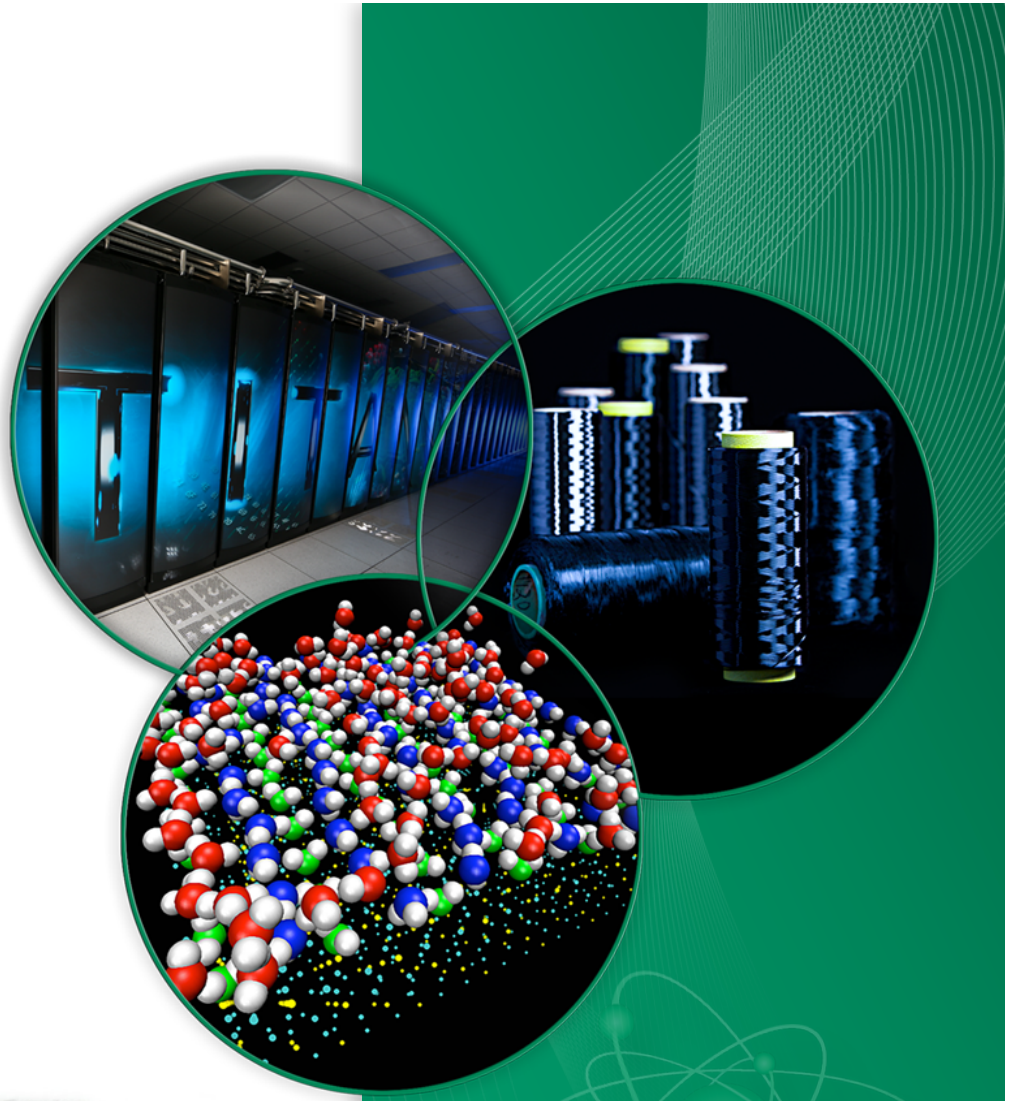
OpenARC: Open Accelerator Research Compiler (mini-workshop)



Seyong Lee

Future Technologies Group

Oak Ridge National Laboratory

lees2@ornl.gov

http://ft.ornl.gov

Agenda

- 9:00 – 10:20 am — Introduction to OpenARC
- 10:20 – 10:50 am — How to use OpenARC (User Guide)
- 10:50 – 11:00 am — Breaktime
- 11:00 – noon — OpenARC Internals (Advanced Topics)
- 1:00 – 2:00 pm — Introduction to OpenACC
- 2:00 – 4:00 pm — OpenACC and OpenARC Experience

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

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, 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 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.

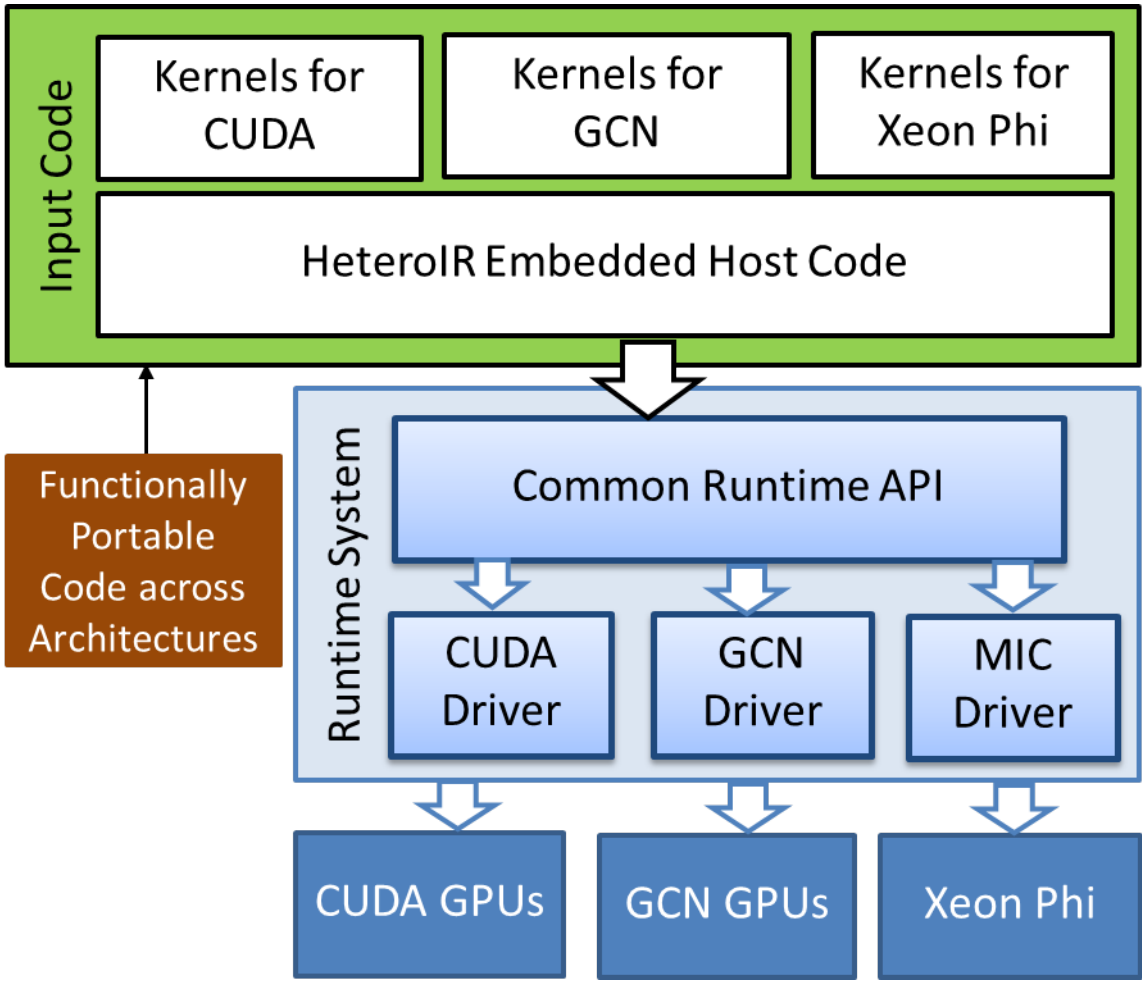
OpenARC Runtime API

- 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

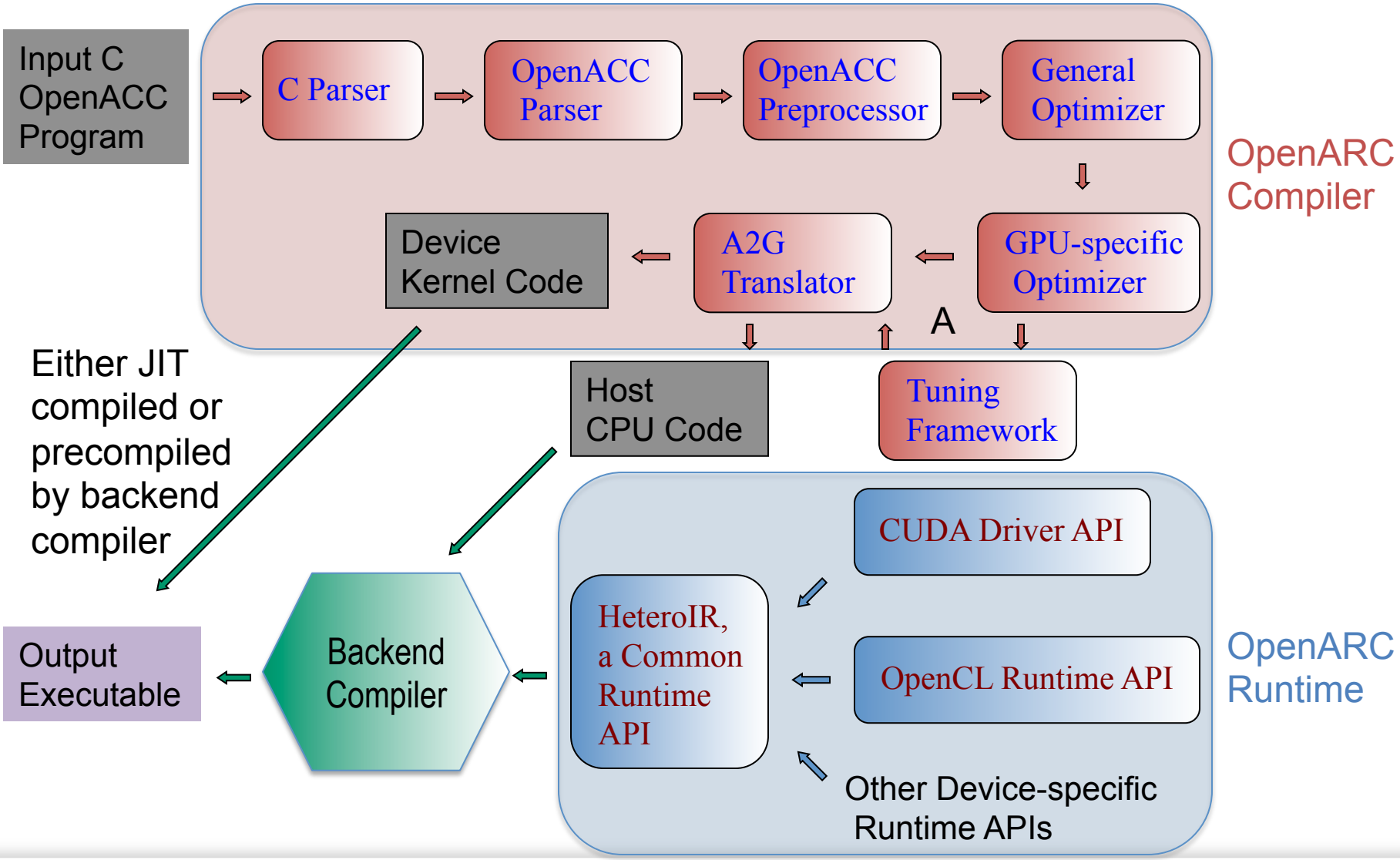
HI_set_device (device type, device number)	Set the device to use
HI_init ()	Initialize the previously selected device.
HI_reset ()	Deinitialize the device.
HI_malloc (pointer, size, queue)	Allocate memory on the device.
HI_memcpy (source, destination, size, queue)	Perform synchronous/asynchronous data 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.

