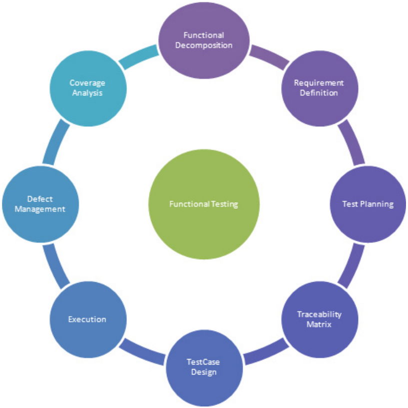
Road Map

1. OpenMP validation testsuite
    - **EWOMP '03**: *An OpenMP Validation Suite*, University of Stuttgart
    - **EWOMP '04**: *Validating OpenMP 2.5 for Fortran and C/C++*, University of Stuttgart & University of Houston
    - **IWOMP '12**: *An OpenMP 3.1 Validation Testsuite*, University of Houston
2. OpenACC validation testsuite
    - Adapted from the OpenMP validation suite
    - **2012**: Joint as an OpenACC member and worked with compiler vendors

OpenACC Compiler Validation Suite

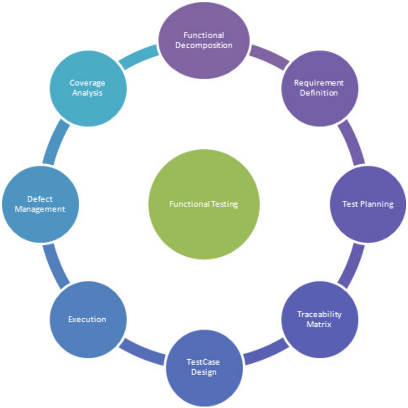
Test Infrastructure

1. Functional decomposition
   - Each functional test checks only one OpenACC feature



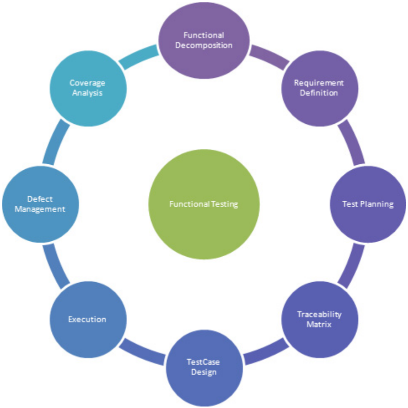
Test Infrastructure

2. Test planning
   - Users can set the compiler configuration details
   - Users can choose any feature sets to test



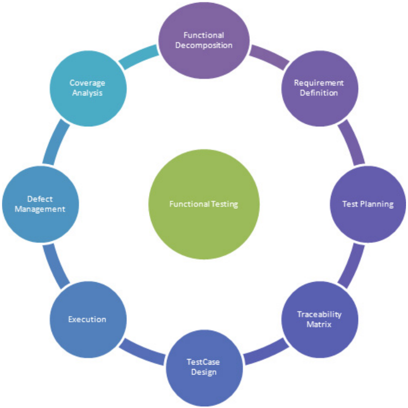
Test Infrastructure

3. Test case design
   - Maximize code reusability
   - Minimize development effort
   - Extensibility
   - Template-based testing



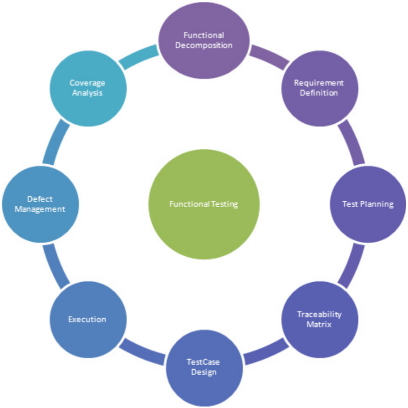
Test Infrastructure

4. Test execution
   - Automated testing
   - The test infrastructure can parse, compile, dispatch, and execute the tests automatically.



Test Infrastructure

5. Result analyzer and bug reporter
    - Not just tell the test “passed” or “fail”
    - A comprehensive bug report including test name, test description, test code snippet, detailed error information, etc..



