Extended OpenACC Programming to Exploit GPU-Specific Features Still at a High Level

Seyong Lee and Jeffrey S. Vetter

Future Technologies Group

Oak Ridge National Laboratory

http://ft.ornl.gov

Outline

- Issues in OpenACC and Other Directive-Based GPU Programming Models
- OpenACCe: Extended OpenACC to Support Architecture-Specific Features at High-Level
  - Extension to Better Support Unified Memory
  - Extension to Support Architecture-Specific Features
- Implementation and Evaluation
- Summary

Motivation

- Scalable Heterogeneous Computing (SHC)
  - Enabled by graphics processors (e.g., NVIDIA CUDA, AMD APU), Intel Xeon Phi, or other non-traditional devices.
  - Emerging solution to respond to the constraints of energy, density, and device technology trends.
  - However, the complexity in SHC systems causes portability and productivity issues.

What is OpenACC?

- Directive-based accelerator programming API standard to program accelerators
  - Consists of the compiler directives, library routines, and environment variables
  - Provide a high-level abstraction over architectural details and low-level programming complexities.
    - Allow parallel programmers to provide hints, known as “directives”, to the compiler, identifying which areas of code to accelerate, without requiring programmers to modify or adapt the underlying code itself.
  - Aimed at incremental development of accelerator code.

Issues In OpenACC and Other Directive-Based Accelerator Programming Models

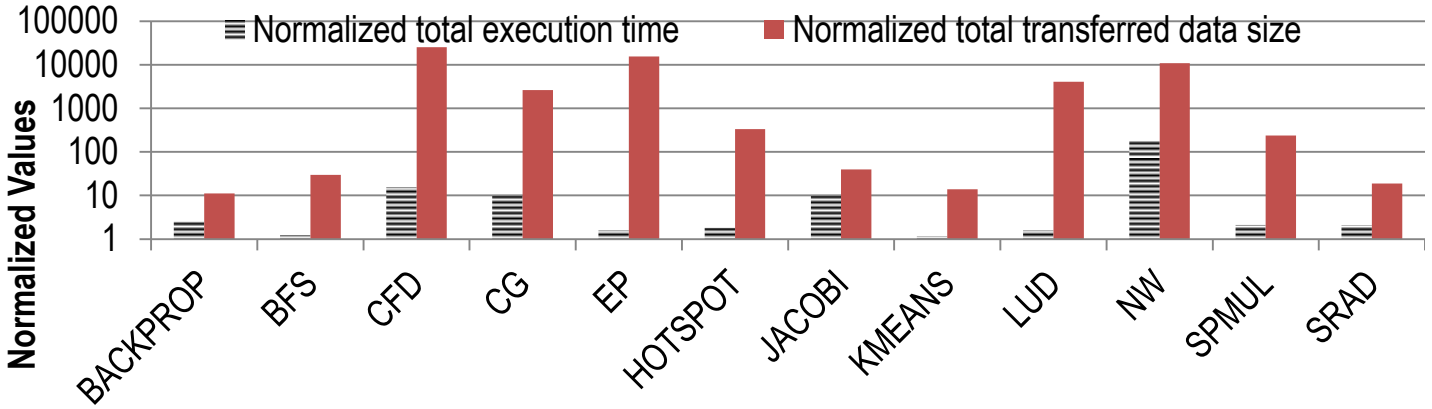
- Too much abstraction puts significant burdens on performance tuning, debugging, scaling, etc.
- We need in-depth evaluation and research on the directive-based, heterogeneous programming to address the two conflicting goals in SHC systems: ***productivity*** and ***performance***.

OpenACCe: Extended OpenACC to Better Support Architecture-Specific Features

- OpenACC Extension to Better Support Unified Memory
- OpenACC Extension to Support Accelerator-Specific Features

OpenACC Extension to Better Support Unified Memory

- Problem
  - Explicit GPU memory management in OpenACC (and other directive-based GPU programming models) can be still complex and error-prone.



