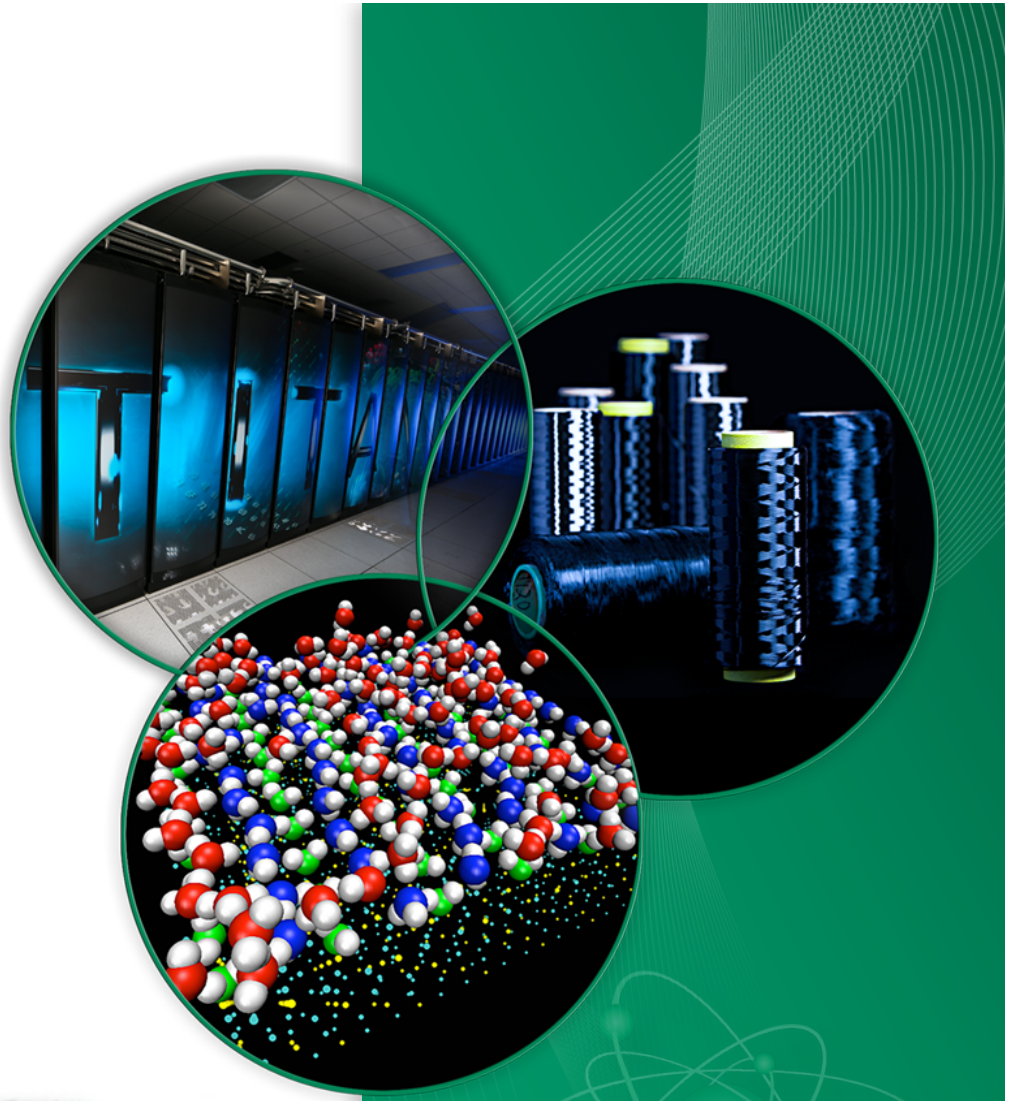
OpenARC: Open Accelerator Research Compiler (mini-workshop)



Seyong Lee

Future Technologies Group

Oak Ridge National Laboratory

lees2@ornl.gov

http://ft.ornl.gov

Motivation

- Scalable Heterogeneous Computing (SHC)
  - Enabled by graphics processors, 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.

Directive-Based Accelerator Programming Models

- Provide abstraction over architectural details and low-level programming complexities.
- However, too much abstraction puts significant burdens on:
  - Performance tuning
  - Debugging
  - Scaling
- We need in-depth evaluation and research on the directive-based, heterogeneous programming to address the two conflicting goals in SHC systems: productivity and portability.

Related Work

- Custom-Directive-based GPU Compiler
  - hiCUDA, OpenMPC
  - PGI accelerator compiler, HMPP, Rstream, etc.
- OpenACC compiler
  - PGI compiler, CAPS compiler, Cray compiler, etc.
  - accULL, OpenUH
- OpenMP4 compiler for accelerator computing
  - HOMP, GCC, Intel compiler

OpenARC: Open Accelerator Research Compiler

- Open-Sourced, High-Level Intermediate Representation (HIR)-Based, Extensible Compiler Framework.
  - Perform source-to-source translation from OpenACC C to target accelerator models.
    - Support full features of OpenACC V1.0 (+ subset of OpenACC V2.0 + array reductions and function calls)
    - Support both CUDA and OpenCL as target accelerator models (NVIDIA GPUs, AMD GPUs, Xeon Phi, Altera FPGAs, etc.)
    - OpenMP V4.0 support is being tested.
  - Provide common runtime APIs for various back-ends
  - Can be used as a research framework for various study on directive-based accelerator computing.
    - Built on top of Cetus compiler framework, equipped with various advanced analysis/transformation passes and built-in tuning tools.
    - OpenARC's IR provides an AST-like syntactic view of the source program, easy to understand, access, and transform the input program.

Design Goals of OpenARC

- Extensibility
  - Extensible OpenARC IR class hierarchy:
    - Its IR class hierarchy is easily extended to embrace new language constructs.
  - Rich semantic annotations:
    - Each OpenARC IR object can be augmented with rich annotations.
    - User can create different annotation types by deriving from a base PragmaAnnotation class.
  - Common, abstract OpenARC runtime API:
    - Unified interface between the program and the target-specific runtime

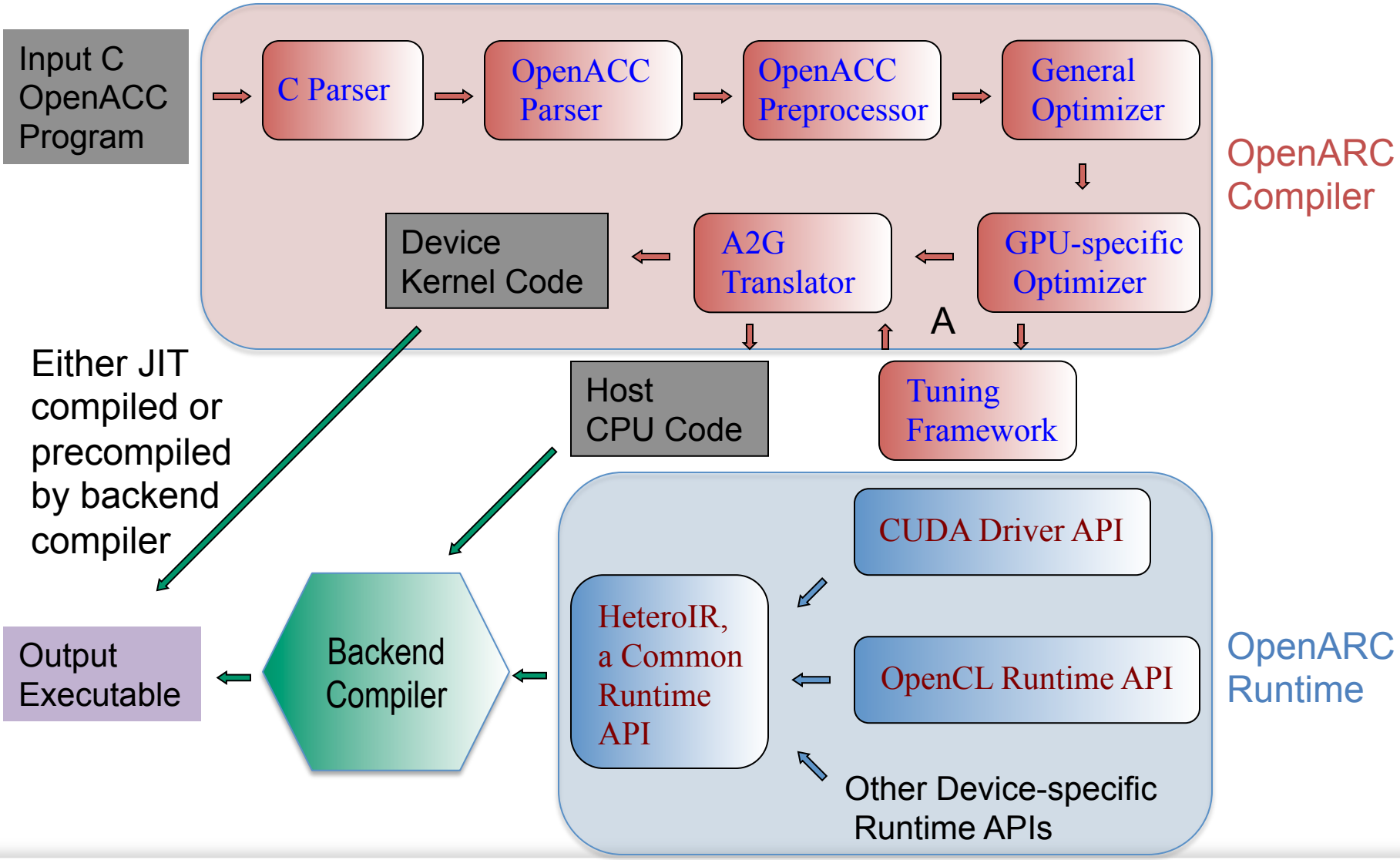
Overall Design of OpenARC (cont.)

- Debuggability
  - AST-like HIR, combined with abstract OpenARC runtime, allows to generate output codes similar to input program.
  - Clear separation between analyses and transformations
    - All analysis outputs are stored as internal annotations, used as input to transformation passes ➔ Easy to build traceability mechanisms

Design Goals of OpenARC (cont.)

- Tunability
  - Provide a rich set of directives to control various compiler optimizations and hardware-specific features.
  - Its built-in tuning tools combined with the rich OpenARC directives allow users to control overall OpenACC-to-Accelerator translation in a fine-grained, but still abstract manner.

OpenARC System Architecture



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

OpenARC Runtime API

- HeteroIR: High-Level, Architecture-Independent Intermediate Representation
  - Used as an intermediate language to map high-level programming models (OpenACC) to diverse heterogeneous devices.
- Primary Constructs in HeteroIR (Partial List)

Configuration
Kernels
Memory
Synchronization

HI_set_device (device type, device number)	Set the device to use
HI_init ()	Initialize the previously selected device.
HI_reset ()	Deinitialize the device.
HI_malloc (pointer, size, queue)	Allocate memory on the device.
HI_memcpy (source, destination, size, queue)	Perform synchronous/asynchronous data transfers.
HI_get_device_address (device address, host address)	Check if the host address has a valid mapping on the device. Return the device address if mapping is present.
HI_free (pointer, queue)	Free the device memory pointed to by the pointer.
HI_register_kernel_arg (kernel name, parameter, …)	Attach the argument to the corresponding kernel.
HI_kernel_call (kernel name, grid size, queue)	Launch the kernel with the specified grid size on the specified queue.
HI_set_async (queue)	Set the specified queue number.
HI_async_wait (queue)	Wait until all actions on the specified queue are finished.
HI_async_wait_all ()	Wait for all queues to finish the actions queued on them.

OpenARC Directive Extension and Environment Variables

- OpenARC Directives

  #pragma omp openmp-directive [clause [,] clause]…]

  #pragma acc openacc-directive [clause [,] clause]…]

  #pragma cetus [clause[,] clause]…]

  #pragma acc internal [clause [,] clause]…]

  #pragma openarc cuda [clause [,] clause]…]

  #pragma openarc resilience-directive [clause [,] clause]…]

  #pragma openarc profile-directive [clause [,] clause]…]

  #pragma aspen aspen-directive [clause [,] clause]…]

- OpenARC Environment Variables
  - Control the program-level behavior of various optimizations or execution configurations for an output GPU program.