HetroIR-based, OpenARC Runtime System



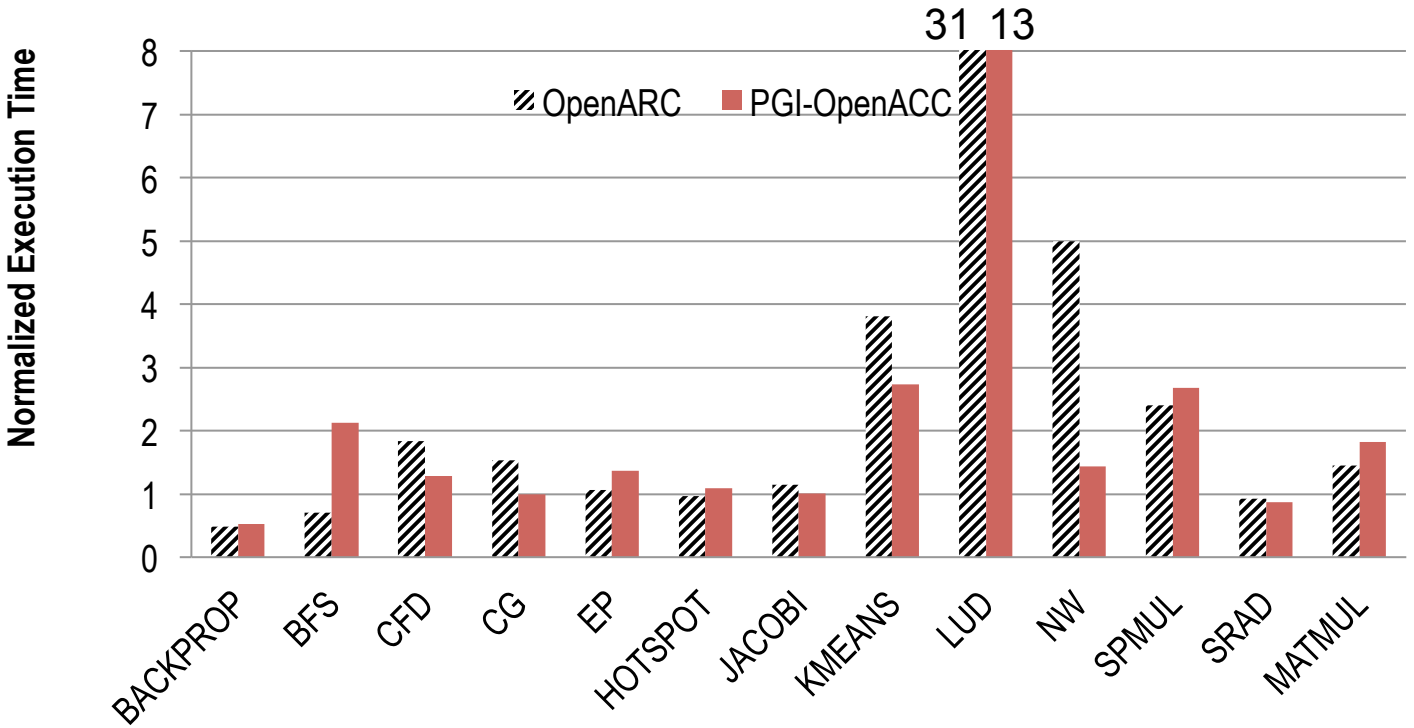
http://ft.ornl.gov/research/openarc

OpenARC System Architecture



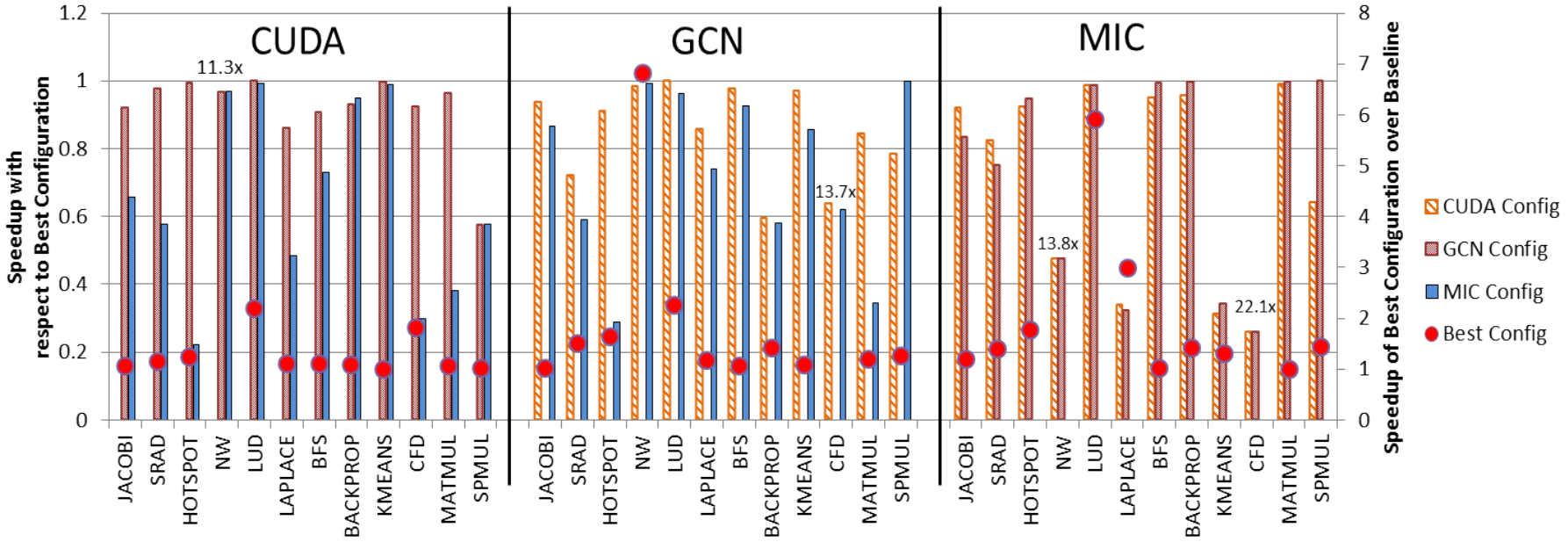
http://ft.ornl.gov/research/openarc

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.

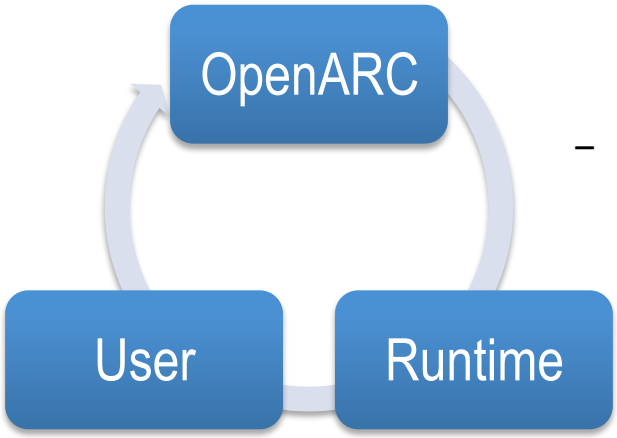
http://ft.ornl.gov/research/openarc

Research Agenda

- Research In Progress
  - Directive-based, interactive program debugging and optimization
  - Directive-based, application-driven cooperative CPU/GPU fault detection
  - autoASPEN: directive-based, automatic performance modeling
  - Directive-based, scalable heterogeneous computing
  - Compiler-driven, custom profiling system for directive programming
  - OpenACC-e: OpenACC extension to support accelerator-specific features at high-level

Research in Progress:
Directive-Based, Interactive Program Debugging and Optimization

- Problem
  - *Too much abstraction* in directive-based GPU programming!
    - Debuggability: difficult to diagnose logic errors and performance problems at the directive level
    - Performance Optimization: difficult to find where and how to optimize
- Solution
  - Directive-based, interactive GPU program debugging and optimization



Iteratively find where and how to fix/optimize

  - OpenARC compiler:
    - Generates runtime codes necessary for *GPU-kernel verification* and *memory-transfer verification and optimization*.
  - Runtime
    - Locate trouble-making kernels by comparing execution results at kernel granularity.
    - Trace the runtime status of CPU-GPU coherence to detect incorrect/missing/redundant memory transfers.