Execution time and transferred data size with OpenACC default memory management scheme normalized to those fully optimized OpenACC version

OpenACC Extension to Better Support Unified Memory (2)

- Problem
  - Unified memory (NVIDIA CUDA 6 or AMD APUs) can simplify the complex and error-prone memory management in OpenACC.
  - However, the current OpenACC model will work well on unified memory only if the whole memory is shared by default.
  - Performance tradeoffs in existing unified memory systems need fine-grained control on using unified memory.

OpenACC Extension to Better Support Unified Memory (3)

- Proposed Solution
  - Extend OpenACC with new library routines to explicitly manage unified memory:
    - Work on both separate memory systems and unified memory systems.
    - Allow hybrid OpenACC programming that selectively combine separate memory and unified memory.

Augmented OpenACC Runtime Routines to Support Unified Memory

Runtime Routine	Description
acc_create_unified (pointer, size)	Allocate unified memory if supported; otherwise, allocate CPU memory using malloc()
acc_pcreate_unified (pointer, size)	Same as acc_create_unified() if input does not present on the unified memory; otherwise, do nothing.
acc_copyin_unified(pointer, size)	Allocate unified memory and copy data from the input pointer if supported; otherwise, allocate CPU memory and copy data from the input pointer.
acc_pcopyin_unified (ponter, size)	Same as acc_copyin_unified() if input data not present on the unified memory; otherwise, do nothing.
acc_delete_unified (pointer, size)	Deallocate memory, which can be either unified memory or CPU memory
Existing runtime routines and internal routines used for data clauses	Check whether the input data is on the unified memory; if not, perform the intended operations.

Hybrid Example to Selectively Combine both Separate and Unified Memories

```
float (*a)[N2]= (float(*)[N2]) malloc(..);
float (*b)[N2]= (float(*)[N2]) acc_create_unified(..);
...
#pragma acc data copy(b), create(a)
for (k = 0; k < ITER; k++) {
        #pragma acc kernels loop independent
        ...//kernel-loop1
} //end of k-loop
acc_delete_unified(a,...);  acc_delete_unified(b,...);
```

OpenACC Extension to Support Accelerator-Specific Features

- Problem
  - High-level abstraction in OpenACC does not allow user's control over compiler-specific or architecture-specific features, incurring noticeable performance gap between OpenACC and low-level programming models (e.g., CUDA and OpenCL)

	OpenACC translated by OpenARC	OpenACC translated by PGI
Normalized execution time over manual CUDA	31	12.8

Performance of Rodinia LUD benchmark on a NVIDIA Tesla M2090

OpenACC Extension to Support Accelerator-Specific Features (2)

- Proposed Solution
  - Extend OpenACC with new, device-aware directives
  - Enable advanced interactions between users and compilers still at high-level.
    - Allow users high-level control over compiler translations.
    - Most extensions are optional; preserve portability
    - Can be used to understand/debug internal translation processes.

Device-Aware OpenACC Extension

- Directive Extension for Device-Specific Memory Architectures

```
#pragma openarc cuda [list of clauses]
where clause is one of the followings:
        global constant, noconstant, texture,
        notexture, sharedRO, sharedRW, noshared,
        registerRO, registerRW, noregister
```

Device-Aware OpenACC Extension (2)

- Multi-Dimensional Work-Sharing Loop Mapping
  - Nested work-sharing loops of the same type is allowed if tightly nested, and the OpenACC compiler applies static mapping of the tightly nested work-sharing loops.
- Fine-Grained Synchronization
  - Add a new barrier directive (#pragma acc barrier) for local synchronizations (among workers in the same gang or vectors in the same worker)

OpenACCe Example

```
#pragma acc kernels loop gang(N/BSIZE) copy(C) copyin(A, B)
#pragma openarc cuda sharedRW(As, Bs)
for(by = 0; by < (N/BSIZE); by++) { //by is mapped to blockIdx.y
#pragma acc loop gang(N/BSIZE)
for(bx = 0; bx < (N/BSIZE); bx++) { //bx is mapped to blockIdx.x
    float As[BSIZE][BSIZE]; float Bs[BSIZE][BSIZE];
#pragma acc loop worker(BSIZE)
    for(ty = 0; ty<BSIZE; ty++) { //ty is mapped to threadIdx.y
#pragma acc loop worker(BSIZE)
    for(tx = 0; tx<BSIZE; tx++) { //tx is mapped to threadIdx.x
        … //computation part1
#pragma acc barrier
        … //computation part2
} } //end of the nested worker loops    } } //end of the nested gang loops
```