OpenARC Application 1:
OpenACC Extension to Better Support Unified Memory

- 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.
- 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
        #pragma acc kernels loop independent
        ...//kernel-loop2
} //end of k-loop
... //b is accessed by CPU
acc_delete_unified(a,...);
acc_delete_unified(b,...);
```

OpenARC Application 2:
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)
- Solution
  - Propose a device-aware OpenACC extension (OpenACC-e)
  - Enable advanced interactions between users and compilers still at high-level.
    - Allow users high-level control over compiler translations.
    - Can be used to understand/debug internal translation processes.
    - Most extensions are optional; preserve portability

OpenACC-e: 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
```

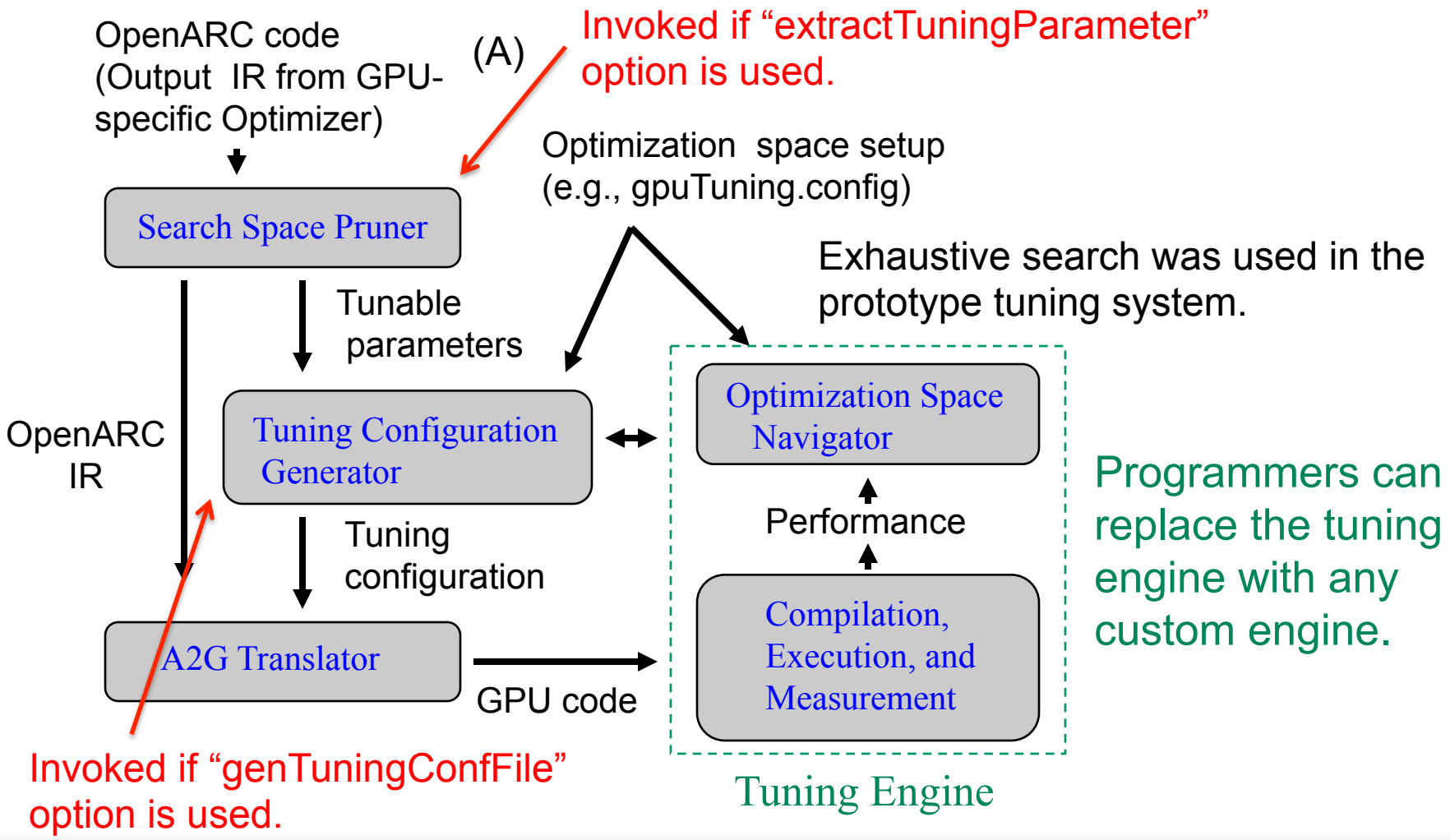
- Multi-Dimensional Work-Sharing Loop Mapping
  - All nested work-sharing loops of the same type, if tightly nested, and OpenARC applies static mapping of the tightly nested work-sharing loops.
- Fine-Grained Synchronization
  - Add a new barrier directive (#pragma acc barrier) for local synchronizations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

OpenACC-e Example

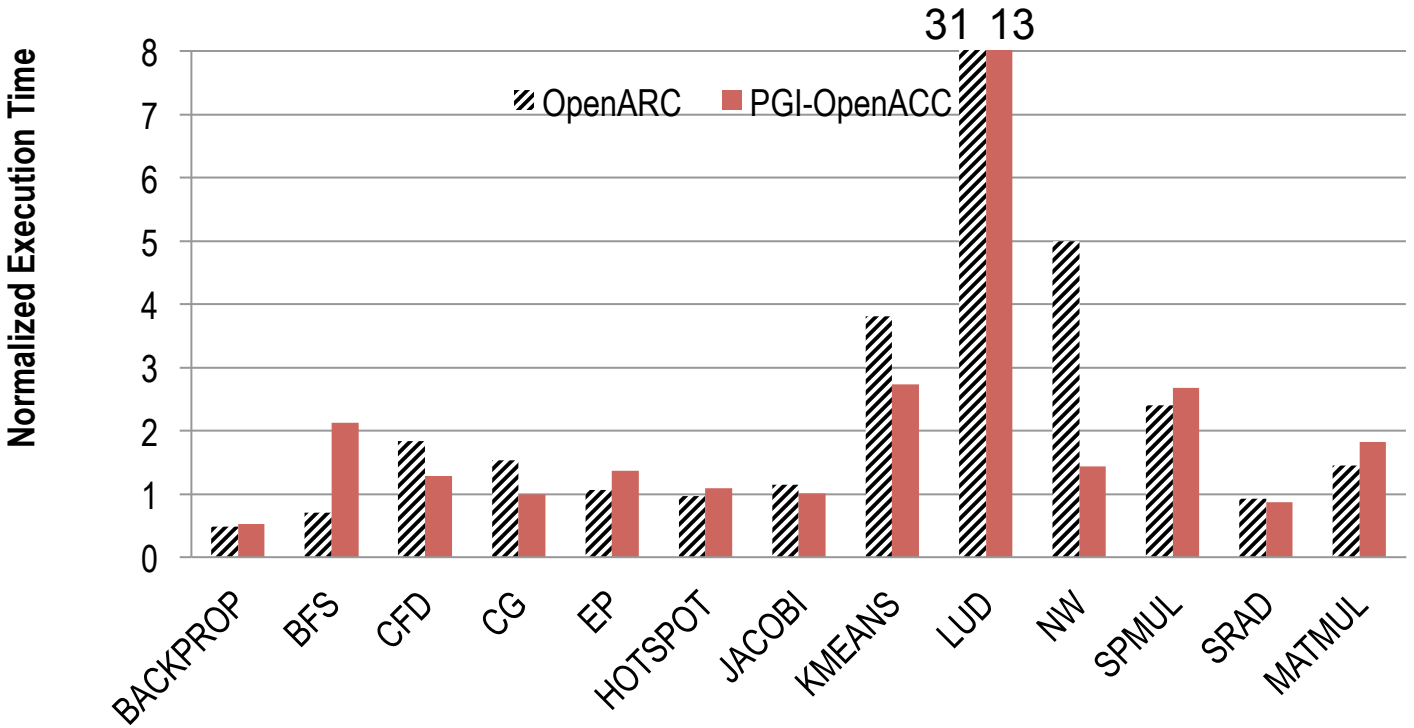
```
#pragma acc kernels loop gang(N/BSIZE) copy(C[0:N*N])
copyin(A[0:N*N], B[0:N*N])
#pragma openarc cuda sharedRW(As, Bs)
for(by = 0; by < (N/BSIZE); by++) { //by is mapped to blockIdx.y
#pragma acc loop gang(N/BSIZE)
for(bx = 0; bx < (N/BSIZE); bx++) { //bx is mapped to blockIdx.x
   float As[BSIZE][BSIZE]; 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
#pragma acc barrier
      … //computation part3
   } } //end of the nested worker loops
} }    //end of the nested gang loops
```

OpenARC Application 3:

Performance Tuning using a built-in Tuning Framework in OpenARC

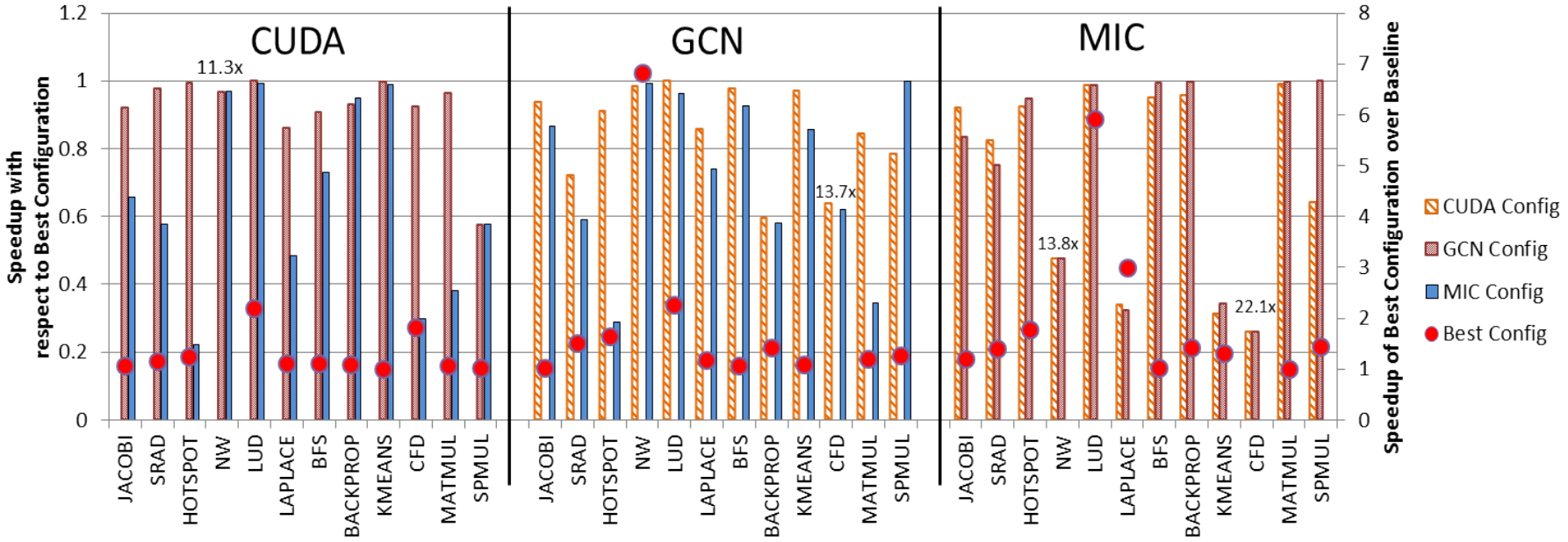


Preliminary Evaluation of OpenARC



Performance of OpenARC and PGI-OpenACC compiler (V13.6) relative to manual CUDA versions (Lower is better.)

Preliminary Evaluation of OpenARC (2)



Performance Portability Evaluation: Best performing OpenACC on one architecture may not produce best performance on another architecture. Tuning can be highly beneficial over the baseline translated code.

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

Summary

- OpenARC is an open-sourced, High-level Intermediate Representation (HIR)-based, extensible compiler framework.
- HIR with a rich set of directives in OpenARC provides a powerful research framework for various source-to-source translation and instrumentation experiments.
- The additional OpenARC directives with its built-in tuning tools allow users to control overall OpenACC-to-GPU translation in a fine-grained, but still abstract manner.

Session 2: How to Use OpenARC (User Guide)

- Getting Started
  - Requirements
  - Installation
  - Environment setup and running OpenARC
  - Known Issues
- OpenARC Research
  - OpenACC to FPGA
  - Interactive program debugging and optimization
  - Directive-based, automatic performance modeling
  - Adaptive integration of MPI and OpenACC
  - Directive-based, application-driven fault injection
  - Programming System for Non-Volatile Memory

Getting Started

- Requirements
  - JAVA SE 7 or later
  - GCC
  - ANTLRv2
    - Default antlr.jar file is included in the OpenARC distribution.
- Installation
  - Obtain OpenARC distribution
    - No public distribution is available yet.
    - Internal Git repository: https://code.ornl.gov/f6l/OpenARC.git
    - Contact us to access the source code (lees2@ornl.gov)

Getting Started (2)

- Installation
  - Build OpenARC compiler:
    - Go to the root directory of OpenARC.
    - Copy “make.header.sample” to “make.header”.
    - Update “make.header” as necessary.
    - Use provided “build.sh” or “build.xml” files.

      $ build.sh bin //compile, jar, and generate a wrapper file.
    - Refer to “REAME.md” file for more details.
  - Build OpenARC runtime:
    - Go to ./openarcrt directory
    - Run “batchmake.bash” script

      $ batchmake.bash

Running OpenARC

- Environment Setup
  - Set OPENARC_ARCH to 0 for CUDA (default), 1 for OpenCL, 2 for OpenCL for Xeon Phi, or 3 for OpenCL for Altera FPGA.
  - Set OpenACC environment variables (e.g., ACC_DEVICE_TYPE, ACC_DEVICE_NUM, etc.) to use non-default configuration.
  - Set OMP_NUM_THREADS to the maximum number of OpenMP threads if OpenMP is used in the input OpenACC program.
  - Set OPENARC_JITOPTION to pass options to the backend runtime compiler (e.g., NVCC options for JIT CUDA kernel compilation or clBuildProgram options for JIT OpenCL kernel compilation.)
  - Set OPENARCRT_VERBOSITY to set the verbosity level (>= 0) of the OpenARC runtime in the profile mode.
- Running OpenARC

  $ java –classpath=<user_class_path> openarc.exec.ACC2GPUDriver <OpenARC-options> <Input C files>

OpenARC commandline options

- To find a list of available OpenARC options, use "-help" or "-dump-options"

  $ java openacc.exec.ACC2GPUDriver –dump-options

- To pass multiple options in one file, use "-gpuConfFile=confFile" option.
- Some Important options
  - macro: used to pass macros to the C preprocessor.
  - addIncludePath: used to pass paths for header files.
  - … (many more optional ones)

Known Issues on the Current OpenARC Implementation

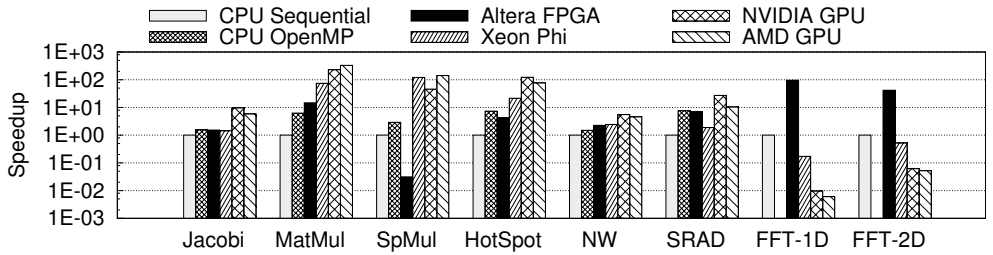
- The C parser supports C99 features only partially.
  - E.g., mixed declaration and code, C99 attributes, and C99 macros are only partially supported.
- GNU C preprocessor does not expand macros in pragma annotation.
  - To expand in annotations, use "#pragma acc #define …" directive or macro commandline option.
- Struct member is not allowed in OpenACC data clauses.
- Partial array passing is allowed only if the start index is 0.
- OpenACC vector clauses are ignored.
- OpenACC data regions can have compute regions in called functions, but a compute region can have a function call only if the called function does not contain worksharing loops.