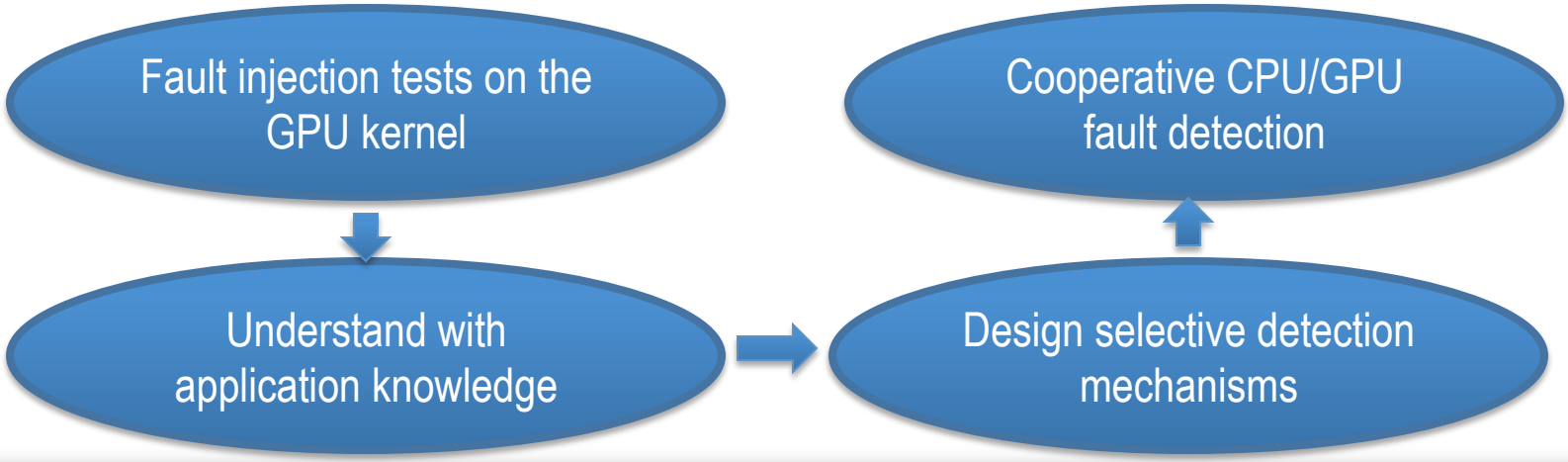
Users

  - Iteratively fix/optimize incorrect kernels/memory transfers based on the runtime feedback and apply to input program.

Research in Progress (2):

Directive-Based, Application-Driven Cooperative CPU/GPU Fault Detection

- Problem
  - *Limitations of the conventional resilience mechanisms*
    - No cooperation for data protection between CPU and GPU
    - Enforce a uniform protection, regardless of application semantics.
- Solution
  - Explore the viability of using an application-driven CPU/GPU cooperative method.
    - A user-level fault injection framework, which uses a directive-based approach to configure fault injection
    - Application-driven, cooperative CPU/GPU fault detection



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

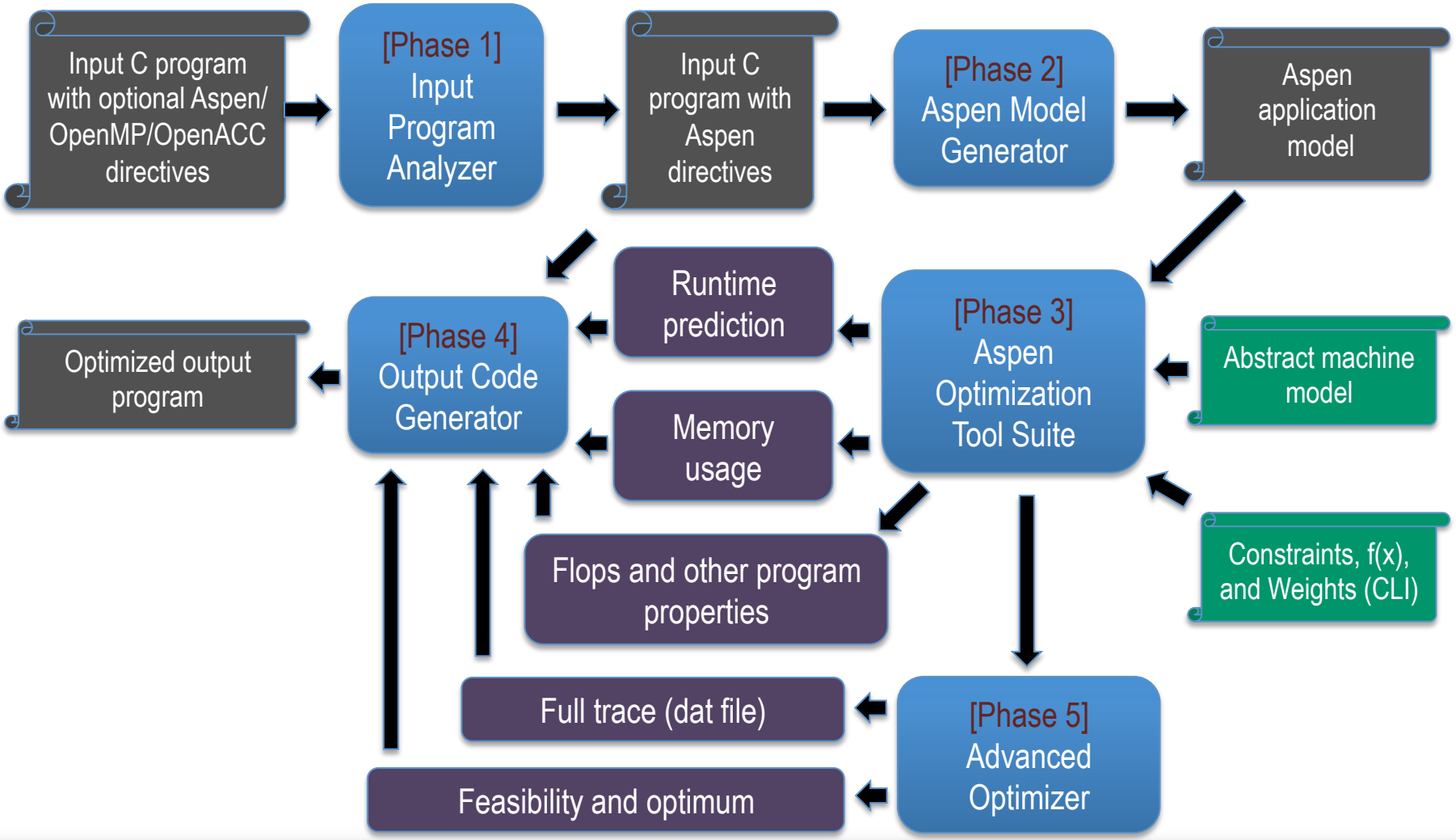
Research in Progress (3):

autoAspen: Directive-Based, Automatic Performance Modeling

- Problem
  - Porting programming systems (compilers, runtime, and libraries) across architectures is increasingly critical as architectures grow more complex and change quickly.
  - Many of performance parameters for the target architecture are implicitly embedded in the programming systems; hard to move a programming system for one configuration to another.
- Solution
  - Propose a method and system for performance portability in programming systems that separates performance parameters of each architecture from the implementation of the programming system.
    - Generate an explicit performance model from the application source code.
    - The parameters for the target architecture are convolved with the performance model to allow intelligent and portable decision by the programming system.

Research in Progress (3):

autoAspen: Process Diagram of The Overall Design Exploration



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 Applications
  - Performance Tuning in OpenARC
  - Directive-based, interactive program debugging and optimization
  - Directive-based, application-driven fault injection
  - autoASPEN: directive-based, automatic performance modeling

Getting Started

- Requirements
  - JAVA SE 6 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:
    - Use provided “build.sh” or “build.xml” files.

      $ build.sh jar //create jar file of OpenARC.
  - Build OpenARC runtime:
    - Go to ./openarcrt directory
    - Copy “make.header.sample” to “make.header”.
    - Update “make.header” as necessary.
    - Run “batchmake.bash” script

      $ batchmake.bash

Running OpenARC

- Environment Setup
  - Set OPENARC_ARCH to 0 for CUDA (default), 1 for OpenCL, or 2 for OpenCL for Xeon Phi.
  - 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

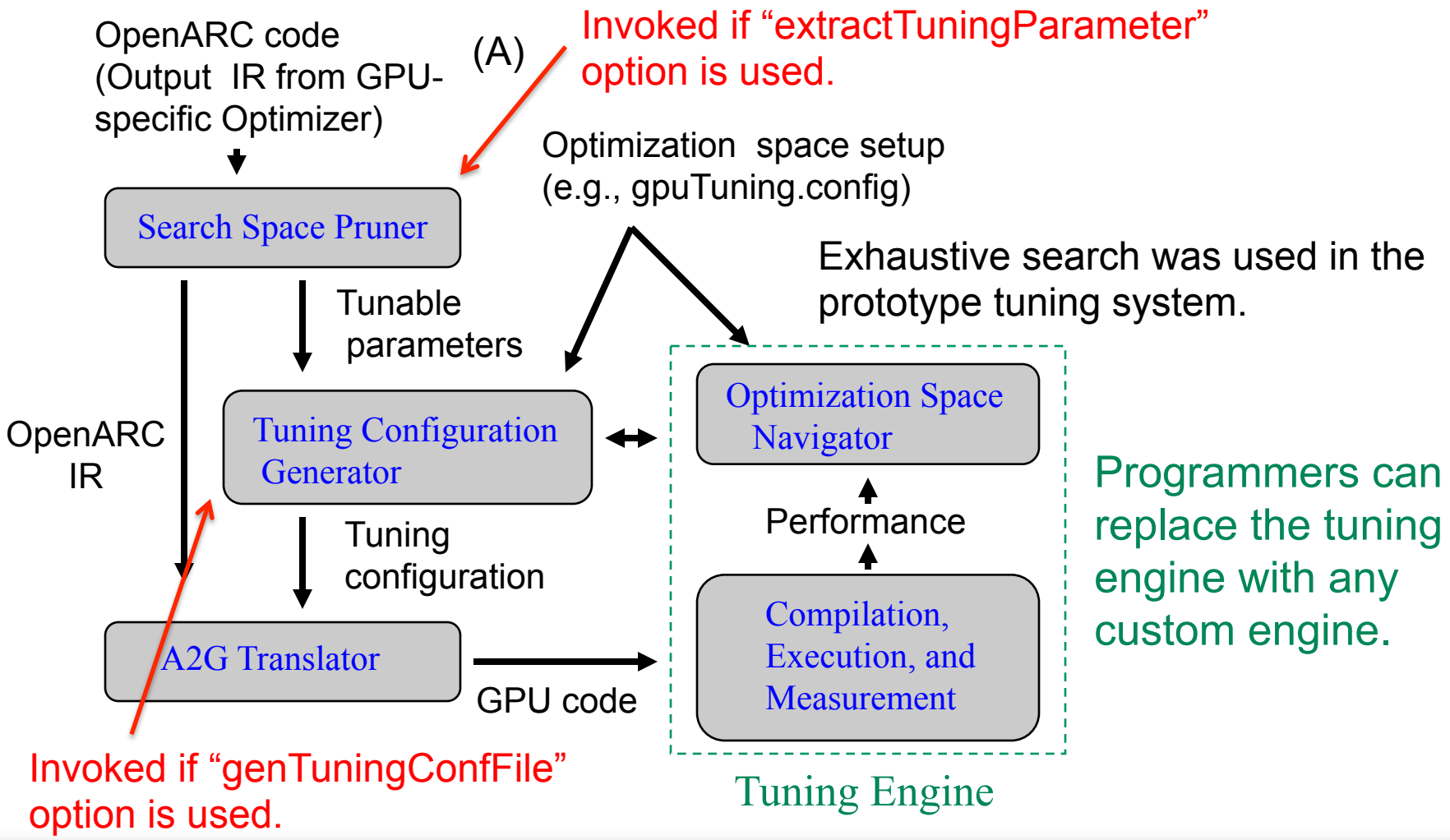
- The C parser supports C99 features only partially.
  - E.g., mixed declaration and code, C99 attributes, and C99 macros are only partially supported.
- GNU C preprocessor does not expand macros in pragma annotation.
  - To expand in annotations, use "#pragma acc #define …" directive.
- Struct member 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 Applications
  - Performance Tuning in OpenARC
  - Directive-based, interactive program debugging and optimization
  - Directive-based, application-driven fault injection
  - autoASPEN: directive-based, automatic performance modeling