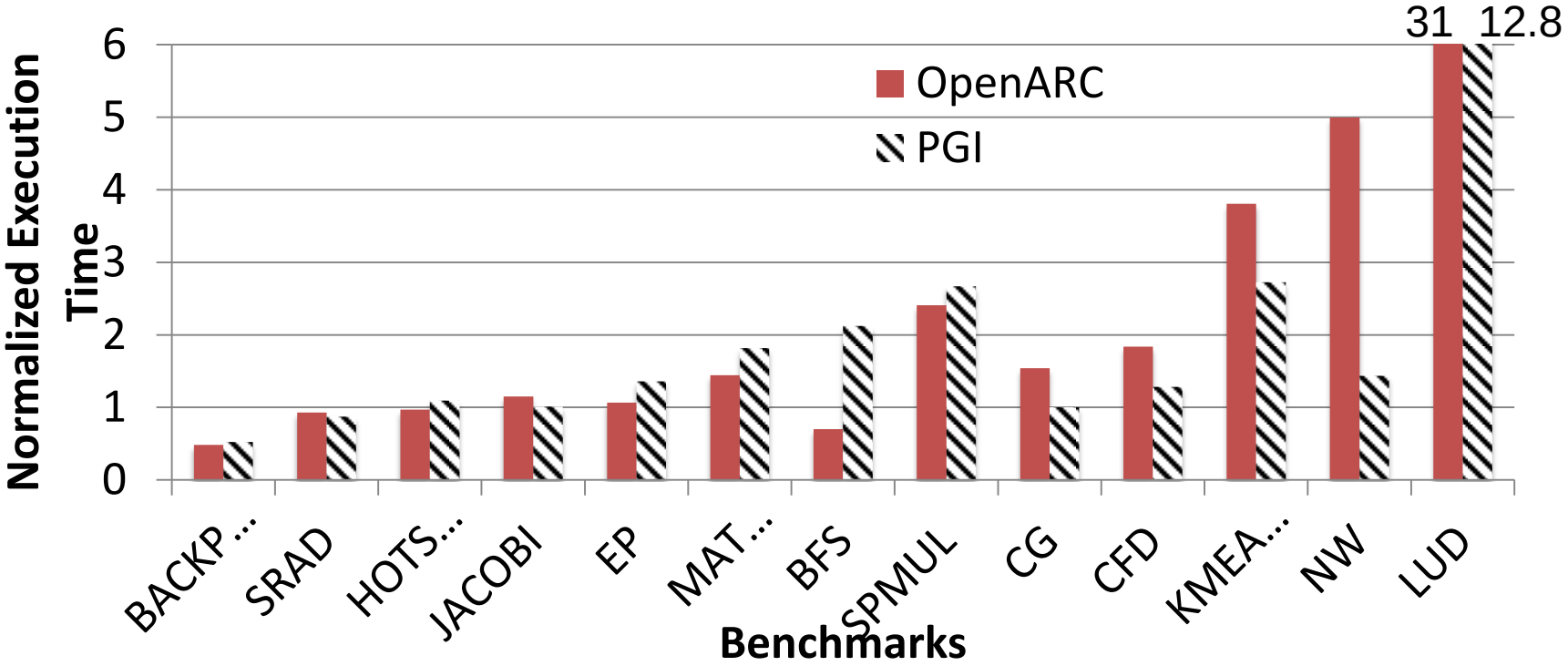
Implementation

- Proposed OpenACC extensions are fully implemented in the Open Accelerator Research Compiler (OpenARC).
- OpenARC: Open-Sourced, High-Level Intermediate Representation (HIR)-Based, Extensible Compiler Framework.
  - Perform source-to-source translation from OpenACC C to target accelerator models (CUDA or OpenCL).
  - Can be used as a research framework for various study on directive-based accelerator computing.