Session 2: How to Use OpenARC (User Guide)

- Getting Started
  - Requirements
  - Installation
  - Environment setup and running OpenARC
  - Known Issues
- OpenARC Research
  - OpenACC to FPGA
  - Interactive program debugging and optimization
  - Directive-based, automatic performance modeling
  - Adaptive integration of MPI and OpenACC
  - Directive-based, application-driven fault injection
  - Programming System for Non-Volatile Memory

OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing

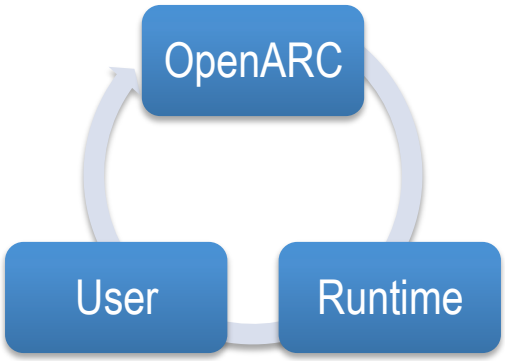
- Problem
  - Reconfigurable computers, such as FPGAs, offer more performance and energy efficiency for specific workloads than other heterogeneous systems, but their programming complexities and low portability have limited their deployment in large scale HPC systems.
- Solution
  - Proposed an OpenACC-to-FPGA translation framework, which performs source-to-source translation of the input OpenACC program into an output OpenCL code, which is further compiled to an FPGA program by the underlying backend Altera OpenCL compiler.
- Recent Results
  - Proposed several FPGA-specific OpenACC compiler optimizations and pragma extensions to achieve higher throughput.
  - Evaluated the framework using eight OpenACC benchmarks, and measured performance variations on diverse architectures (Altera FPGA, NVIDIA/AMD GPUs, and Intel Xeon Phi).

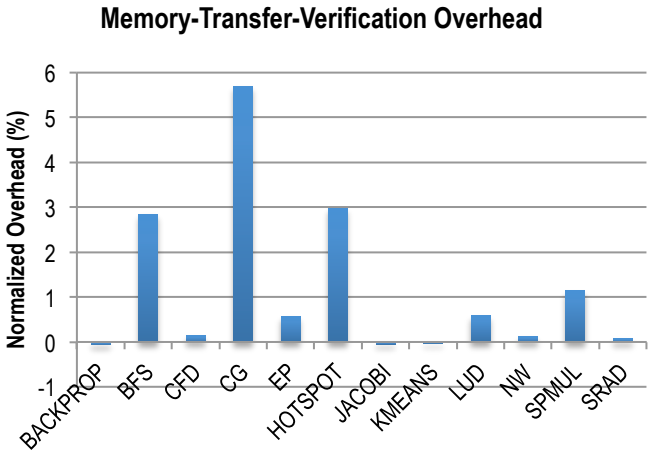


- Impact
  - Proposed translation framework is the first work to use a standard and portable, directive-based, high-level programming system for FPGAs.
  - Preliminary evaluation of eight OpenACC benchmarks on an FPGA and comparison study on other accelerators identified that the unique capabilities of an FPGA offer new performance tuning opportunities different from other accelerators.

Interactive Program Verification and Optimization with OpenARC

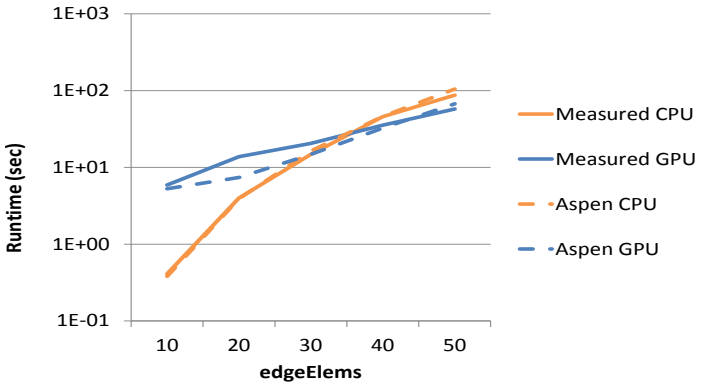
- Problem
  - *Too much abstraction* in directive-based GPU programming!
    - Debuggability: difficult to diagnose logic errors and performance problems
    - Performance: difficult to find where and how to optimize
- Solution
  - Directive-based, interactive GPU program verification and optimization
    - OpenARC compiler: generates runtime codes for verification and optimization
    - Runtime: 1) locate trouble-making kernels and 2) detect incorrect/missing/redundant memory transfers.
    - Users: iteratively fix/optimize incorrect kernels/ memory transfers based on the runtime feedback and apply to input program.
- Results
  - Evaluation using twelve OpenACC applications could detect all active errors affecting program outputs and optimize memory transfers comparable to a fully manual memory management scheme.





COMPASS: A Framework for Automated Performance Modeling and Prediction

- Problem
  - Flexible, accurate performance predictions offer numerous benefits such as gaining insight into and optimizing applications and architectures. However, the development and evaluation of such performance predictions has been a major research challenge, due to the architectural complexities.
- Solution
  - Designed and implemented an automated performance model generation and prediction system called COMPASS.
  - COMPASS generates a structured Aspen performance model from the target application's source code using automated static analysis, and evaluates this model using various performance prediction techniques.
- Recent Results
  - The proposed system automatically generated performance models for various types of applications (Kernel benchmarks, Rodinia benchmarks, and a DOE proxy application (LULESH)), and the models were used for various performance predictions, such as resource usage and runtime predictions.

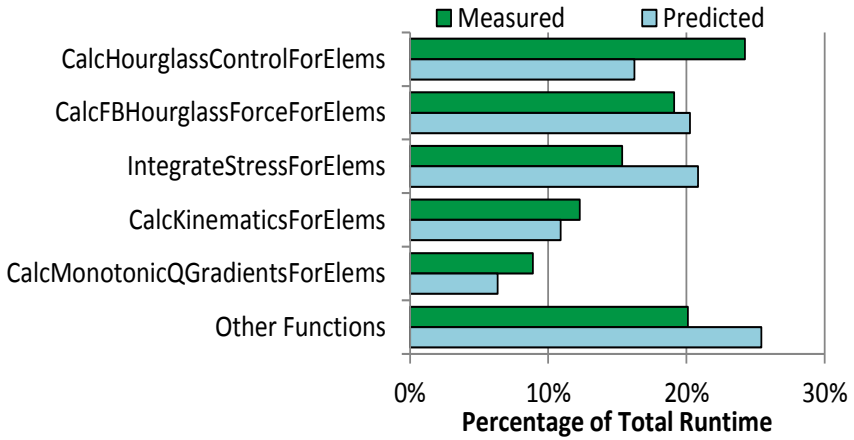


Measured and predicted runtime of the entire LULESH program on CPU and GPU

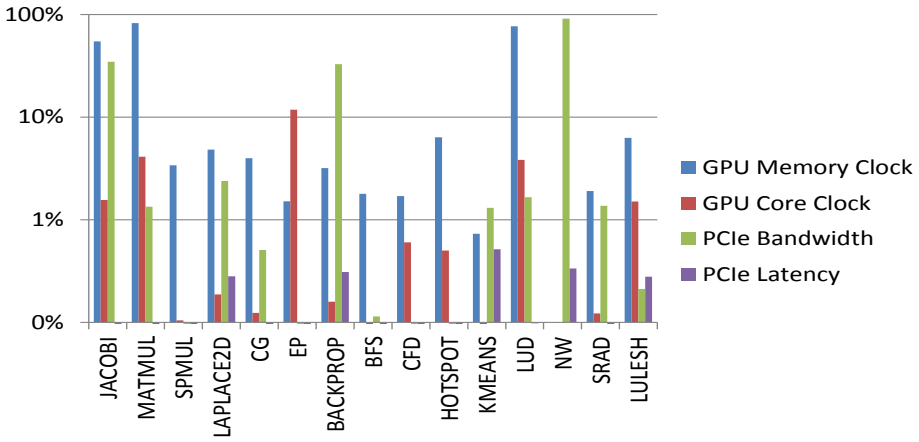
http://ft.ornl.gov/research/openarc

COMPASS: A Framework for Automated Performance Modeling and Prediction (cont.)

- Recent Results (cont.)
  - COMPASS prototype automatically generates the performance model of LULESH, which enables automatic performance prediction of LULESH on current and future DOE systems, without access to these systems.
  - The new kernel-level performance prediction tool of COMPASS can identify the five most expensive fictions in LULESH, which exactly match empirical results measured by gprof.
  - Resource usage prediction of COMPASS for various applications including LULESH shows that the prediction error is less than 20% from the measured values in all cases.
- Impact
  - The tight yet orthogonal integration of the performance model with the programming system can be used for various purposes, from answering basic questions such as predicting resource usages for current and future architectures, to complex performance questions, such as predicting runtimes under different application and system configurations.



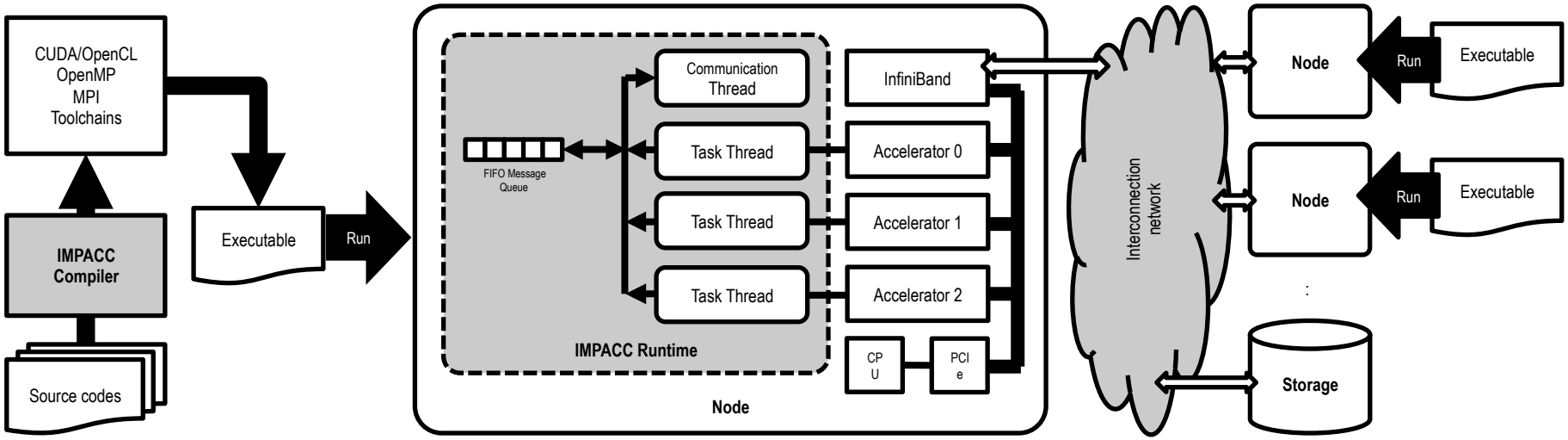
Predicted gprof Output of LULESH



Sensitivity to HW Parameters

IMPACC: A Framework for Adaptive Integration of MPI and OpenACC

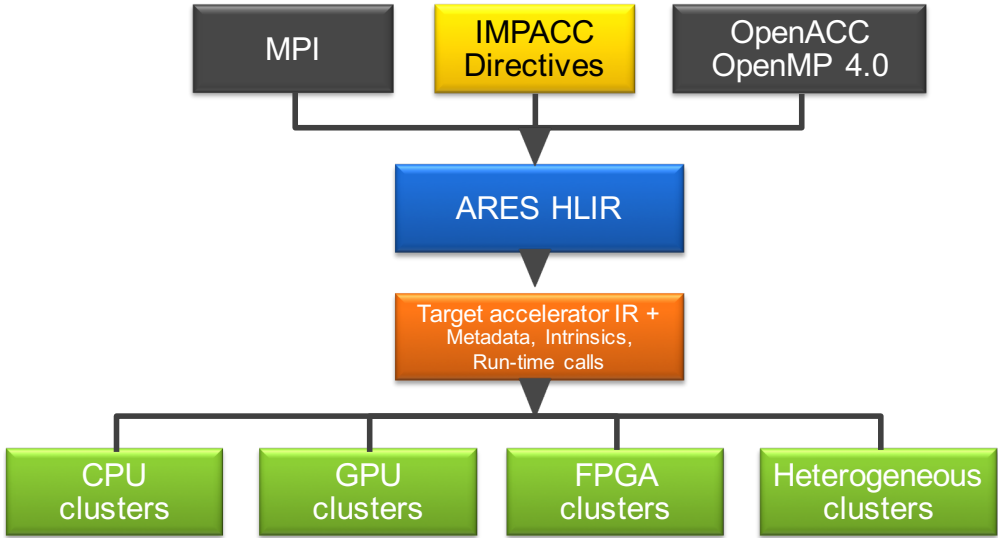
- Problem
  - Hybrid MPI+OpenACC programming model for heterogeneous clusters causes some inefficiencies and complexities, such as redundant data movement and excessive synchronization.
- Solution
  - Integrates MPI communication routines and OpenACC accelerator memory copy to eliminate redundant communication both between MPI tasks and between MPI and OpenACC memory model layers.
  - Integrates non-blocking MPI communication to the OpenACC asynchronous activity queue in order to allow seamless streamlining of asynchronous intra-node/internode communication.