OpenARC Application:

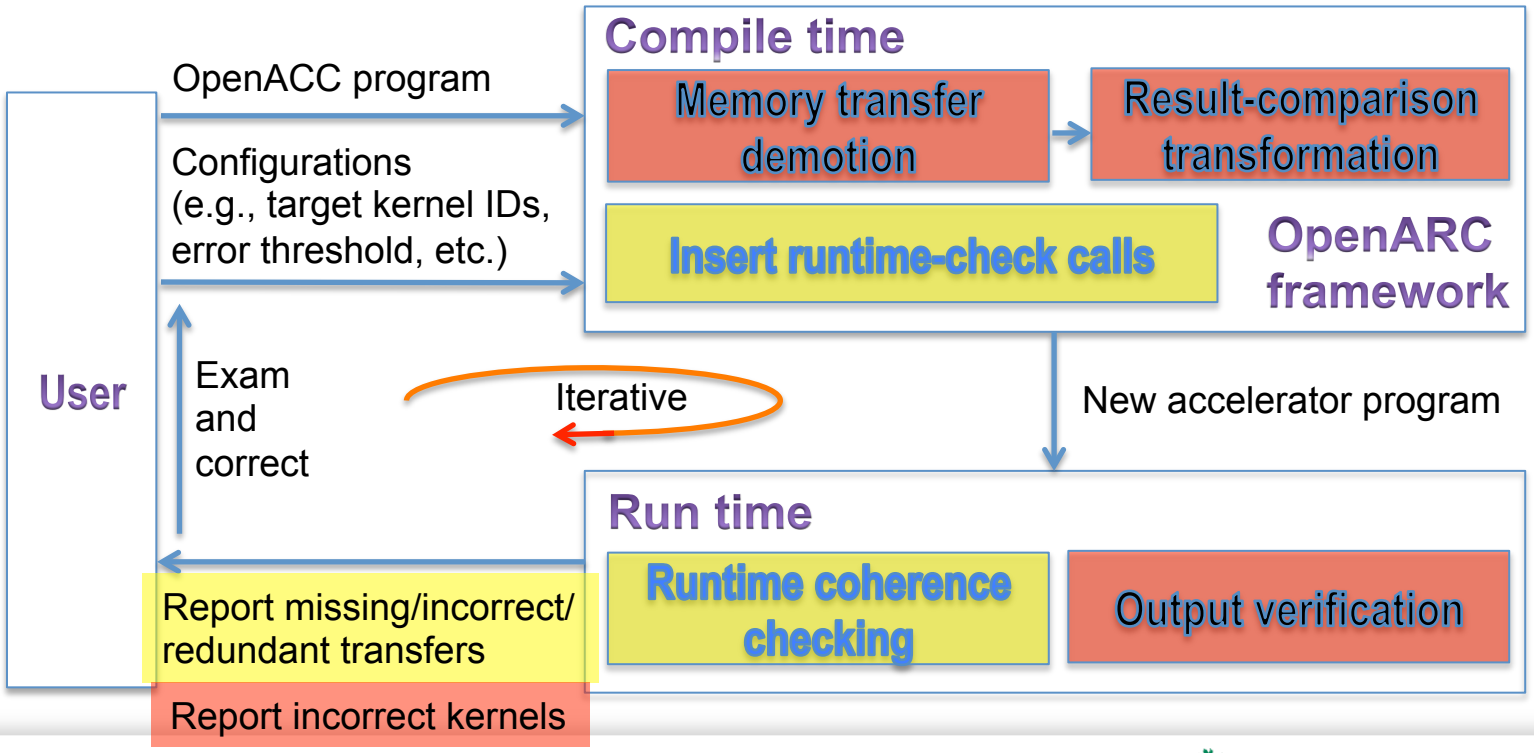
Performance Tuning using a built-in Tuning Framework in OpenARC



OpenARC Application 2:
Directive-Based Interactive Program Debugging and Optimization

Kernel verification (invoked if "programVerification=2" option is used.)

Memory-transfer verification and optimization (invoked if "programVerification=1")



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

OpenARC Application 3:
Directive-Based, Application-Driven Fault Injection

- We develop a user-level fault injection framework for general C programs.
  - Propose a set of user-directives and C library functions.
  - OpenARC automatically inserts fault injection codes according to the user-directives.
- We fully emulate the randomness of fault occurrences.
  - Temporal randomness
  - Spatial randomness
- We use a CPU/GPU cooperative approach for emulating random fault injection

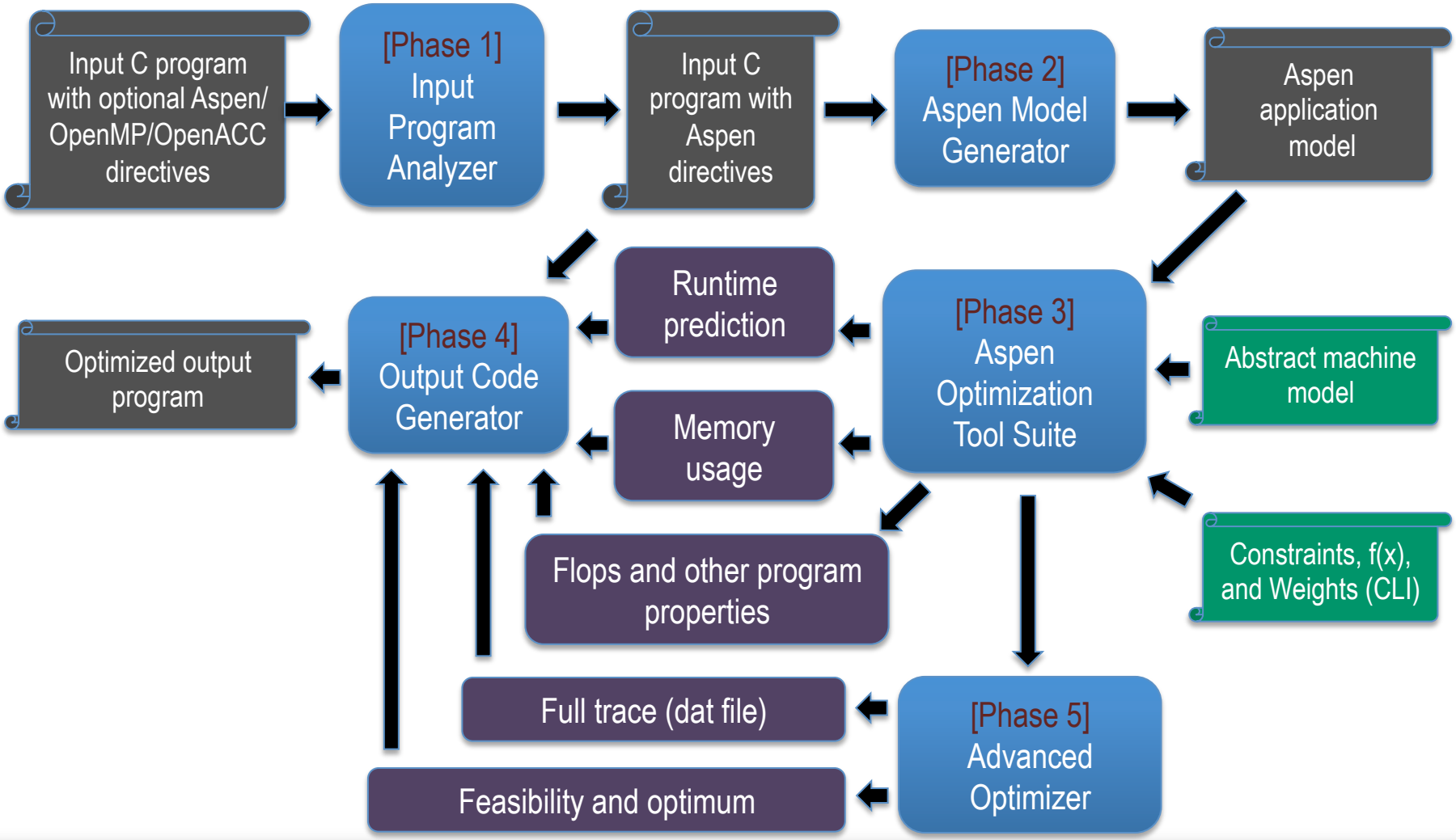
Directive-Based, Application-Driven Fault Injection (2)

- The directives controls fault injection configuration
  - Which program variable at which target region?
  - The condition to enable faults?
  - Number of fault injections and number of faulty bits
- Directive Format (partial list)

```
#pragma acc resilience [clause[[,] clause]...]

   structured block

// where clause is one of the following: ftregion, ftcond(condition), ftdata(list),
num_faults(exp), num_ftbits(exp), ftthread(exp)

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

// where clause is one of the following: ftdata(list), ftthread(exp)
```

OpenARC Application 4:

autoAspen: Directive-Based, Automatic Performance Modeling



autoAspen: Directive-Based Automatic Performance Modeling (2)