Evaluation

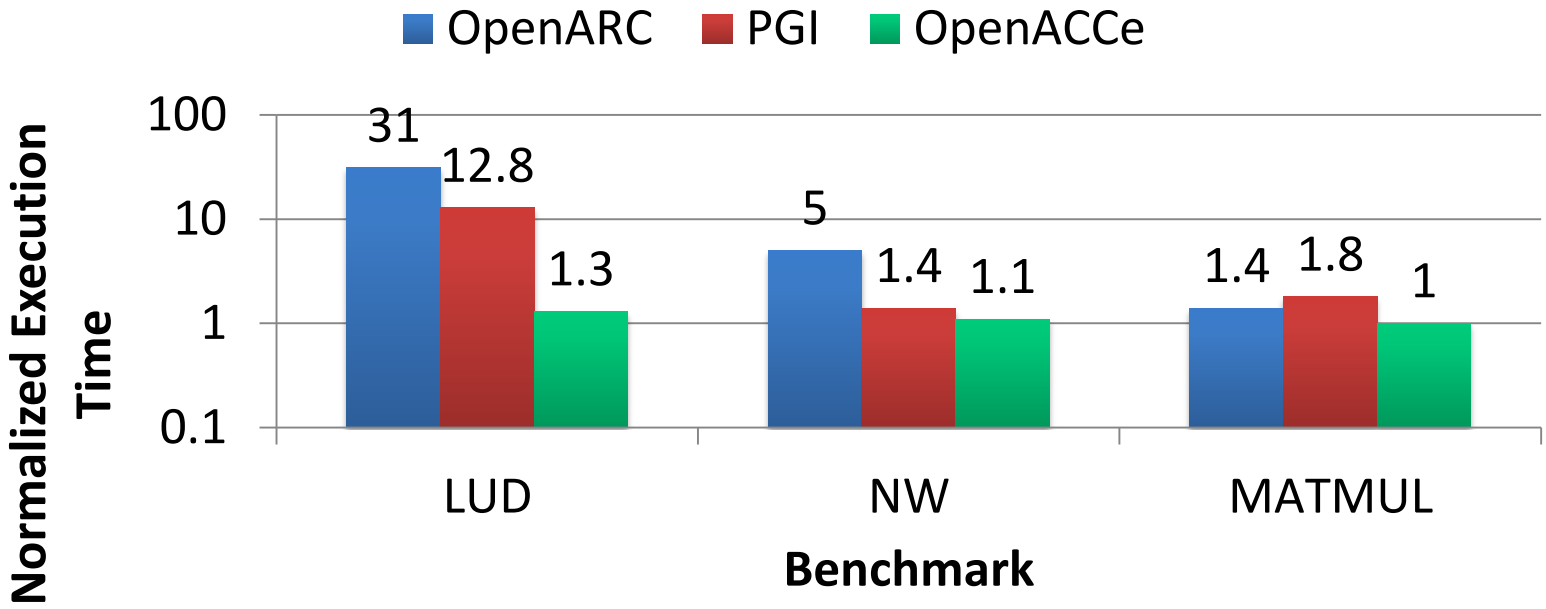
- Experimental Setup
  - 13 OpenACC programs from NPB, Rodinia, and kernel benchmarks are translated to CUDA programs by OpenARC.
  - Test Platforms
    - Unified memory test:
      - NVIDIA Tesla K40c and Intel Xeon E5520 CPU
      - NVCC V6.5 and GCC V4.4.7
    - All the other tests:
      - NVIDIA Tesla M2090 GPU and Intel Xeon X5600 CPU
      - NVCC V5.0, GCC V4.4.6, PGI V13.6