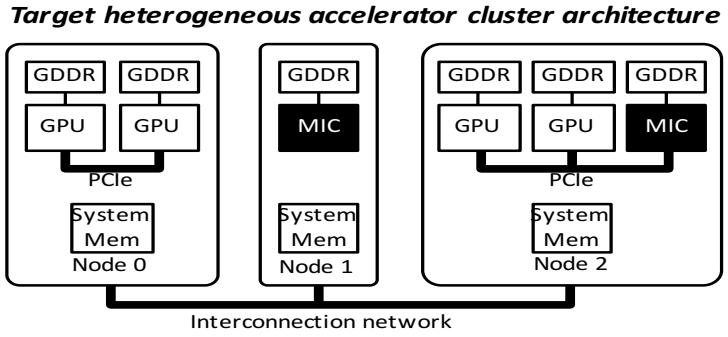


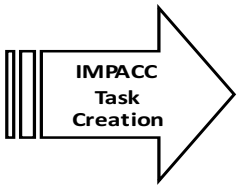
http://ft.ornl.gov/research/openarc

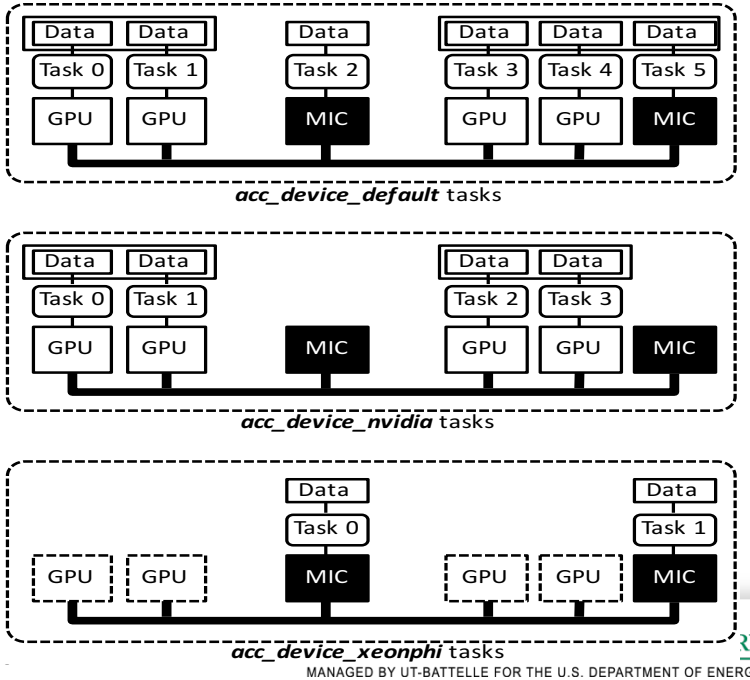
IMPACC: A Framework for Adaptive Integration of MPI and OpenACC

- Approach
  - The code written with MPI + OpenACC/ OpenMP 4.0 + IMPACC directives is translated into ARES HLIR.
  - IMPACC compiler translates ARES HLIR into the target accelerator IR + metadata, intrinsic and run-time calls and finally generates the executable binary for the target accelerator-based systems such as CPU, GPU, Xeon Phi, FPGA, and heterogeneous clusters.



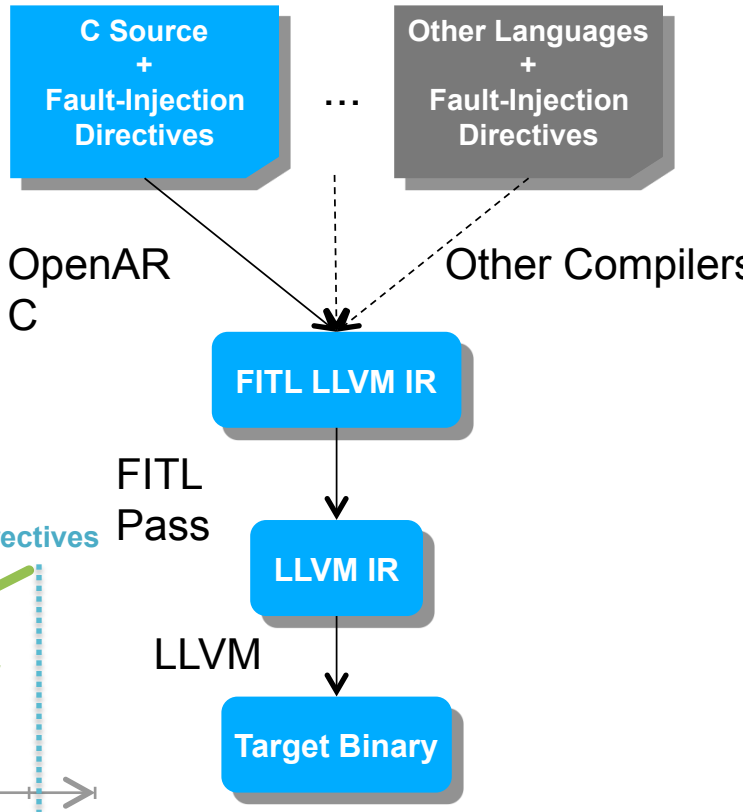


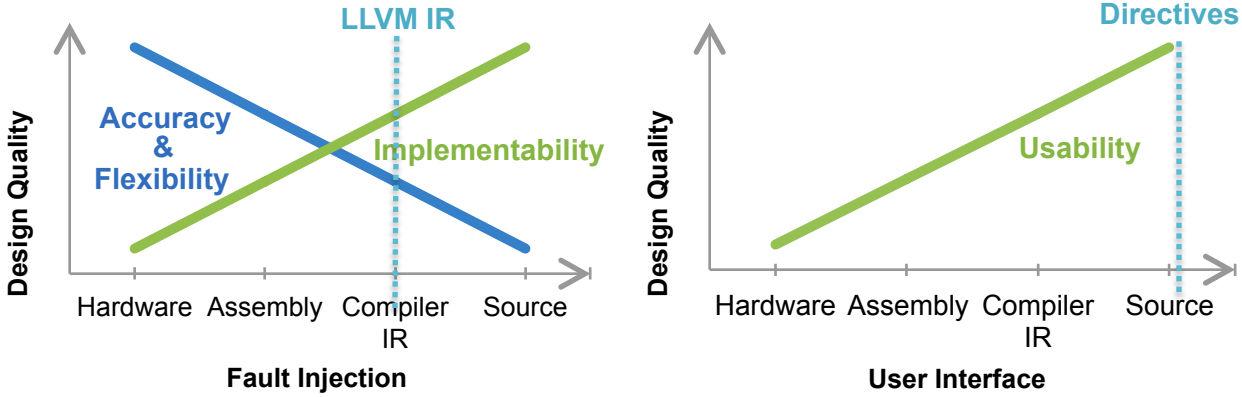




FITL: Directive-Driven Fault-Injection for LLVM

- Problem
  - Extreme-scale will expose hardware faults to apps.
  - Fault injectors are needed to study app resilience.
  - Low-level fault injectors are hard to configure and understand in terms of app source.
- Solution
  - Novel set of fault-injection pragmas for C
  - FITL: a set of LLVM extensions for fault injection
  - Abstractions for translating pragmas to FITL
  - Good balance between usability and accuracy/ implementability by combining fault-injection pragmas and LLVM-level implementation

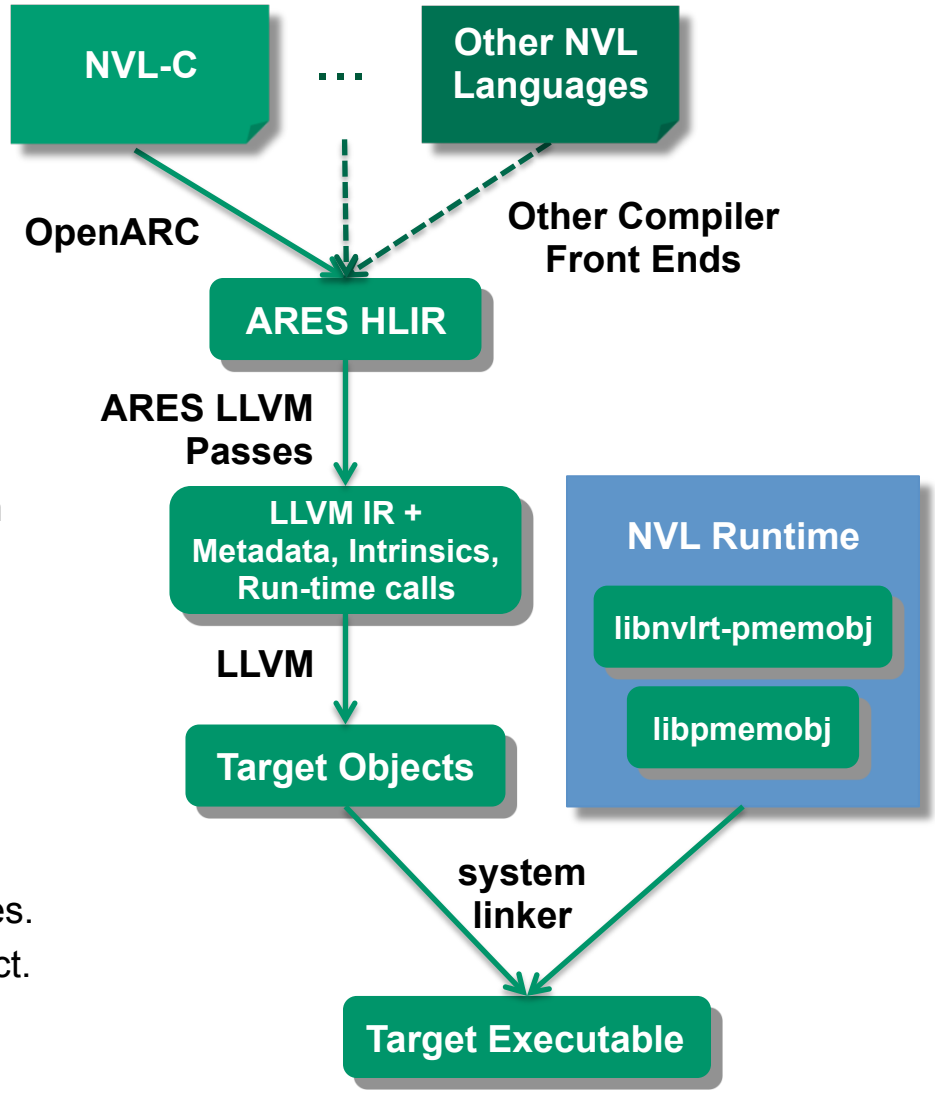




Design Quality vs. Level of Representation

NVL-C: Reliable Programming for NVM

- Problem
  - DRAM is fast and byte-addressable but power-hungry, expensive, and volatile.
  - HDD is cheap and persistent but slow.
  - HPC trends: DRAM-flop ratio shrinking, no node-local HDD.
  - Flash and future NVM tech will fill gaps but require new programming systems.
- Solution
  - NVL-C is a novel NVM programming system that extends C.
  - Currently uses Intel's pmemobj library for allocations and transactions.
  - Critical compiler components are implemented as reusable LLVM extensions.
  - Future work:
    - NVL-Fortran, NVL-C++, etc.
    - Target other persistent memory libraries.
    - Contribute components to LLVM project.



NVL-C: Reliable Programming for NVM

- Impact
  - Minimal, familiar, programming interface:
    - Minimal C language extensions.
    - App can still use DRAM.
  - Pointer safety:
    - Persistence creates new categories of pointer bugs.
    - Best to enforce pointer safety constraints at compile time rather than run time.
  - Transactions:
    - Prevent corruption of persistent memory in case of application or system failure.
  - Language extensions enable:
    - Compile-time safety constraints.
    - NVM-related compiler analyses and optimizations.
  - LLVM-based:
    - Core of compiler can be reused for other front ends and languages.
    - Can take advantage of LLVM ecosystem.

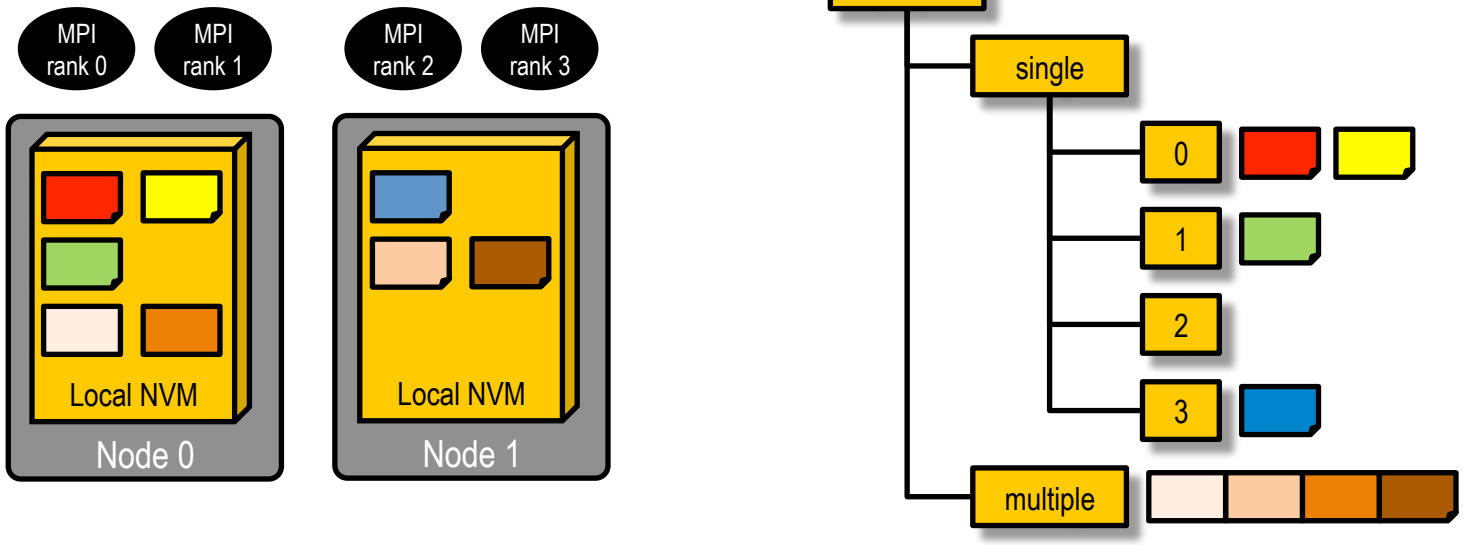
```
#include <nvl.h>
struct list {
  int value;
  nvl struct list *next;
};
void remove(int k) {
  nvl_heap_t *heap
    = nvl_open("foo.nvl");
  nvl struct list *a
    = nvl_get_root(heap, struct list);
  #pragma nvl atomic
  while (a->next != NULL) {
    if (a->next->value == k)
      a->next = a->next->next;
    else
      a = a->next;
  }
  nvl_close(heap);
}
```