- The proposed Aspen directives are primarily used by OpenARC to annotate important properties of the target program, such as computations, memory accesses, communications, control flows, etc.
- Aspen directives allow users to control how OpenARC analyzes and generates a performance model in an interactive manner.
- Aspen Modeling Directive Format
  - #pragma aspen enter modelregion [label(name)]
  - #pragma aspen exit modelregion [label(name)]
  - #pragma aspen modelregion [label(name)]
  - #pragma aspen declare [clause [[,] clause]…]
  - #pragma aspen control [clause [[,] clause]…]

Breaktime

- It's time for a break!
  - We will resume at 11:00AM

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

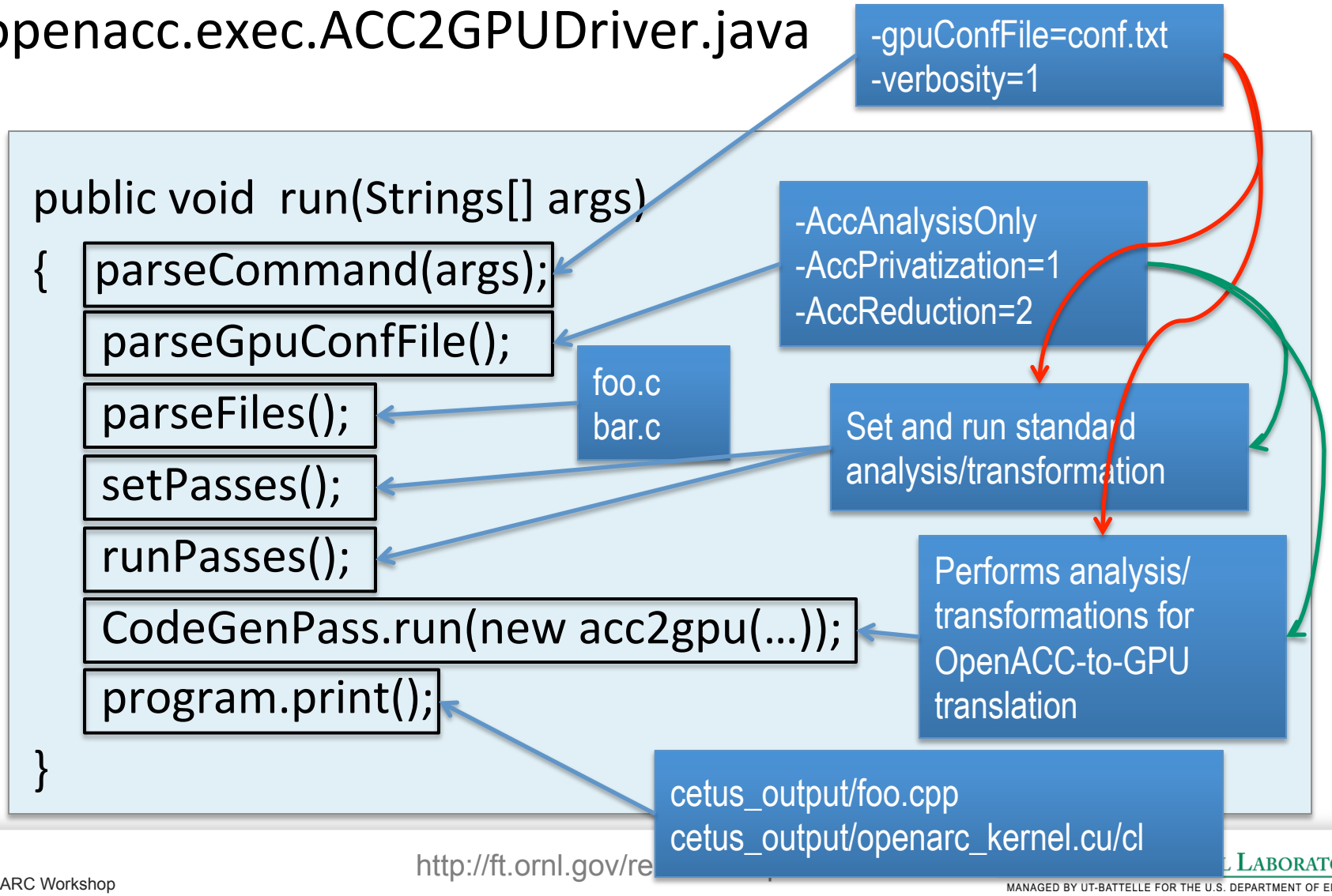
- Built on top of the Cetus compiler infrastructure
- What is Cetus?
  - Source-to-source C compiler written in Java and maintained at Purdue University (http://cetus.ecn.purdue.edu)
  - Provides an internal C parser (Antlr, 23K+ lines)
  - Intermediate representations (IR) (23K+ lines)
  - Compiler 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 65K+ lines for OpenACC-to-Accelerator translation.

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



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

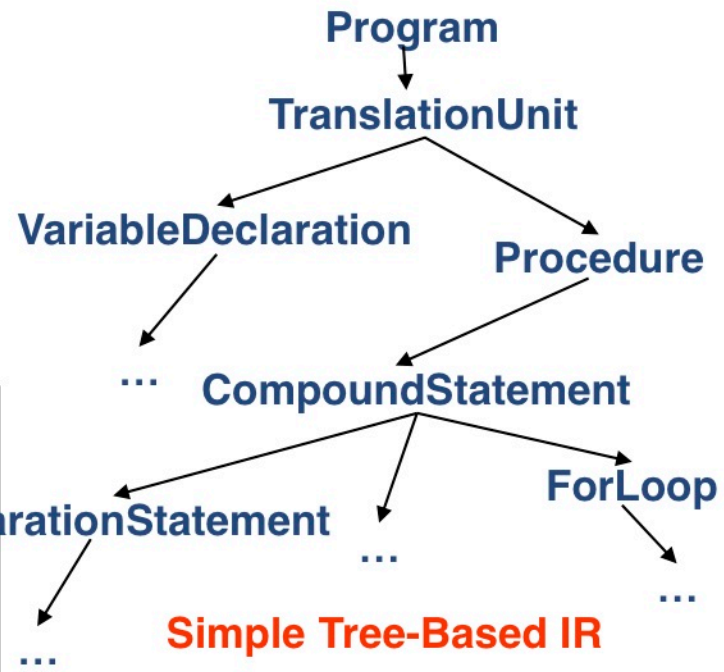
- OpenARC source-to-source translation – "parse and print"

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));
    }
}
```



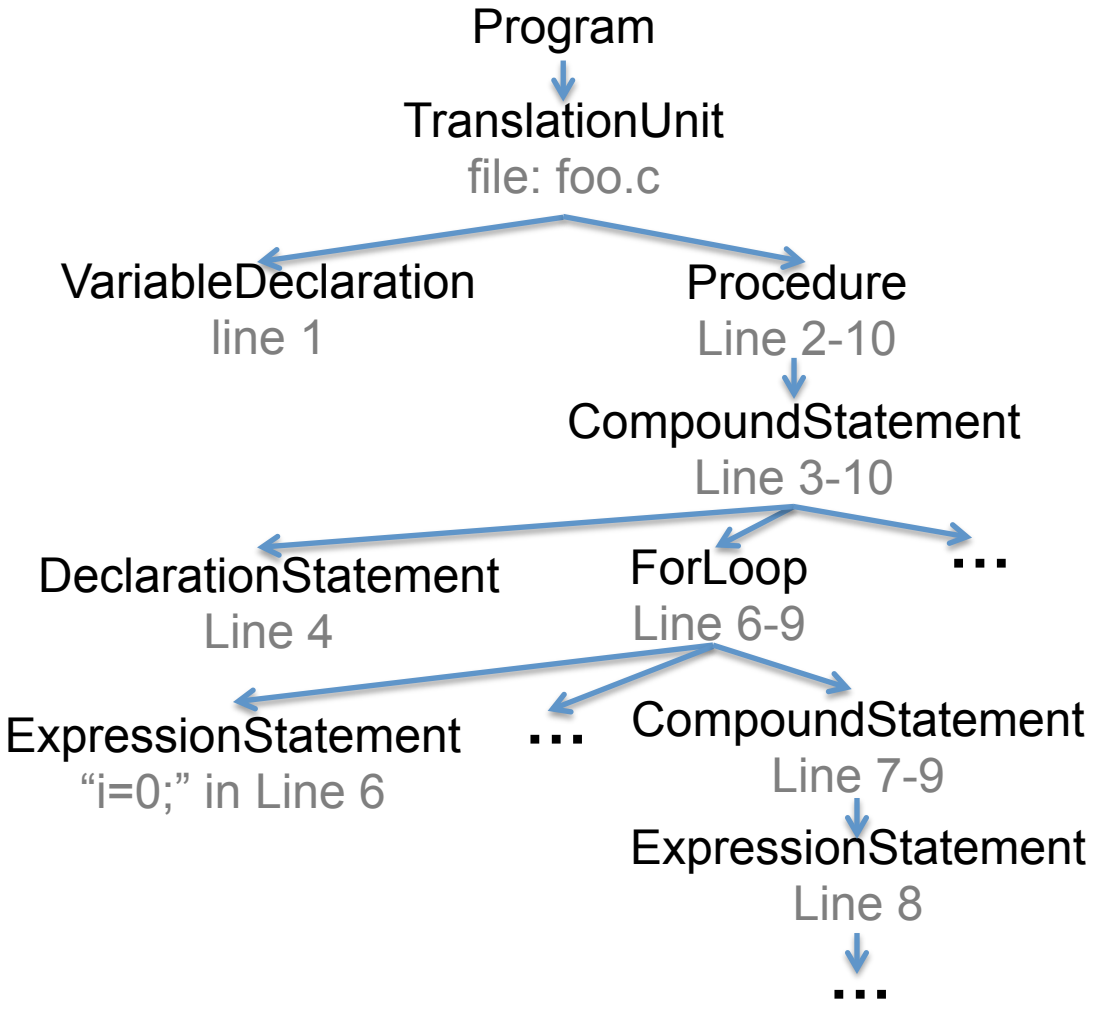
http://ft.ornl.gov/research/openarc

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   }
```