Template-based Testing

```
<accts:test>
<accts:directive>
acc kernels copyin
</accts:directive>

<accts:dependences>
acc loop, acc data
</accts:dependences>

<accts:testcode>
  ...
<accts:functional>
 //Test case 1
 //Test case 2
</accts:functional>
  ...
<accts:cross>
// Cross test case 1
// Cross test case 2
</accts:cross>
...
</accts:testcode>
</accts:test>
```

1. Template is based on `xml` syntax
2. Requires minimum effort to develop each test case
3. The test infrastructure will parse the template and generate the standalone test cases automatically

Cross Testing

```
#pragma acc parallel num_gangs(10)
{
  #pragma acc loop
  for(int i = 0; i < N; i ++)
    A[i] = A[i] + 1;
}

```

Listing 1: Functional test for loop directive

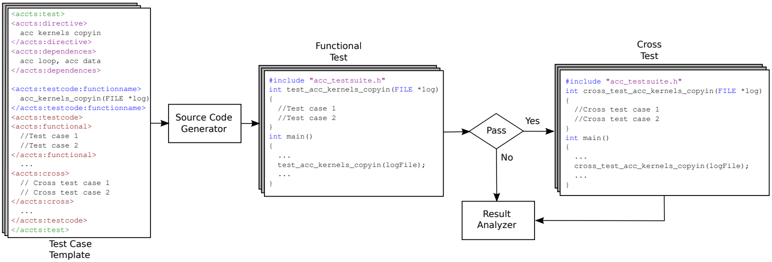
```
#pragma acc parallel num_gangs(10)
{
  for(int i = 0; i < N; i ++)
    A[i] = A[i] + 1;
}

```

Listing 2: Cross test

1. A deeper test methodology complementary to functional tests
2. Methodology: If remove/replace the the tested directive, the cross test should yield an "incorrect" result

Test Flow



1. Minimum development effort
2. Automated testing
3. Extensibility
4. Compatibility

Functional Testing

1. More than 160 functional tests covering nearly all OpenACC 1.0 C & Fortran directives, clauses, library routines, and environment variables.
2. Each functional test checks only one OpenACC feature
3. Basic methodology: Compares the result of the directive being tested with a pre-calculated value calculated on the host

Issues Found

1. More than 120 bugs reported to CAPS, PGI and CRAY since early 2012
2. Most of the bugs were quickly fixed in the next compiler release
3. Some bugs are due to ambiguities in the OpenACC specification
   - Most of them are clarified in the OpenACC specification 2.0

Issue #1

```c
int gangs = 8;

// Working
#pragma acc parallel num_gangs(8)

// Not working
#pragma acc parallel num_gangs(gangs)

```

Listing 3: Test for num_gangs() clause in parallel construct

1. Test variable expressions inside gang/worker/vector clauses
2. CAPS compiler prior version 3.1.0 only accepted constant expressions

Issue #2

```
#pragma acc kernels copyin(a[0:n], b[0:n]) copyout(c[0:n]) async(tag1)
for(int i = 0; i < n; i ++) {
  c[i] = a[i] + b[i];
}

// Test if the asynchronous test finishes
int done = acc_async_test(tag1);

#pragma acc kernels copyin(a[0:n], b[0:n]) copyout(d[0:n]) async(tag2)
for( int i = 0; i < n; i ++) {
  d[i] = a[i] + b[i];
}

```

Listing 4: Test for async clause in kernels construct

1. PGI compiler before 13.x failed on asynchronous activities
2. Worked around if moved the copyin and copyout clauses into a data directive

Interesting Findings #1: Device Type

1. OpenACC 1.0 specification defines fours types of devices
   - acc_device_none
   - acc_device_default
   - acc_device_host
   - acc_device_not_host
2. Different compilers have different namespace for `acc_device_not_host`
   - CAPS 3.3.3: acc_device_cuda, acc_device_opencl
   - PGI 13.4: acc_device_nvidia, acc_device_radeon, acc_device_xeonphi
3. Jeopardize the portability of OpenACC codes
4. OpenACC 2.0 standardizes the potential namespace

Interesting Findings #2: Unstructured Data lifetimes

```
#pragma acc data copyin(a[0:n])
   copyout(b[0:n])
 {
     ...
 }
```