PAPYRUS: Parallel Aggregate Persistent Storage

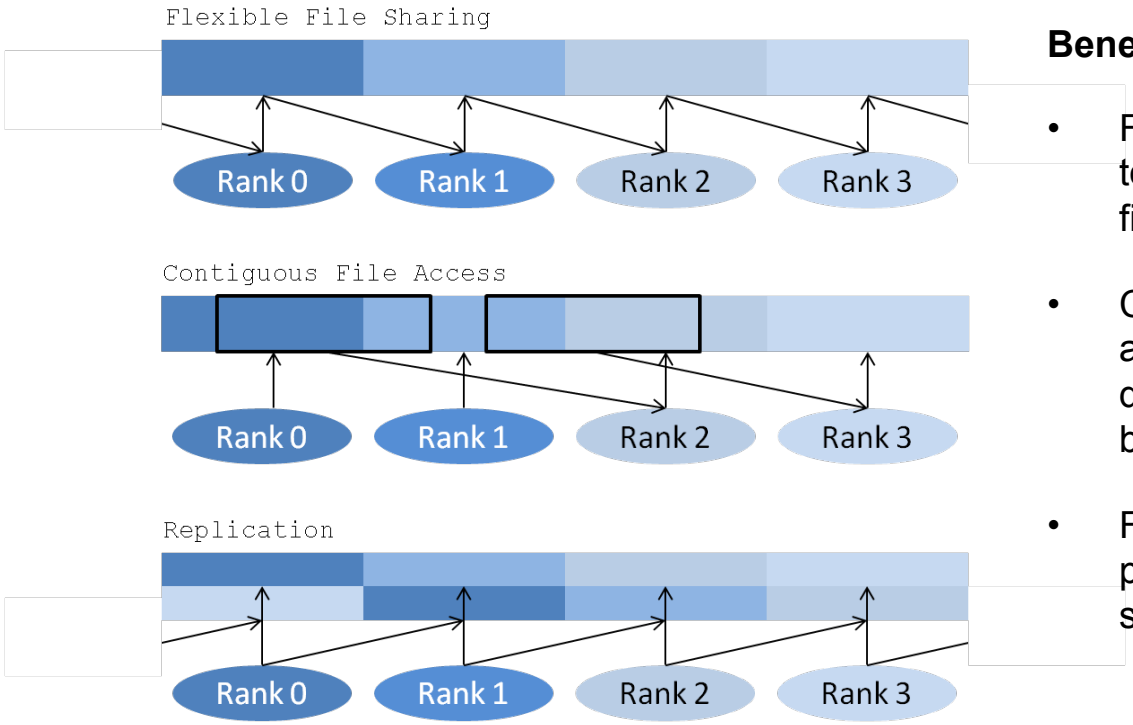
- Problem
  - Non-volatile memories (NVM) will play a more significant role in future HPC systems due to their advantages in terms of power, densities, performance and cost.
  - As these emerging memory devices become pervasive and diverse, however, these new memory hierarchies must be exposed to the software developers as first-class language entities to promote their efficiency and correctness.
- Solution
  - We propose Papyrus: a novel programming system that provides features for scalable, aggregate, persistent main memory.
  - Key contributions:
    - Design a novel, flexible, intuitive programming model for aggregate NVM that facilitates scalable, productive, safe application development on HPC systems with distributed NVMs.
    - Implement a prototype Papyrus as a library and runtime system.
    - Propose novel optimizations to support correct and efficient execution for an underlying memory systems including aggregated NVMs.
    - Evaluate Papyrus on a number of scalable applications to demonstrate its flexibility, ease of use, performance, and correctness.

PAPYRUS: Parallel Aggregate Persistent Storage (cont.)

- Papyrus Virtual File System
  - Lightweight, user-level file system, on top of which various advanced features, such as virtual global address space for aggregated NVMs, transaction, checkpointing, etc., will be added.
  - Provided as a library and runtime system to work with diverse types of parallel applications such as MPI applications.
    - All MPI ranks share the same virtual /papyrus file system.
    - All MPI ranks can create, read, write, and mmap to any files in the /papyrus file system by the Papyrus runtime routines.
    - A file in /papyrus/single/*N* is stored to the local NVM in the node where MPI rank *N* runs on
    - A file in /papyrus/multiple is striped across multiple local NVMs in the nodes

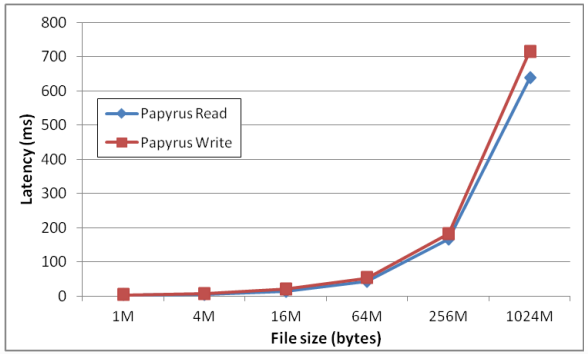


PAPYRUS: Parallel Aggregate Persistent Storage (cont.)



Benefits

- Flexible file access enables programmers to define locality and distribute portions of files
- Contiguous file access allows uniform access to distributed files. Programmers do not need to be aware of boundary between portions
- File replication can be defined by programmers to implement application-specific fault tolerant strategies.



Preliminary results

- 32 compute nodes connected with FDR Infiniband interconnect
- 32 MPI ranks write/read files to and from their neighbors

Session 3: OpenARC Internals (Advanced Topics)

- OpenARC Implementation
- OpenARC Internal Representation (IR)
  - OpenARC IR class hierarchy
  - Working on OpenARC IR
  - Example Transformations
- Built-in Cetus Passes for Program Analysis and Transformation

Implementation of OpenARC

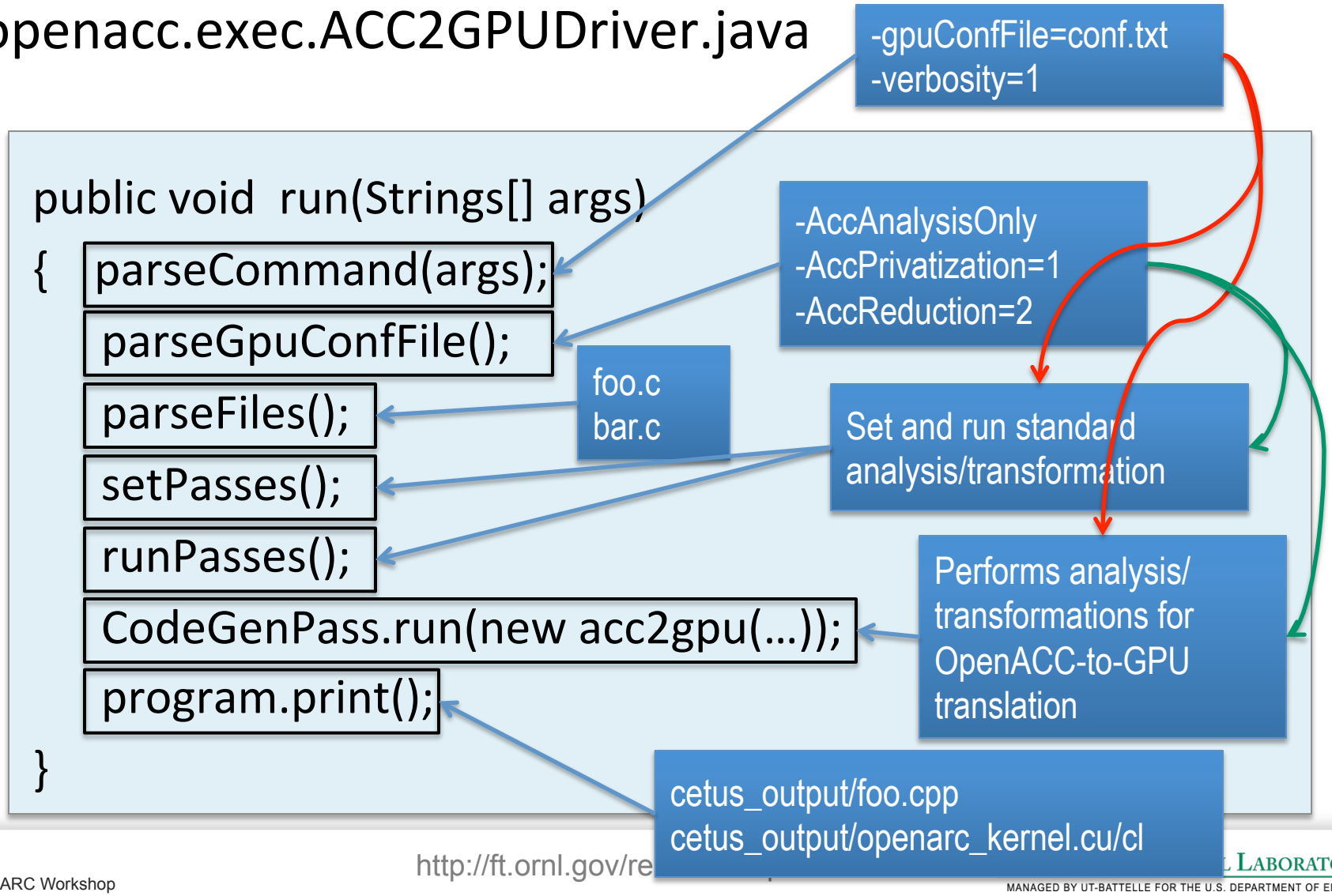
- Built on top of the Cetus compiler infrastructure
- What is Cetus?
  - Source-to-source C compiler written in Java and maintained at Purdue University (http://cetus.ecn.purdue.edu)
  - Provides an internal C parser (Antlr, 23K+ lines)
  - Intermediate representations (IR) (23K+ lines)
  - Compiler passes (45K+ lines and growing)
    - Contains various compile-time analysis/transformation passes, including privatization, reduction recognition, symbolic expression manipulators, induction variable substitution, etc.
- OpenARC adds 92K+ lines for OpenACC-to-Accelerator translation. (Modified JLLVM (44K+ lines) and LLVM are also included for LLVM support.)

OpenARC Source Structure

- OpenARC source
  - Cetus.base — C Parser
  - Cetus.hir, Openacc.hir — OpenARC IR
  - Cetus.analysis, Openacc.analysis; Cetus.transforms, Openacc.transforms — Analysis/ transformation passes
  - Cetus.exec, Openacc.exec — Main driver for parser and various passes
  - Cetus.codegen, Openacc.codegen — Custom codegen pass

OpenARC Main Driver

- openacc.exec.ACC2GPUDriver.java



http://ft.ornl.gov/re

OpenACC-to-GPU code generator

- openacc.codegen.acc2gpu.java

```
public void  start()
{   read command-line options and set parameters
    AccAnnotationParser()
    KernelCallingProcCloning()
    AccLoopDirectivePreprocessor()
    AccAnalysis()
    AccPrivatization/AccReduction()
    Various analyses to be added here.
    …
```

OpenACC-to-GPU code generator (2)

- openacc.codegen.acc2gpu.java

```
//latter part of public void  start()
    …
    KernelsSplitting()
    CompRegionConfAnalysis()
    CollapseTransformation()
    ACC2GPUTranslator()
    renameOutputFiles()
}
```

Depending on the target architecture, different derived classes are invoked.

GPUInitializer()
handleDeclareDirectives()
handleDataRegions()
convComputeRegions()
handleUpdateDirectives()
handleHostDataDirectives()
handleWaitDirectives()

Example of "Built-in" OpenARC Functionality

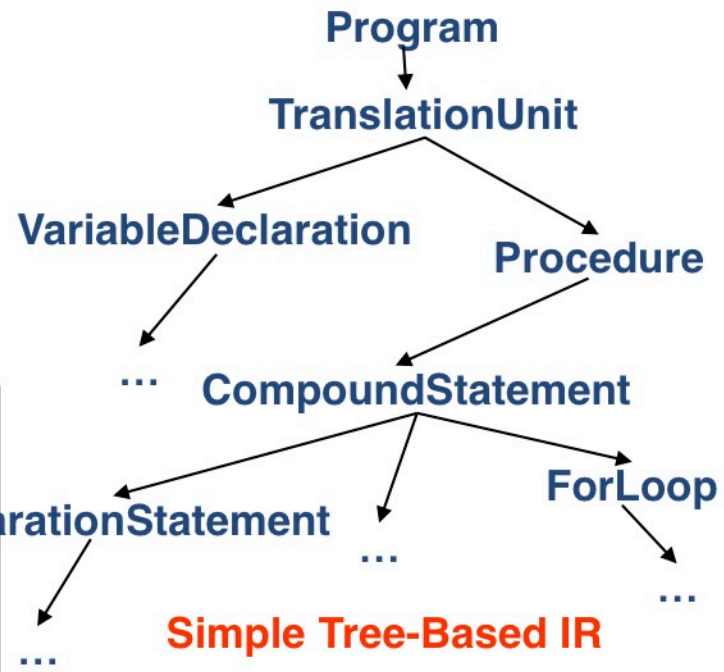
- OpenARC source-to-source translation – "parse and print"

Input

```
int temp;
int main(void) {
        int i,j,c,a[100];
        c = 2;
        for (i=0; i<100; i++) {
                a[i] = c*a[i]+(a[i]-1);
        }
}
```

Output

```
int temp;
int
main (void)
{
  int i, j, c, a[100];
  c = 2;
  for (i = 0; i < 100; i++)
    {
      a[i] = ((c * a[i]) + (a[i] - 1));
    }
}
```