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

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
	…

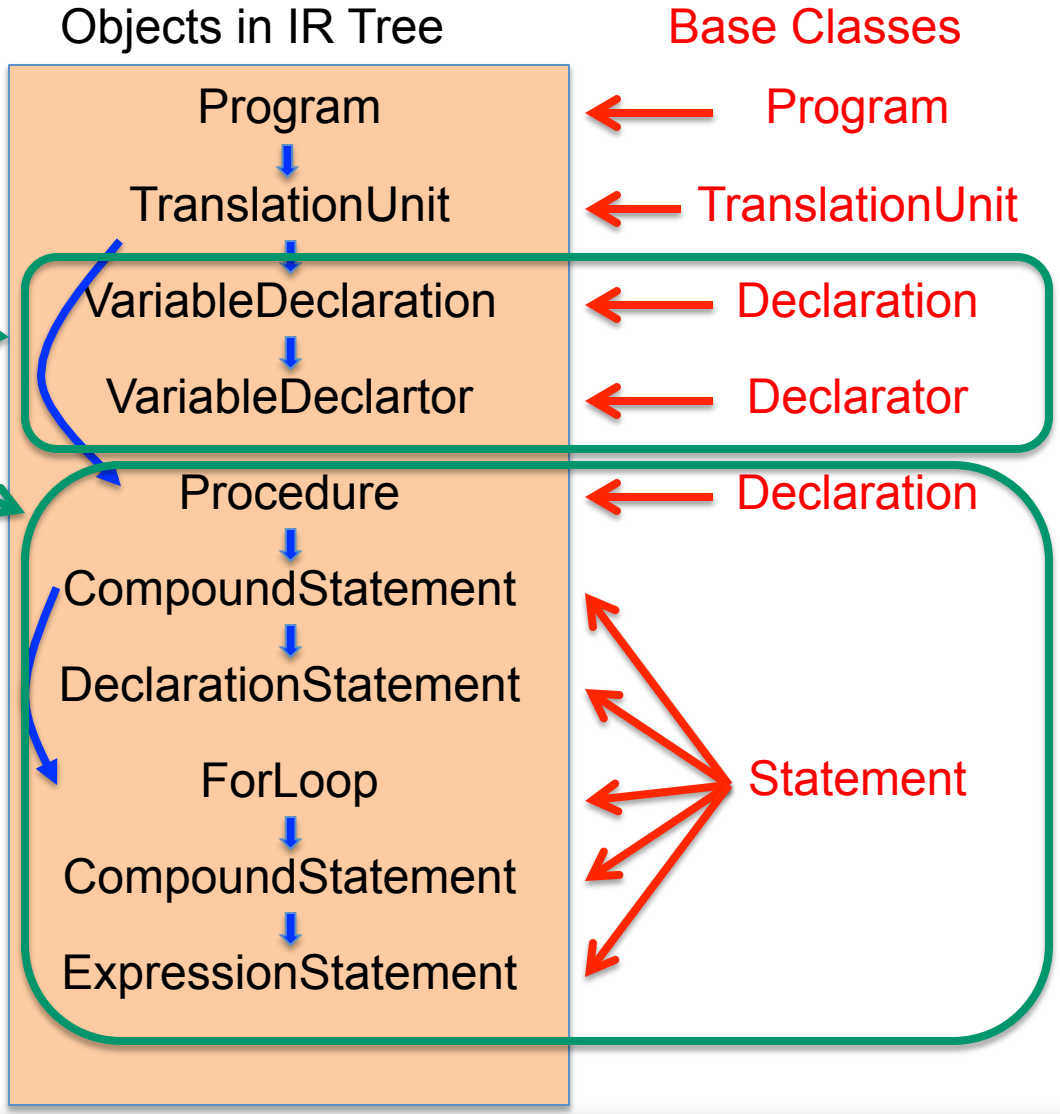
Traversable

Not traversable

OpenARC IR Tree versus Class Hierarchy

➔ IR/Syntax Tree
➔ Class Hierarchy

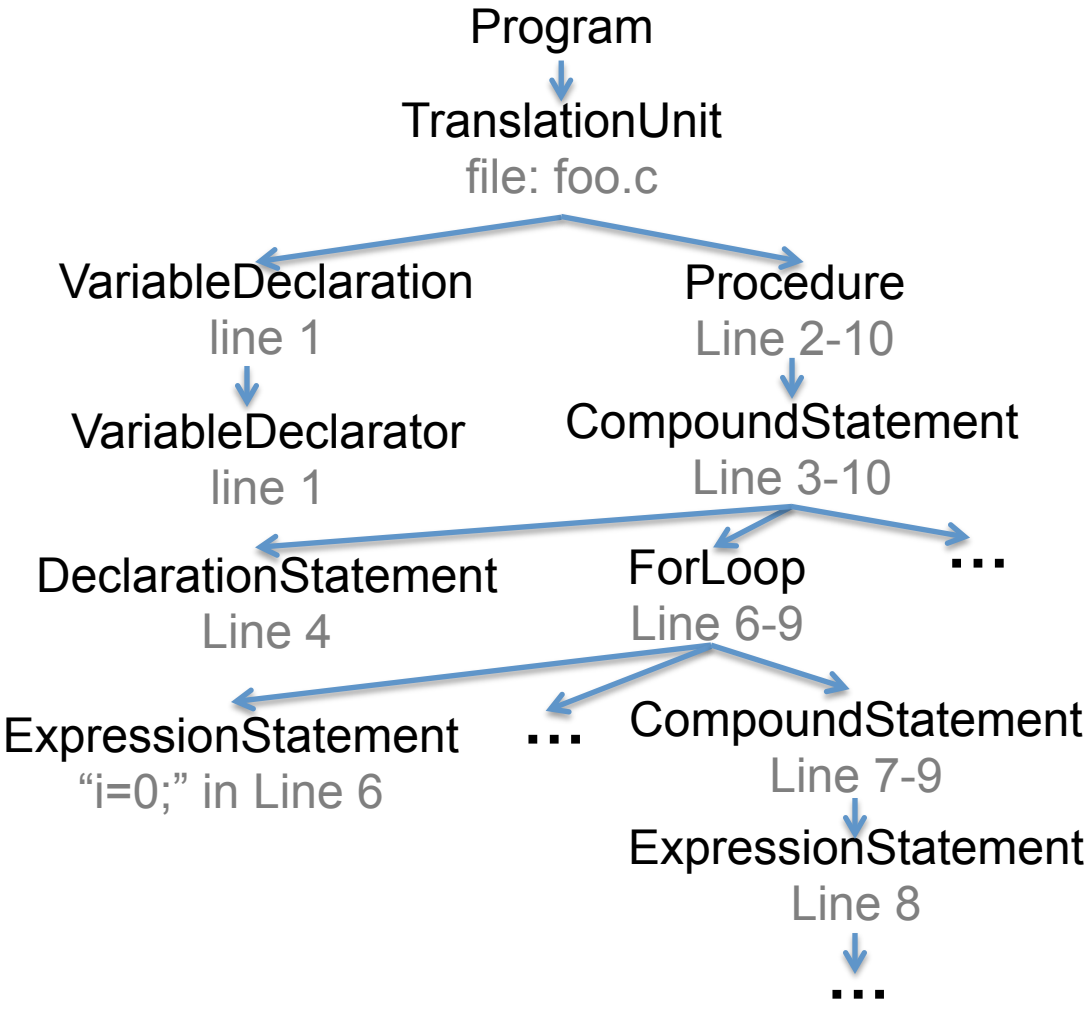
```
0  /* file: foo.c */
1    int c = 10;
2    int main(void)
3    {
4        int i, a[100], b[100];
5        #pragma acc parallel loop
6        for( i=0; i<100; i++ )
7        {
8            a[i] = c*b[i];
9        }
10       …
11   }
```



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   }
```



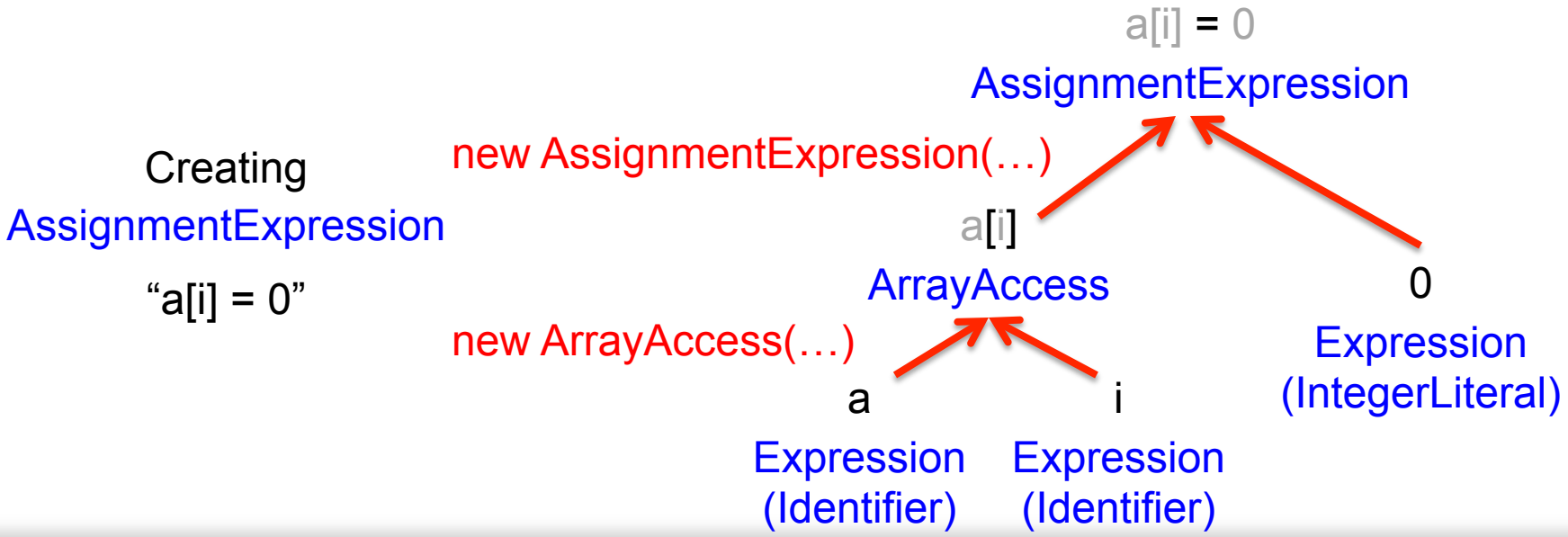
http://ft.ornl.gov/research/openarc

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.

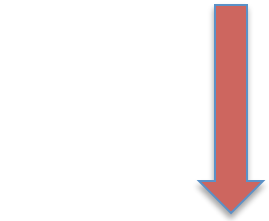


Modifying OpenARC IR Objects

- 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.

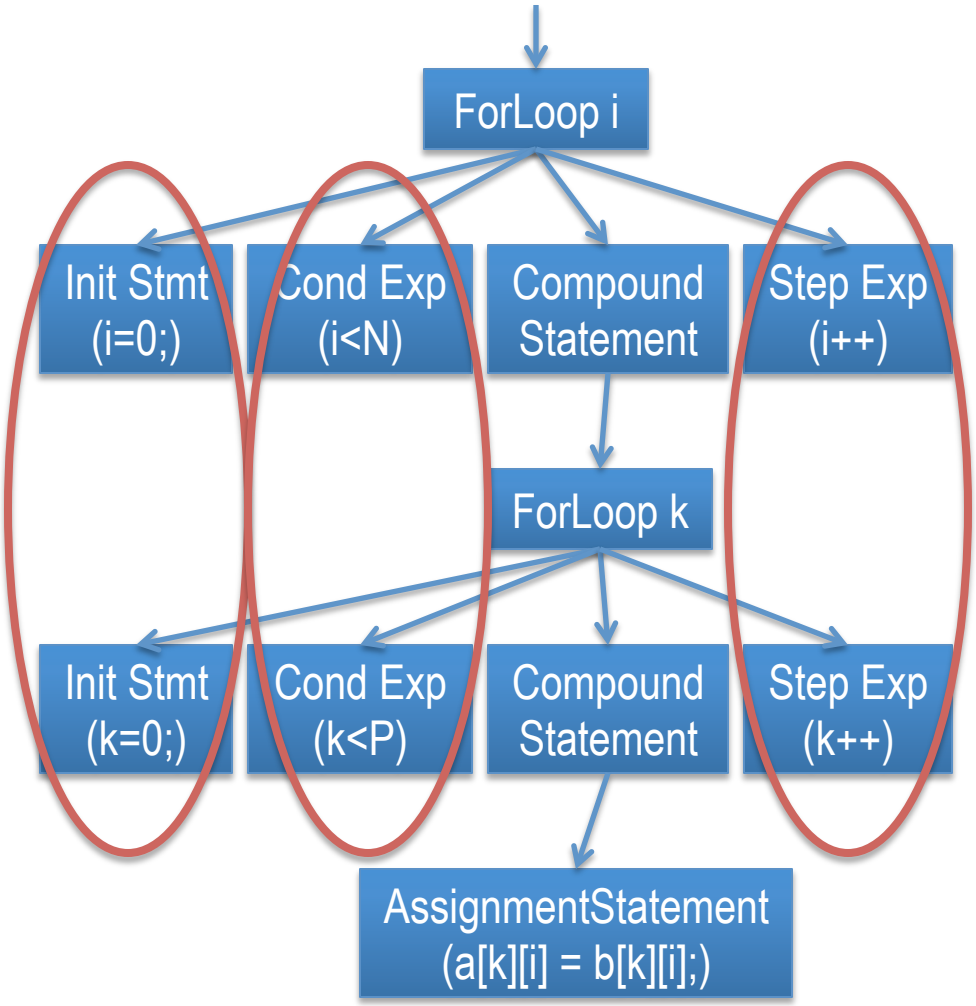
Example Transformation – Loop swap