Listing 5: Structured data lifetimes

```
void init(...) {
  ...
  #pragma acc enter data copyin(a[0:n])
  ...
}
...
void fini(...) {
  ...
  #pragma acc exit data copyout(b[0:n])
  ...
}
```

Listing 6: Unstructured data lifetimes

1. OpenACC 1.0 allows only structured data lifetime
2. Large software is usually composed by multiple files
3. OpenACC 2.0 defines two new directives: `enter/exit data`

Interesting Findings #3: Loop Nesting

```
#pragma acc kernels loop gang
for (int i = 0; i < n; i ++) {
  #pramga acc loop worker
  for (int j = 0; j < nn; j ++) {
    #pragma acc loop vector
    for (int k = 0; k < nnn; k ++ ) {
      ...
    }
  }
}
```

Listing 7: Parallelization of nested loops

1. Defines gang/worker/vector specifying hierarchical levels of parallelism
2. Multiple permutations and combinations are possible
3. Big performance inconsistencies cross different compilers
4. OpenACC 2.0 restricts the order

Interesting Findings #4: Procedure Calls

1. OpenACC 1.0 does not allow procedure calls inside a parallel/kernels region
2. Inconvenience for large applications
3. OpenACC 2.0 introduces the `routine` directive

Conclusion and Future Work

1. An OpenACC compiler validation suite to identify compiler bugs and check conformance to the standard
2. First-hand experience
3. OpenACC 2.0 validation suite is on the way

Beyond the Validation Suite ...

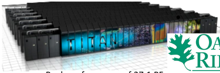
1. "Gray" box testing
    - The quality of handwriting codes depends on the expertise of the test developer
    - Likely to fall into the "comfort zone" of the tested compiler
    - Random testing (black box testing)
    - *Csmith: Finding and Understanding Bugs in C Compilers*, Yang *et. al*, PLDI '11
    - Random testing is good at detecting compiler "corner cases", but hard to check conformance to the standard

Beyond the Validation Suite ...

1. "Gray" box testing
   - The quality of handwriting codes depends on the expertise of the test developer
   - Likely to fall into the "comfort zone" of the tested compiler
   - Random testing (black box testing)
   - *Csmith: Finding and Understanding Bugs in C Compilers*, Yang *et. al*, PLDI '11
   - Random testing is good at detecting compiler "corner cases", but hard to check conformance to the standard
2. Performance "bugs"
   - Some compiler failures may not affect correctness but only performance

Software

1. Integrated into the test harness of the TITAN machine at Oak Ridge National Lab
2. Freely available for OpenACC members
3. Public access



- Peak performance of 27.1 PF
- 18,688 Compute Nodes each with:
  - Core AMD Opteron CPU (32 GB Memory)
  - NVIDIA Tesla "K20x" GPU (6 GB Memory)
- 512 Service and I/O nodes
- 200 Cabinets
- 710 TB total system memory
- Cray Gemini 3D Torus Interconnect

Application	Test	GPU	# nodes	Status
OpenACC_C	HMPP_GNU	yes	1	5 of 5 passed, failures (0 build, 0 submit, 0 run)
OpenACC_C	HMPP_INTEL	yes	1	6 of 6 passed, failures (0 build, 0 submit, 0 run)
OpenACC_C	HMPP_PGI	yes	1	9 of 9 passed, failures (0 build, 0 submit, 0 run)
OpenACC_C	PGI	yes	1	9 of 9 passed, failures (0 build, 0 submit, 0 run)
OpenACC_Fortran	HMPP_GNU	yes	1	9 of 9 passed, failures (0 build, 0 submit, 0 run)
OpenACC_Fortran	HMPP_INTEL	yes	1	9 of 9 passed, failures (0 build, 0 submit, 0 run)
OpenACC_Fortran	HMPP_PGI	yes	1	7 of 7 passed, failures (0 build, 0 submit, 0 run)
OpenACC_Fortran	PGI	yes	1	10 of 10 passed, failures (0 build, 0 submit, 0 run)
OpenCL_OpenACC	GNU	yes	1	5 of 5 passed, failures (0 build, 0 submit, 0 run)
OpenCL_OpenACC	PGI	yes	1	9 of 9 passed, failures (0 build, 0 submit, 0 run)

Q & A

Thank You For Your Attention