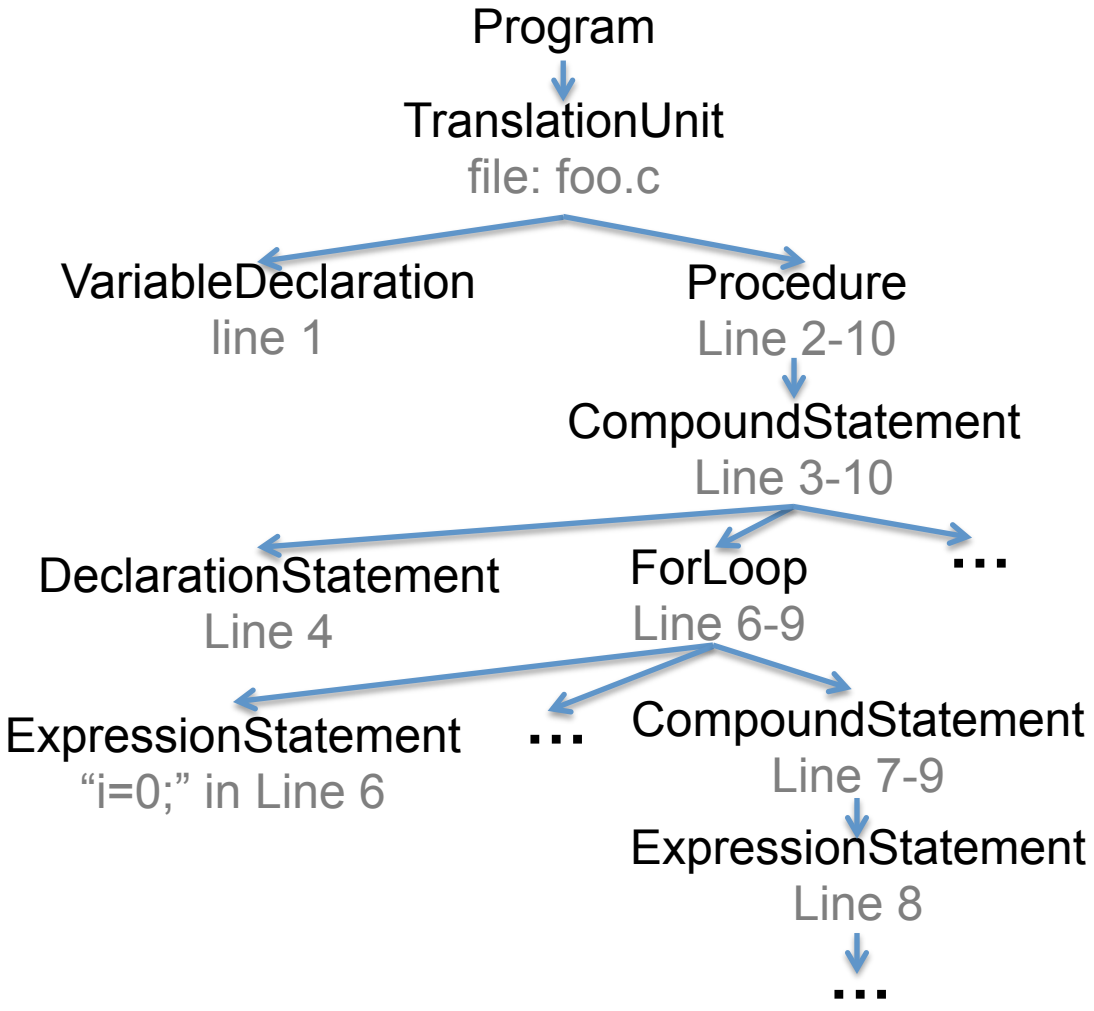
http://ft.ornl.gov/research/openarc

Session 3: OpenARC Internals (Advanced Topics)

- OpenARC Implementation
- OpenARC Internal Representation (IR)
  - OpenARC IR class hierarchy
  - Working on OpenARC IR
  - Example Transformations
- Built-in Cetus Passes for Program Analysis and Transformation

OpenARC High-Level Intermediate Representation Example

```
0  /* file: foo.c */
1    int c = 10;
2    int main(void)
3    {
4        int i, a[100], b[100];
5        #pragma acc parallel loop
6        for( i=0; i<100; i++ )
7        {
8            a[i] = c*b[i];
9        }
10       …
11   }
```



Major Parts of Class Hierarchy

Base Classes	Sub Classes
Program	
TranslationUnit	
Declaration	
	Procedure
	VariableDeclaration
	ClassDeclaration
	AnnotationDeclaration
	…
Statement	
	CompoundStatement
	ForLoop
	ExpressionStatement
	AnnotationStatement
	…
Expression	
	BinaryExpression
	FunctionCall
	…

Base Classes	Sub Classes
Declarator	
	VariableDeclarator
	ProcedureDeclarator
	…
Annotation	
	CommentAnnotation
	CodeAnnotation
	PragmaAnnotation
	CetusAnnotation
	OmpAnnotation
	ACCAnnotation
	ASPENAnnotation
	…
BinaryOperator	
	AssignmentOperator
	AccessOperator
Specifier	
	PointerSpecifier
	…

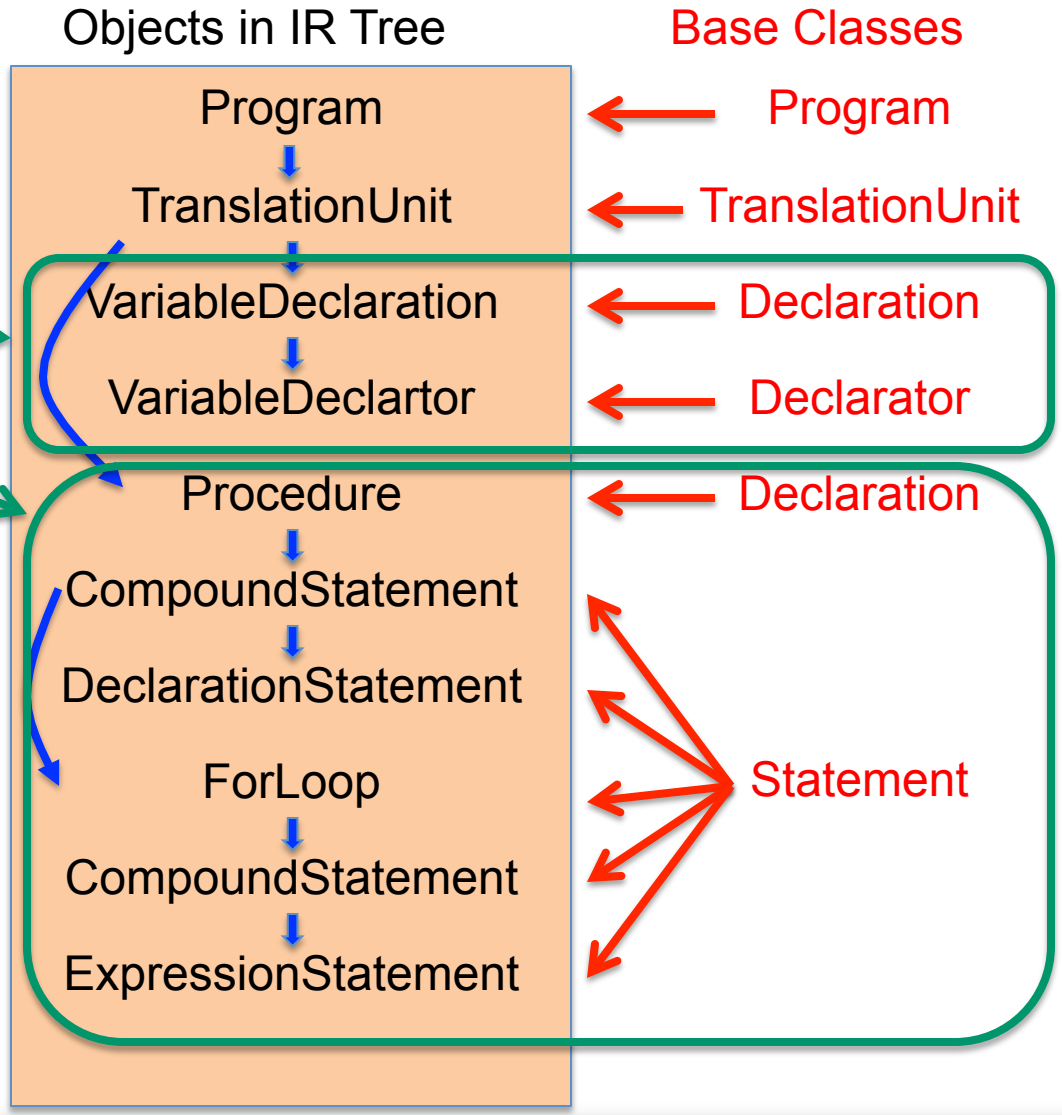
Traversable

Not traversable

OpenARC IR Tree versus Class Hierarchy

➔ IR/Syntax Tree
➔ Class Hierarchy

```
0  /* file: foo.c */
1    int c = 10;
2    int main(void)
3    {
4        int i, a[100], b[100];
5        #pragma acc parallel loop
6         for( i=0; i<100; i++ )
7         {
8            a[i] = c*b[i];
9         }
10       …
11   }
```

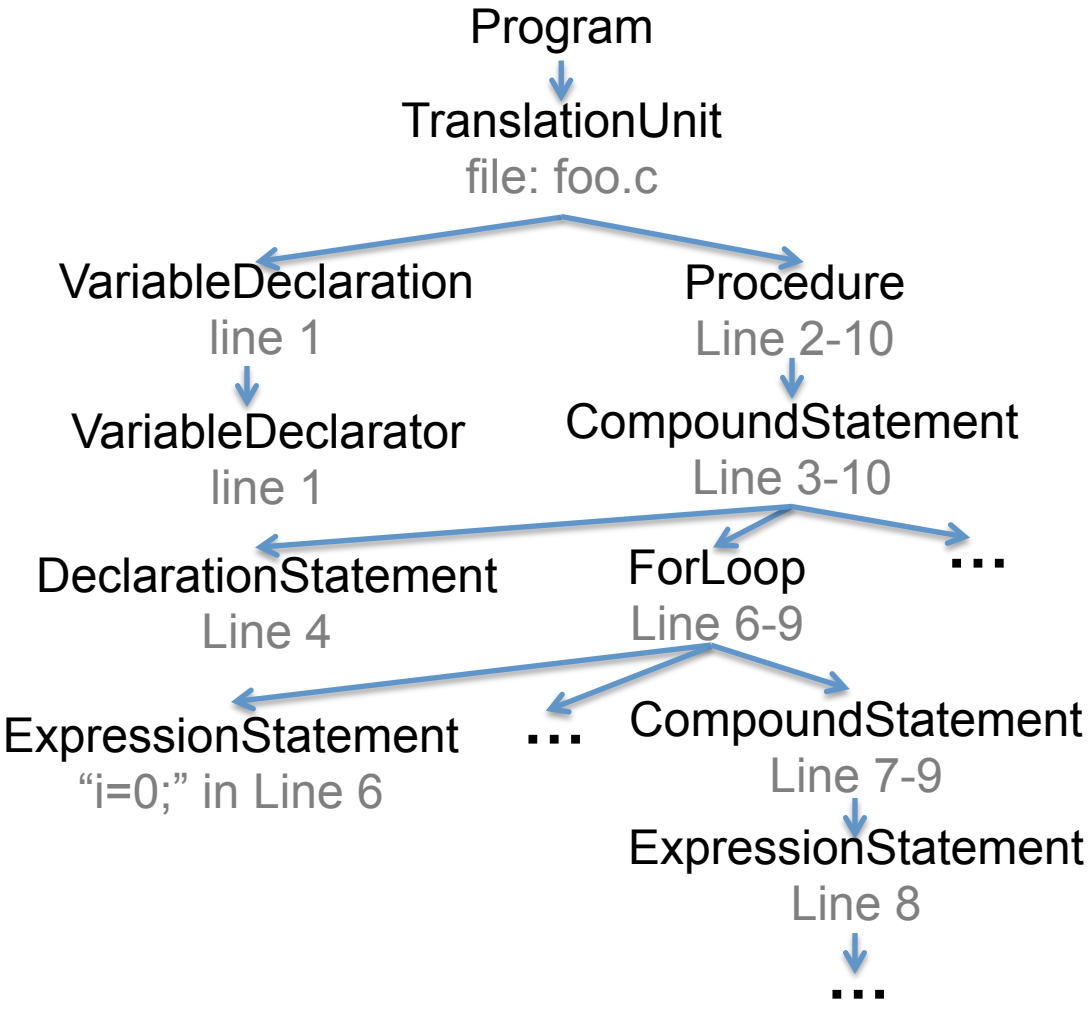


http://ft.ornl.gov/research/openarc

Working on OpenARC IR

- IR Iterators
  - DepthFirstIterator
  - BreadthFirstIterator
  - FlatIterator
  - PostOrderIterator

```
0  /* file: foo.c */
1    int c = 10;
2    int main(void)
3    {
4        int i, a[100], b[100];
5        #pragma acc parallel loop
6        for( i=0; i<100; i++ )
7        {
8           a[i] = c*b[i];
9        }
10       …
11   }
```



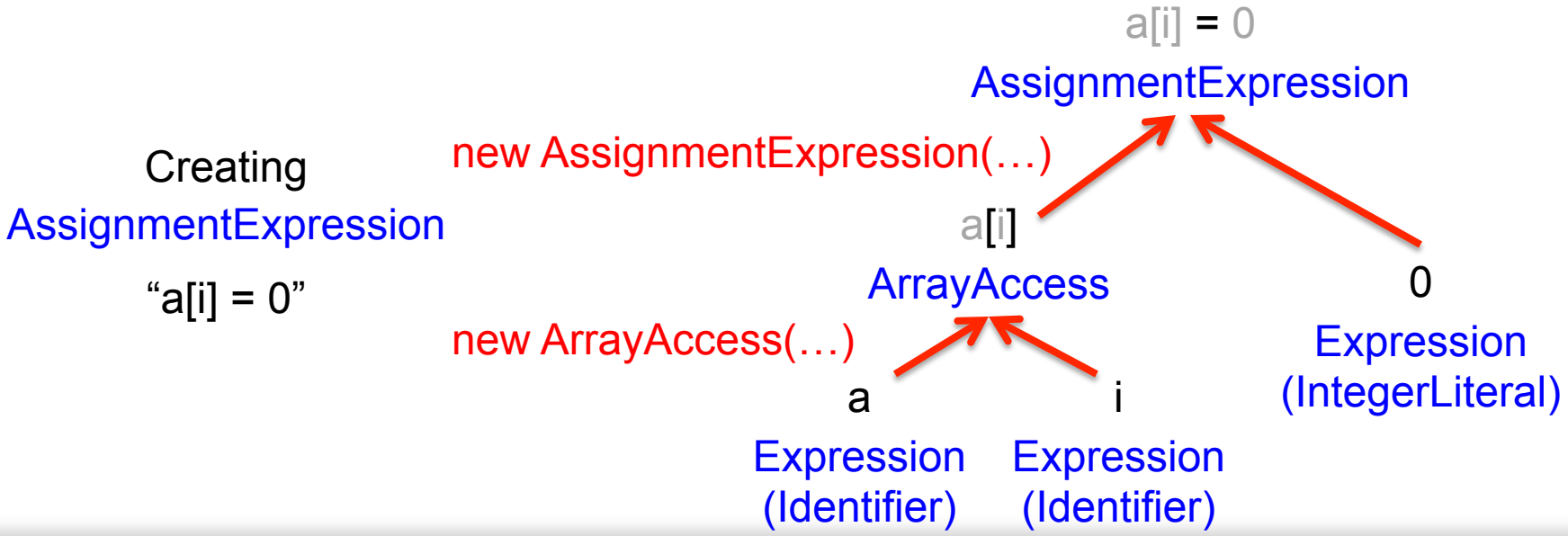
http://ft.ornl.gov/research/openarc

Other Tools assisting IR Traversing

- CFGraph: supports creation of statement-level control flow graphs.
  - Nodes in a CFGraph object contain information gathered from their corresponding OpenARC IR.
- CallGraph: create a static call graph for the program.
- And many others …

Creating OpenARC IR Objects

- Equivalent to building a tree from leaves
  - Some IRs are created first then populated.
    - E.g., CompoundStatement
  - IR constructors perform the task.
  - Use cloning if creating from an existing IR object.