```
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];
  }
}
```

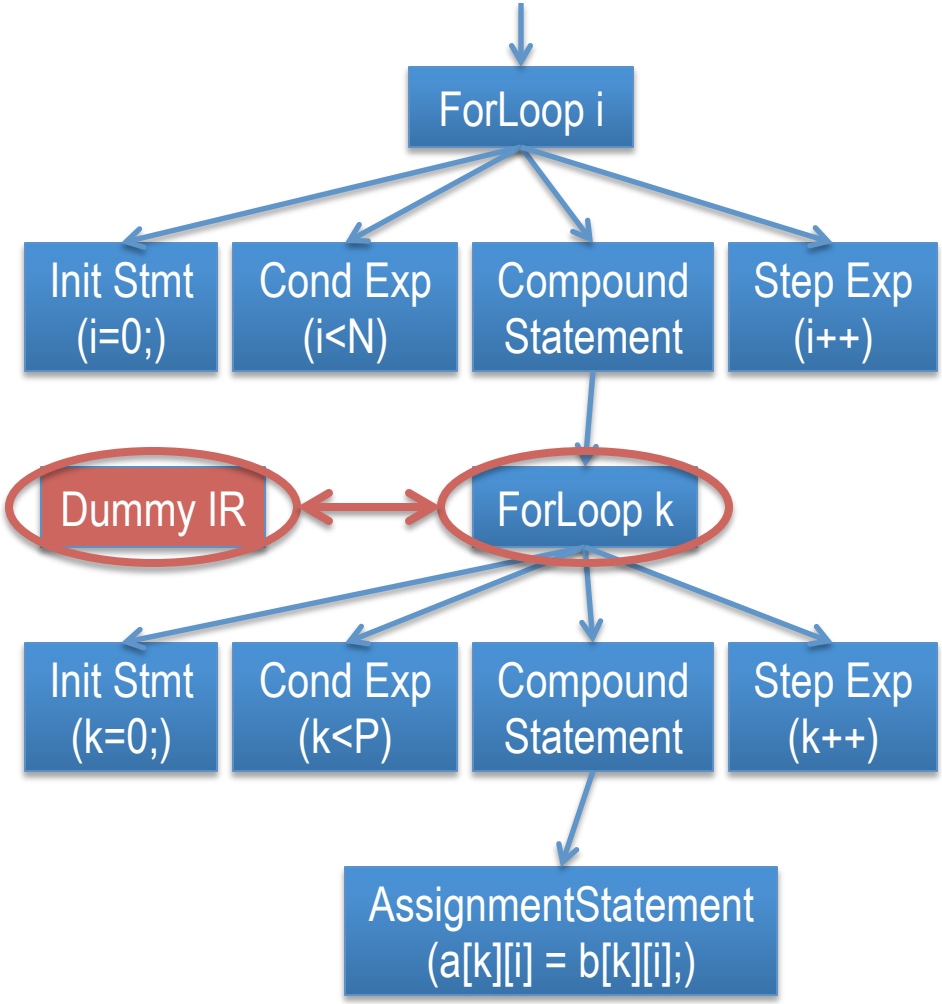
Swap each child except loop body



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

Example Transformation – Loop swap (cont.)

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

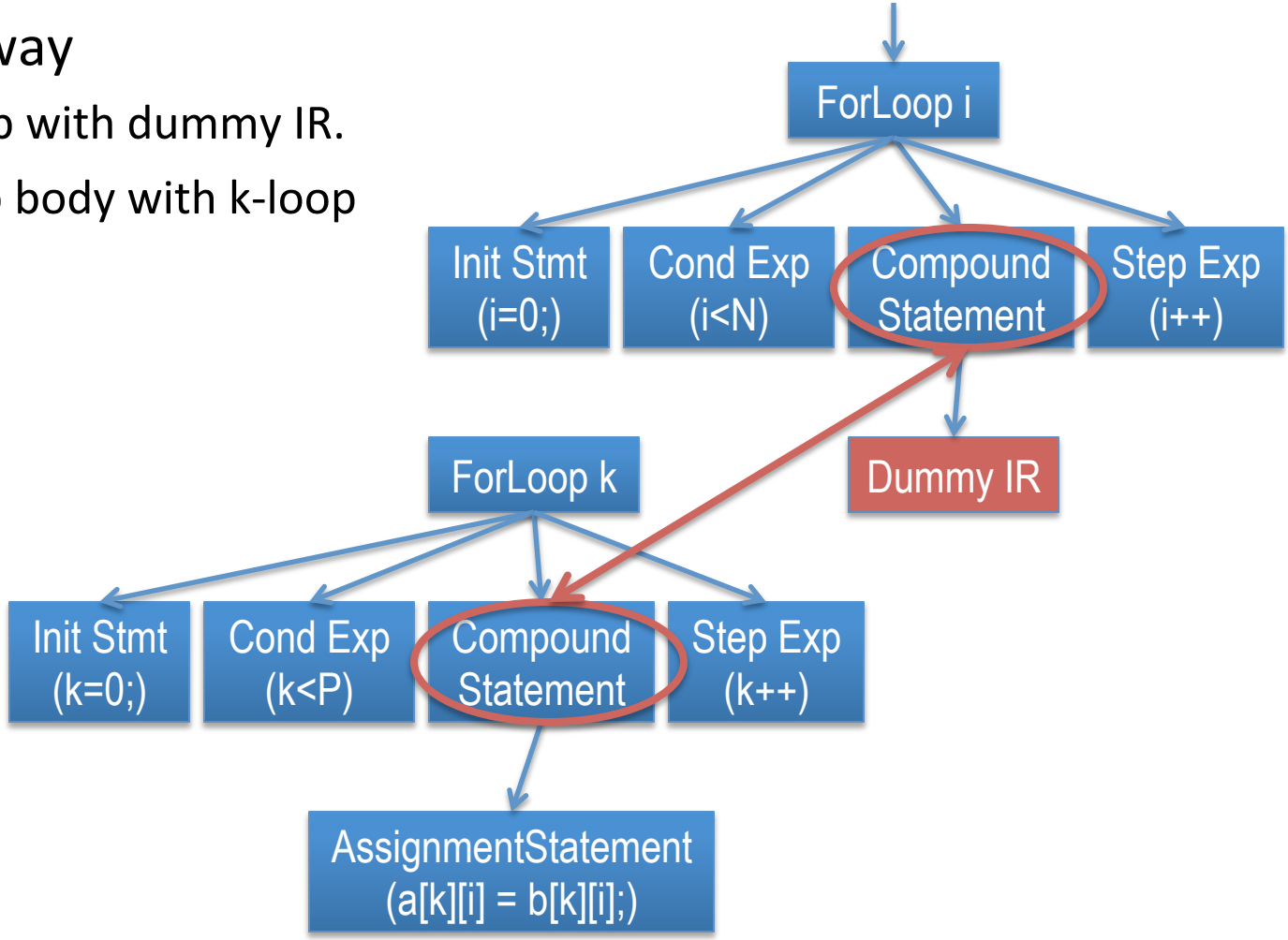


Example Transformation – Loop swap (cont.)

- Alternative way

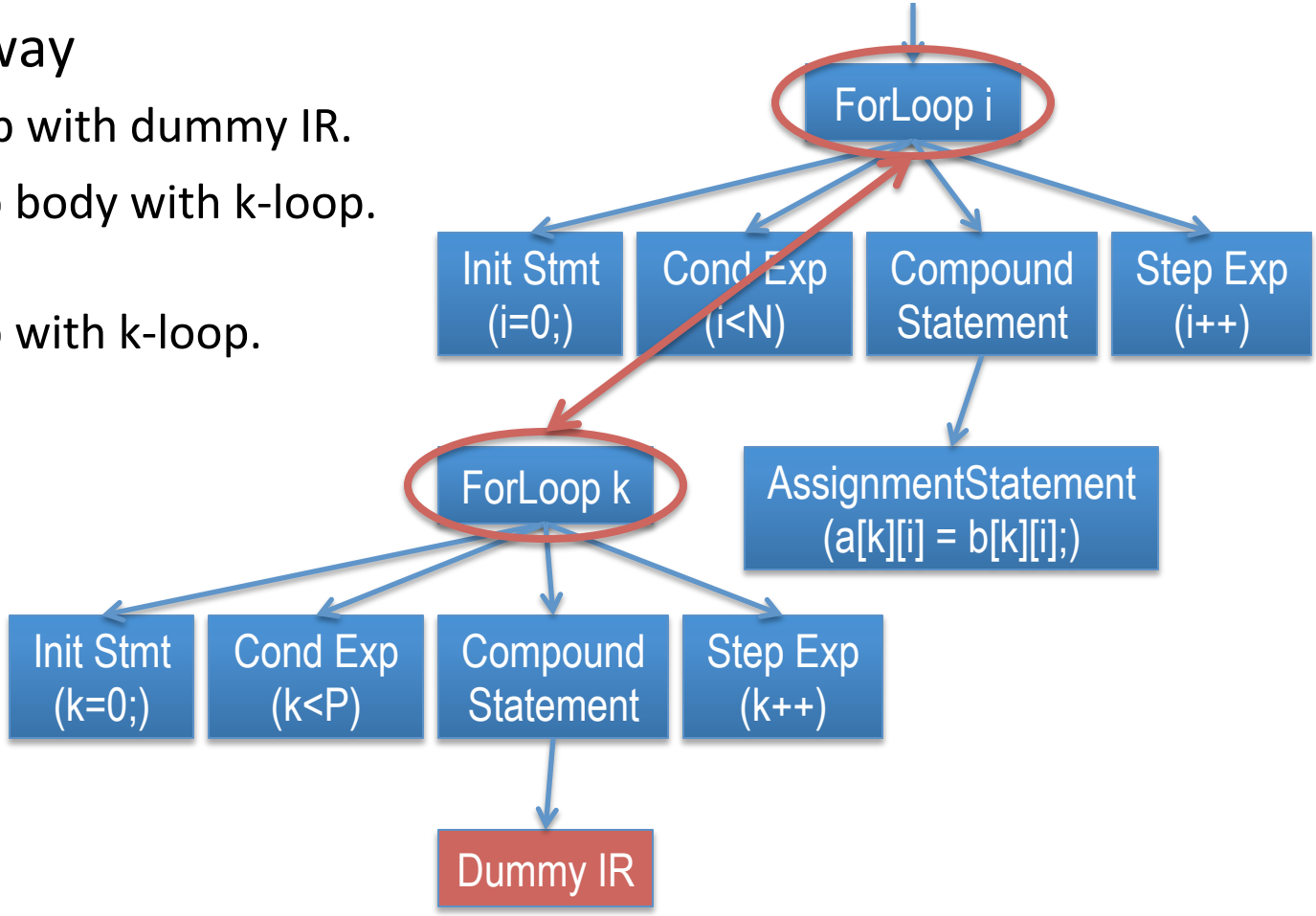
  1) Swap k-loop with dummy IR.