Performance of Standard OpenACC



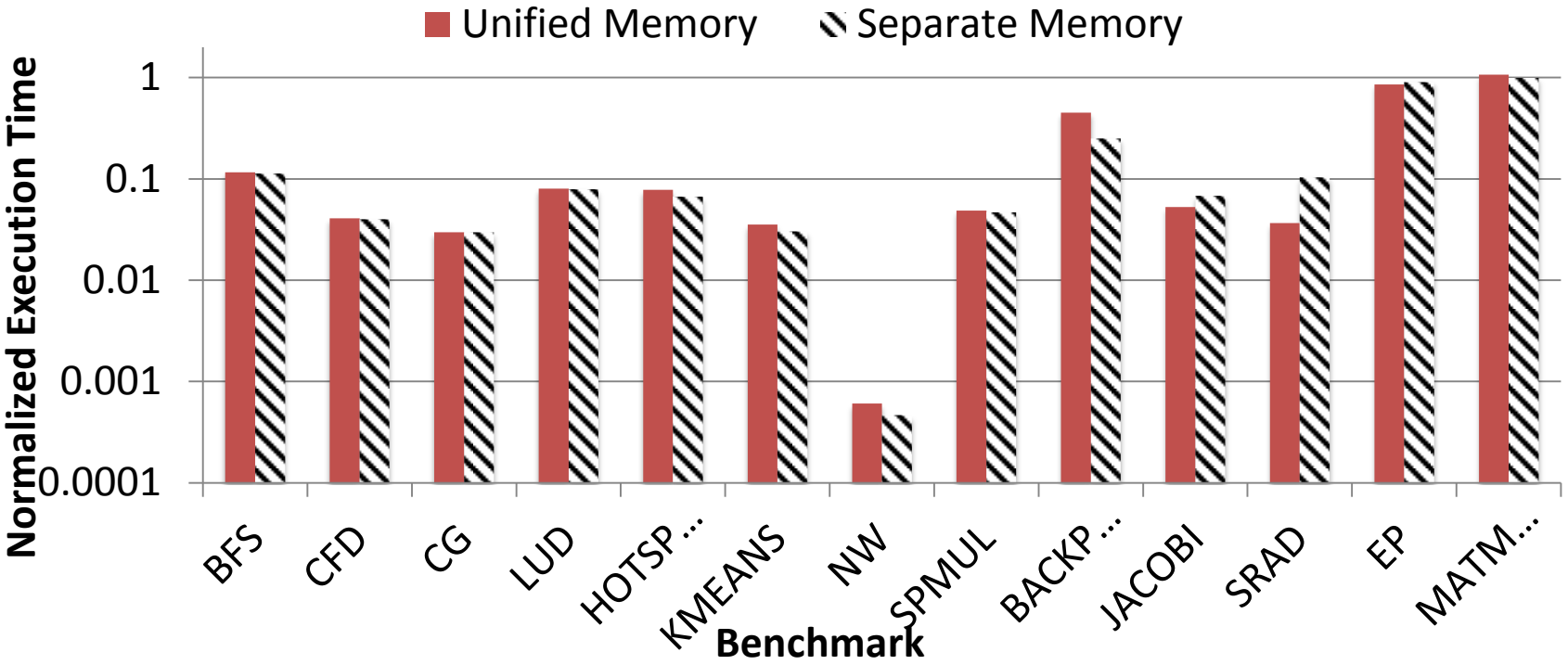
The execution times are normalized to those of hand-written CUDA versions. Lower is better.

OpenACCe Performance



The execution times are normalized to those of hand-written CUDA versions. Lower is better.

Unified Memory vs. Separate Memory



Execution times are normalized to no-memory-transfer-optimized versions.

Summary

- The proposed OpenACC extension to control architecture-specific features demonstrates that directive-based GPU programming models can achieve both programmability and performance.
- However, the proposed extension also exposes architectural details, while it hides complex language syntax of low-level CUDA models, raising a question of right balance between productivity and performance.
- Reference: Seyong Lee and Jeffrey S. Vetter, *OpenARC: Extensible OpenACC Compiler Framework for Directive-Based Accelerator Programming Study*, WACCPD, 2014

Thank You!

Contact Us

http://ft.ornl.gov/research/openarc

Email: lees2@ornl.gov

OpenACC Example