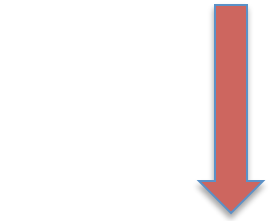


Modifying OpenARC IR Objects

- Equivalent to inserting/replacing a node in a tree
- IR methods handle most modifications.
- Other modifications are usually search/ replace.
  - Create a new expression or statement.
  - Traverse the IR to locate the position for the new node to be inserted or replaced with.
  - Perform insertion or replacement.

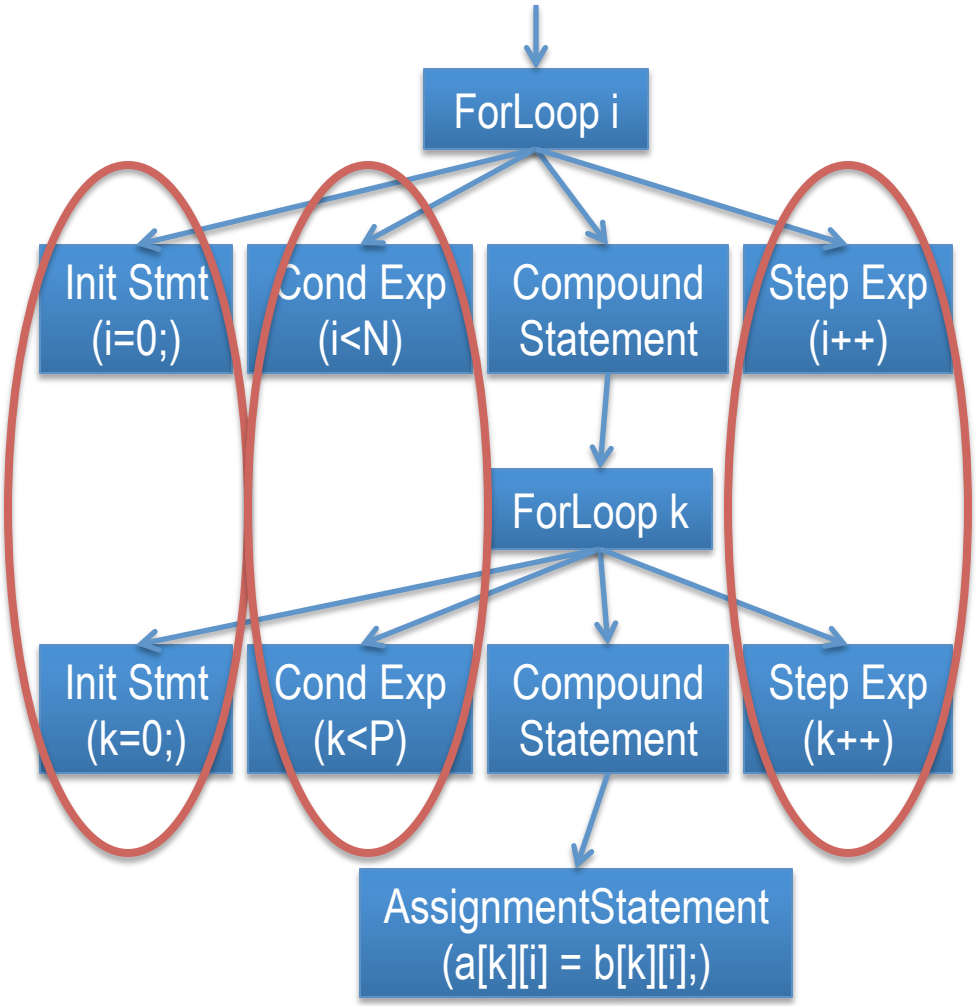
Example Transformation – Loop swap

```
for(i=0; i<N; i++) {
  for(k=0; k<P; k++) {
    a[k][i] = b[k][i];
  }
}
```



```
for(k=0; k<P; k++) {
  for(i=0; i<N; i++) {
    a[k][i] = b[k][i];
  }
}
```

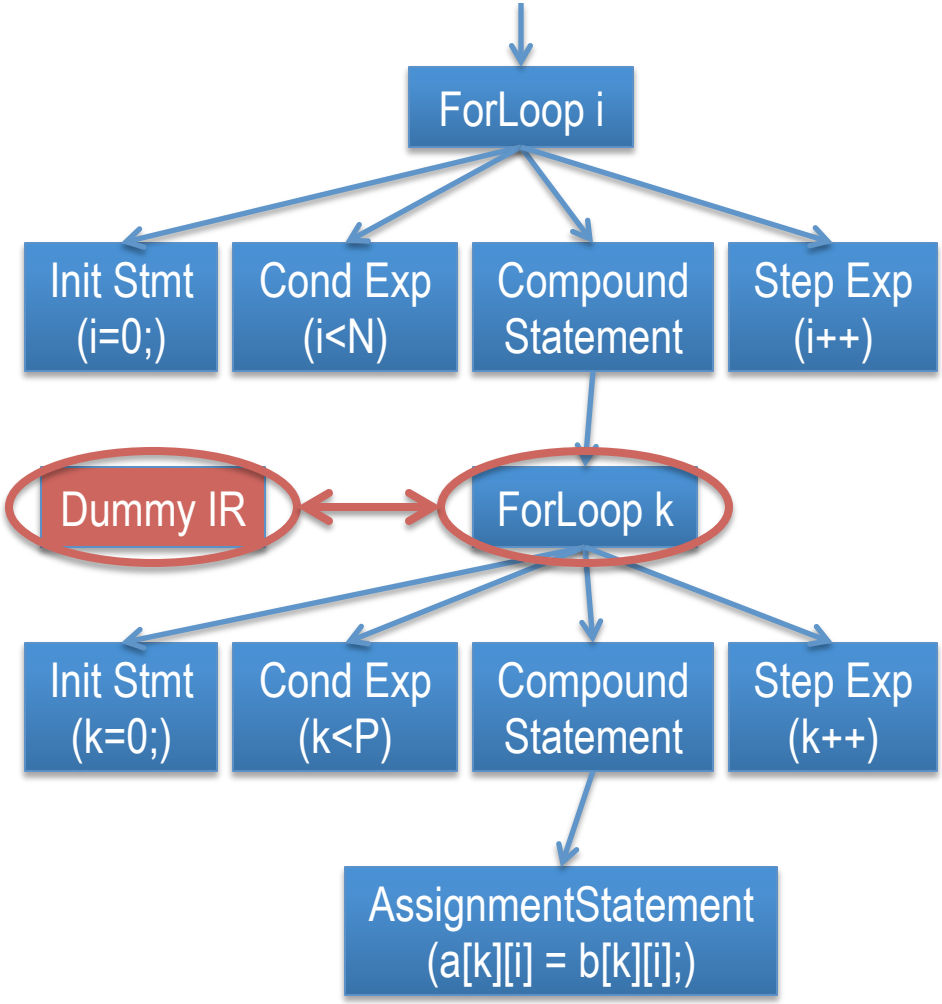
Swap each child except loop body



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

Example Transformation – Loop swap (cont.)

- Alternative way
  1) Swap k-loop with dummy IR.

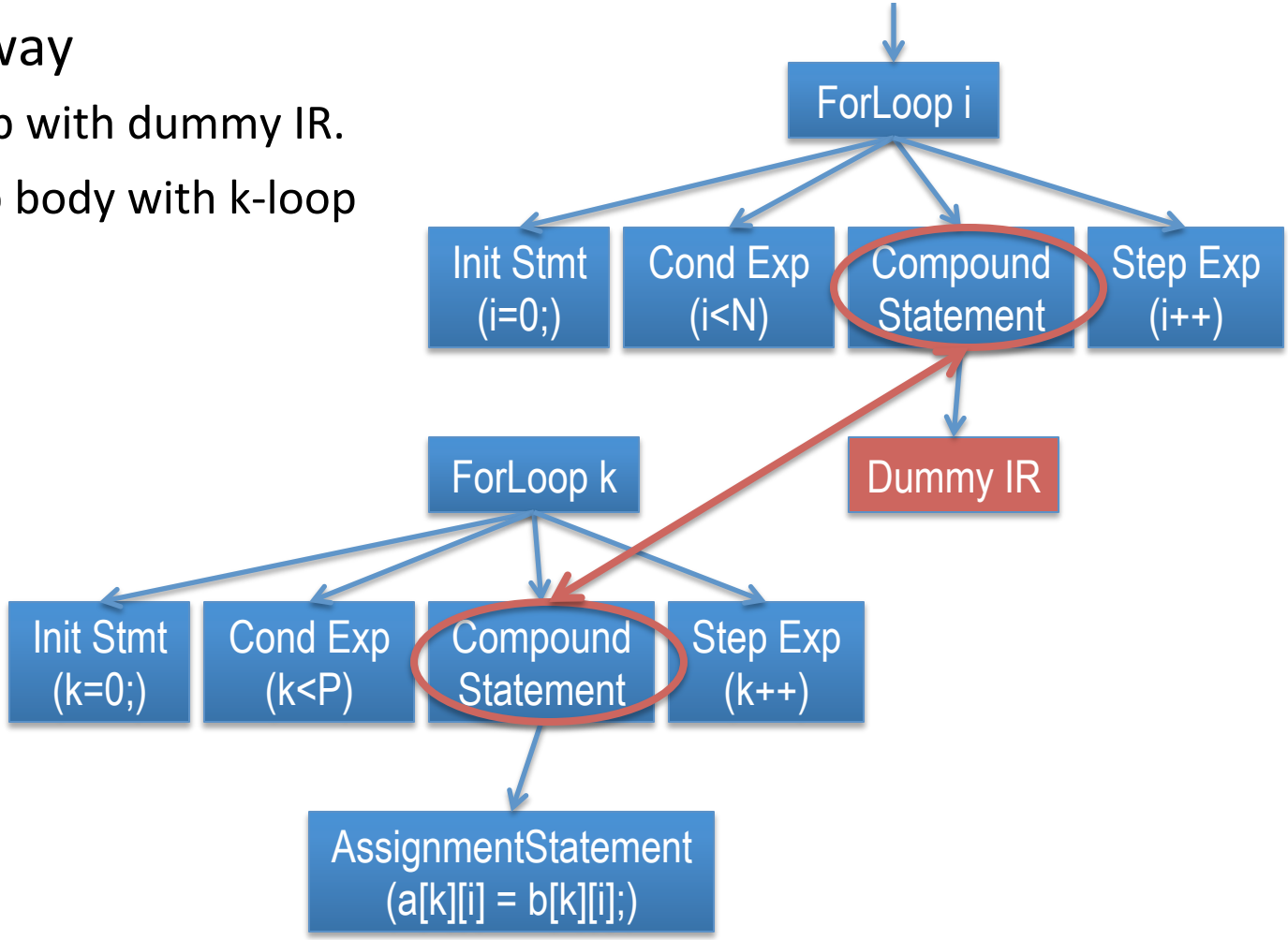


Example Transformation – Loop swap (cont.)