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



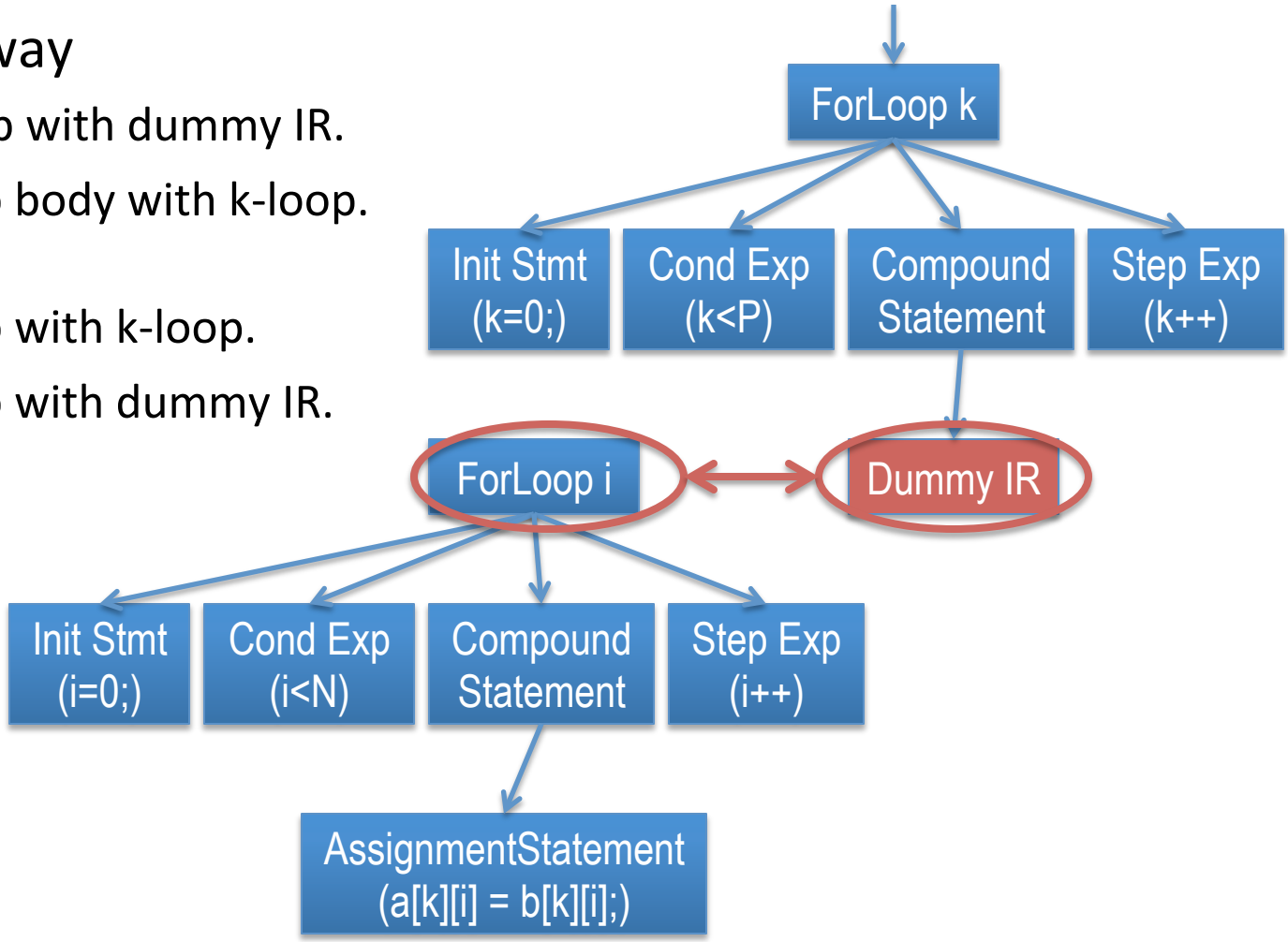
Example Transformation – Loop swap (cont.)

- 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.



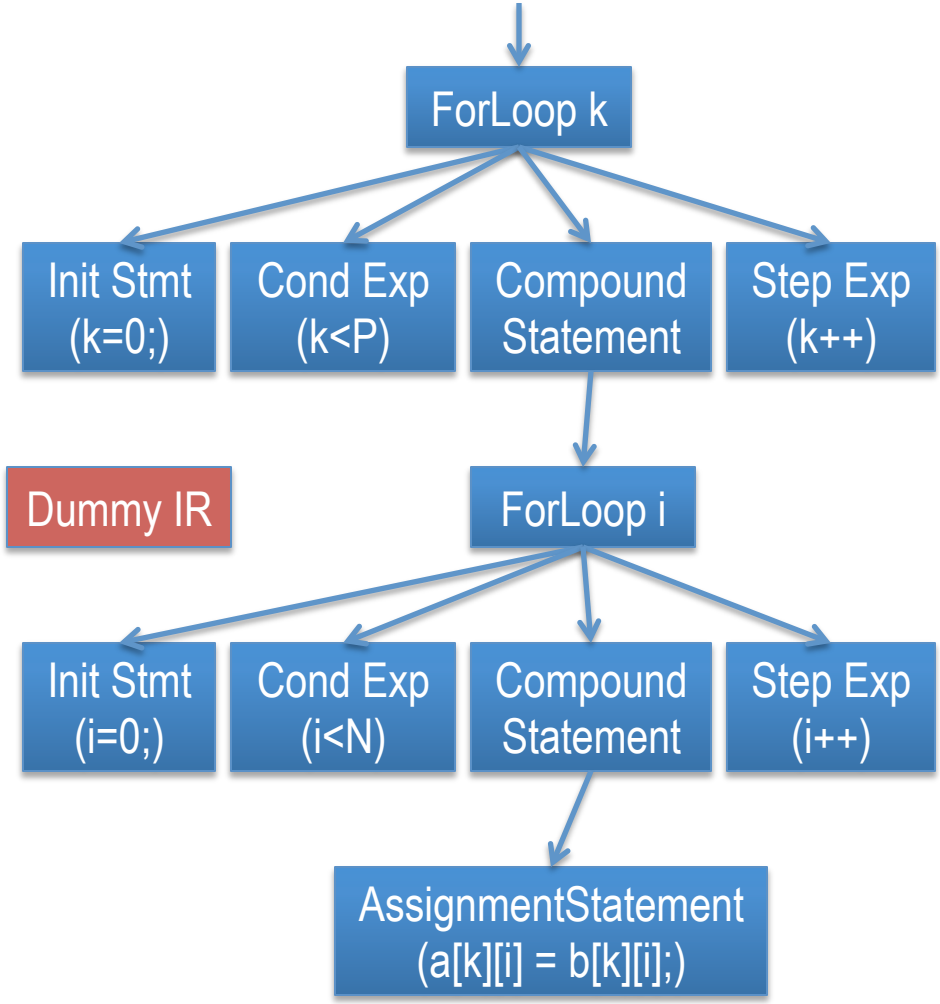
Example Transformation – Loop swap (cont.)

- 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.

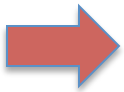


Example Transformation 2: Loop Unrolling

- 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