- Alternative way

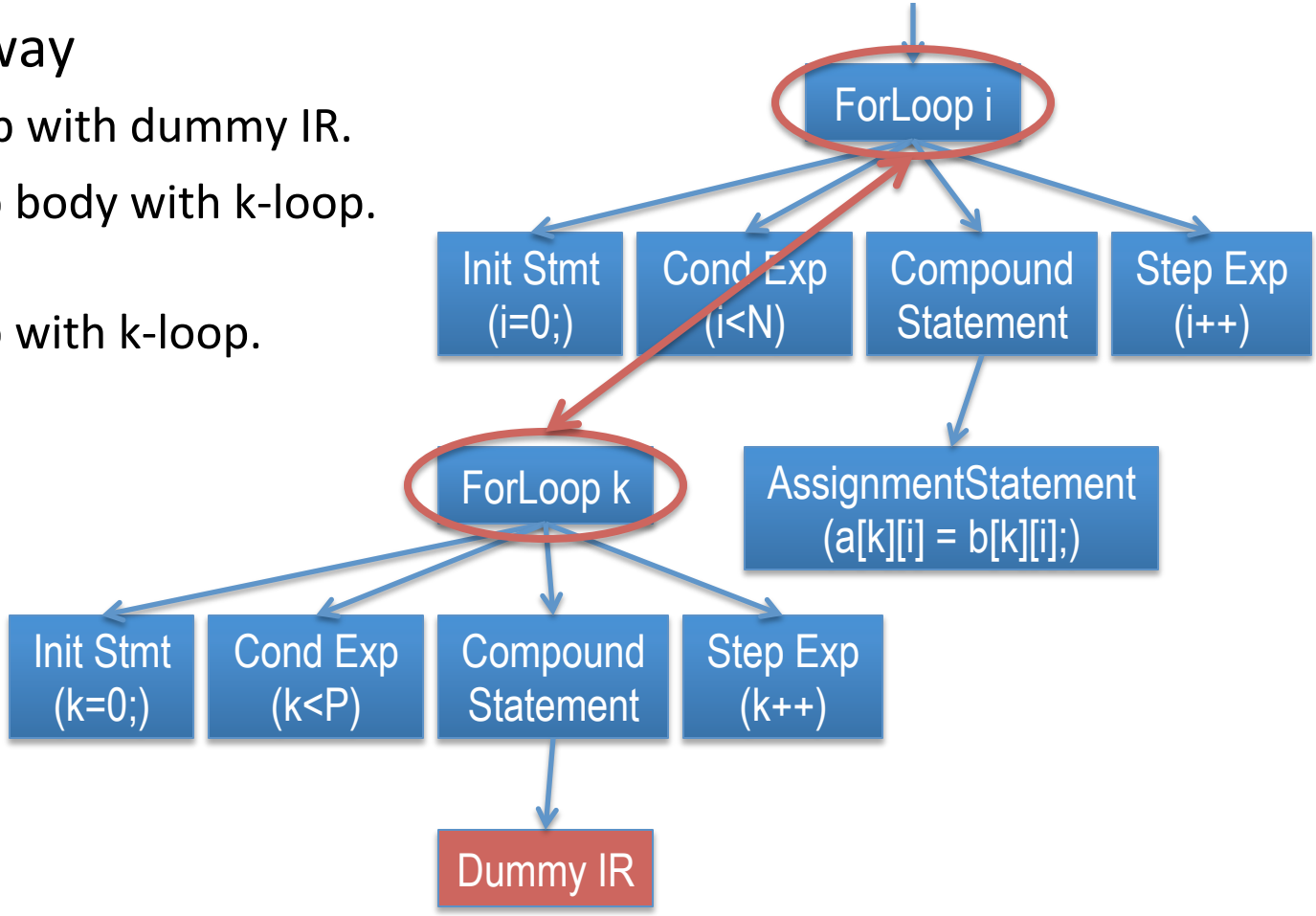
  1) Swap k-loop with dummy IR.

  2) Swap i-loop body with k-loop body.



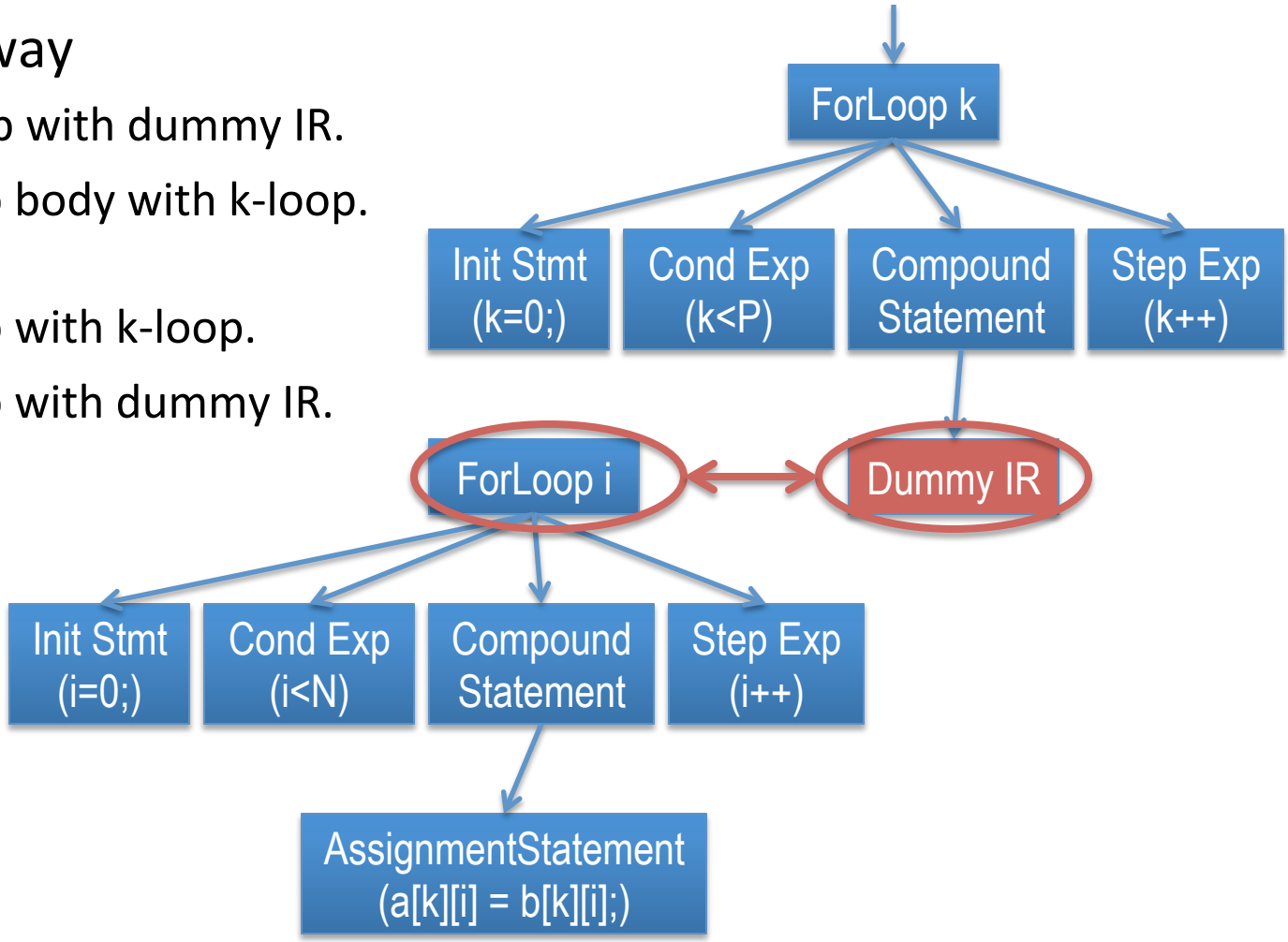
Example Transformation – Loop swap (cont.)

- Alternative way
  1) Swap k-loop with dummy IR.
  2) Swap i-loop body with k-loop. Body.
  3) Swap i-loop with k-loop.



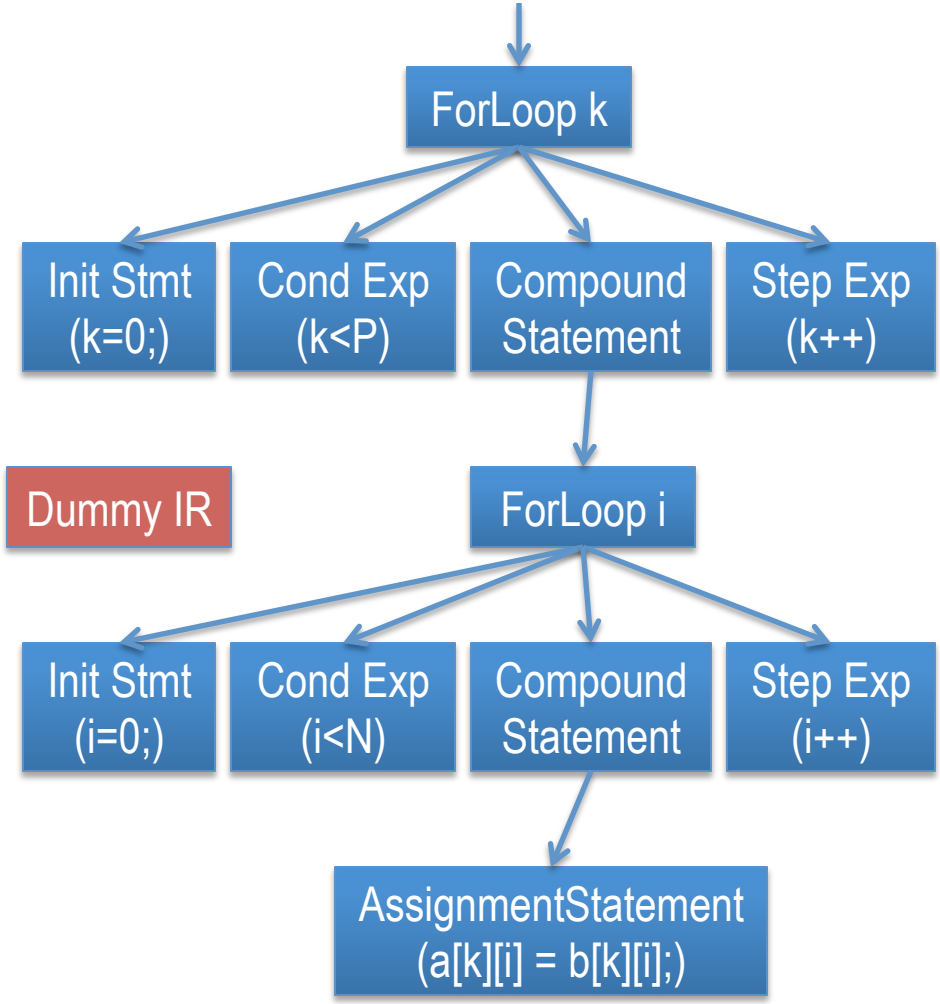
Example Transformation – Loop swap (cont.)

- Alternative way
  1) Swap k-loop with dummy IR.
  2) Swap i-loop body with k-loop. body.
  3) Swap i-loop with k-loop.
  4) Swap i-loop with dummy IR.



Example Transformation – Loop swap (cont.)

- Alternative way
  1) Swap k-loop with dummy IR.
  2) Swap i-loop body with k-loop. body.
  3) Swap i-loop with k-loop.
  4) Swap i-loop with dummy IR.

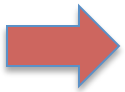


Example Transformation 2: Loop Unrolling

- Loop Unrollling by Factor of 2

1) Find loop index, I
2) Create expression i+=2
3) Replace step expression (i++) with i+=2

```
For (i=0; i<10; i++)
{
    fx[i] = fx[i] + x;
    fy[i] = fy[i] + y;
    fz[i] = fz[i] + z;
}
```



```
For (i=0; i<10; i+=2)
{
    fx[i] = fx[i] + x;
    fy[i] = fy[i] + y;
    fz[i] = fz[i] + z;
    fx[i+1] = fx[i+1] + x;
    fy[i+1] = fy[i+1] + y;
    fz[i+1] = fz[i+1] + z;
}
```

1) Copy the statements in the loop body.
2) Find loop index, i
3) Search & replace expression i with i+1
4) Insert the modified statements

Creating/Inserting Annotations

- Annotations are used in OpenARC for
  - Information exchange between passes - PragmaAnnotation
    - E.g., set of modified/used variables
  - Information for backend compilers – PragmaAnnotation
    - E.g., OpenMP pragmas, OpenACC pragmas
  - Information for code readers – CommentAnnotation
  - Inline arbitrary codes by the compiler – CodeAnnotation
- Annotation can be either
  - Stand-alone annotation or
  - Attached annotation: associated with a specific statement or a declaration

Creating/Inserting Annotations (cont.)

- Steps
  - Create an annotation with an appropriate type
    - CommentAnnotation, PragmaAnnotation, …
  - Locate the position for the annotation
    - Attached annotation: the associated statement/declaration
    - Stand-alone annotation: the reference statement/declaration
  - Insert the annotation
    - Attached annotation: insert directly to the statement/declaration
      - Use Annotatable interface, which supports annotate(), getAnnotation(), …
      - All Declaration/Statement IRs implement the Annotatable interface.
    - Stand-alone annotation: encapsulate in a dummy statement/ declaration and insert around the reference.
      - Use AnnotationStatement in a CompoundStatement or
      - AnnotationDeclaration in a TranslationUnit

Creating Custom PragmaAnnotations

- Option 1: use existing PragmaAnnotation
  - Annotation class extends HashMap, and thus it can keep arbitrary (key, value) pairs.
  - Useful for internal use among compiler passes
- Option2: derive a custom Annotation class derived from PragmaAnnotation.
  - May need to provide additional annotation parser to automatically attach the annotations to other IRs.
    - Default C parser creates all annotations as stand-alone type.
    - Additional parsers (OmpParser, ACCParser) are used to reason their semantics and attach them to related IRs accordingly.

Session 3: OpenARC Internals (Advanced Topics)

- OpenARC Implementation
- OpenARC Internal Representation (IR)
  - OpenARC IR class hierarchy
  - Working on OpenARC IR
  - Example Transformations
- Built-in Cetus Passes for Program Analysis and Transformation

Analysis Passes

- Data dependence analysis
  - Banerjee-Wolfe Test
  - Range Test
- Pointer alias analysis
- Symbolic range analysis
  - Generates a map from variables to their valid symbolic bounds at each program point.
- Array privatization
  - Compute privatizable scalars and arrays
- Reduction recognition
  - Detects reduction operations

Analysis Passes (cont.)

- Symbolic expression manipulators
  - Normalizes and simplifies expressions.
  - Examples
    - 1+2*a+4-a ➔ 5+a (folding) (a*2)/(8*c) ➔ a/(4*c) (division)
- Call Graph and CFG generators
  - CFG provided either at basic-block level or at statement level
- Basic use/def set computation for both scalars and array sections
- And many others…

Transformation Passes

- Induction variable substitution pass
  - Identifiers and substitues induction variables.
- Loop parallelizer
  - Depends on induction variable substitution, reduction recognition, and array privatization.
  - Performs loop dependence analysis.
  - Generates “parallel loop” annotations.
- Program normalizers
  - Single call, single declarator, single return normalizers.
- Loop outliner
  - Extract loops out into separate subroutines.
- And many others…


Thank You!
For more information, please refer to
the OpenARC website
http://ft.ornl.gov/research/openarc
OAK RIDGE
National Laboratory