Building Source-to-Source Tools for High-Performance Computing

June. 25th, 2020

CASC
Center for Applied Scientific Computing

C3PO'20: Compiler-assisted Correctness Checking and Performance Optimization for HPC

Dr. Chunhua "Leo" Liao

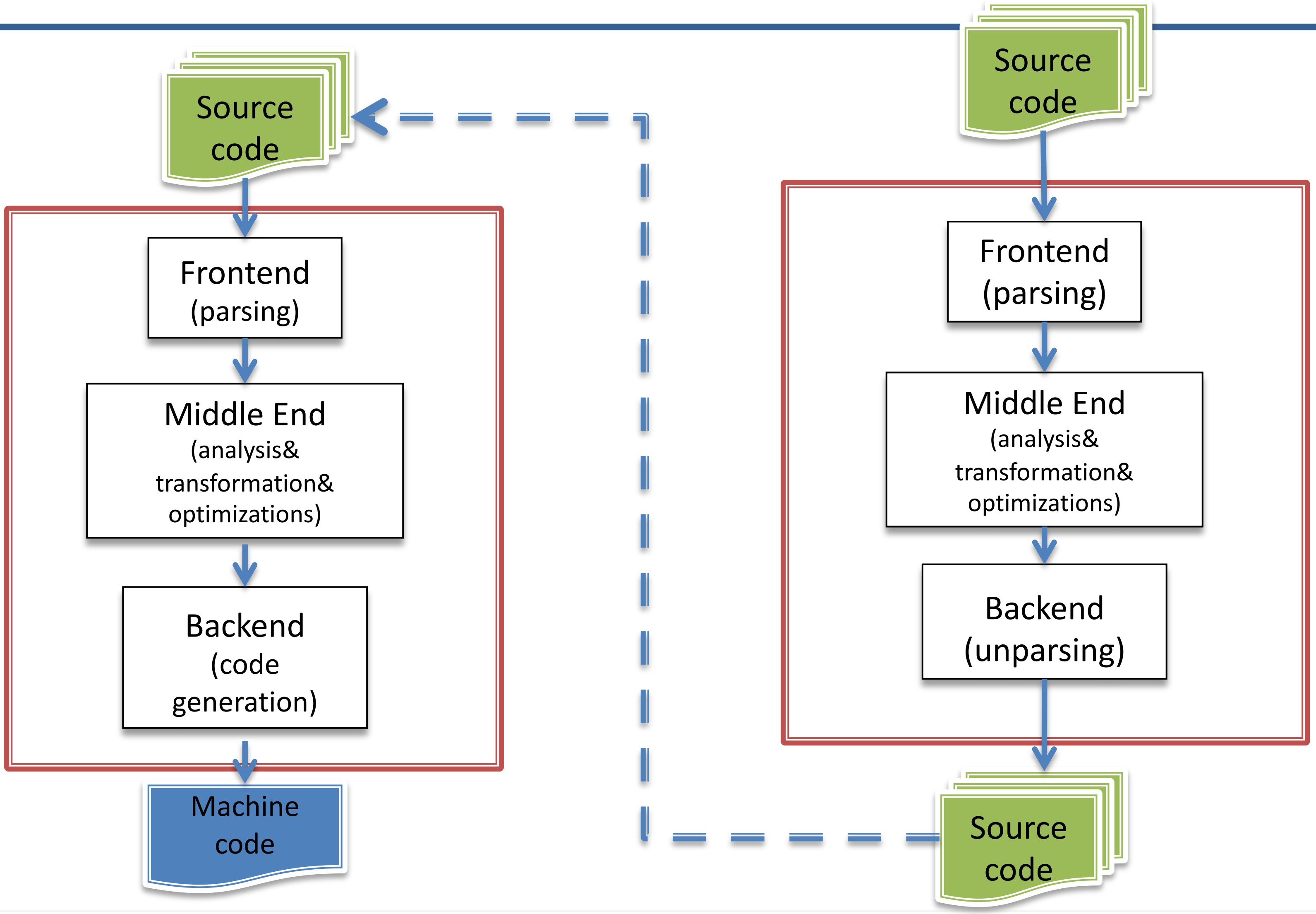
LLNL-PRES-811505

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Agenda

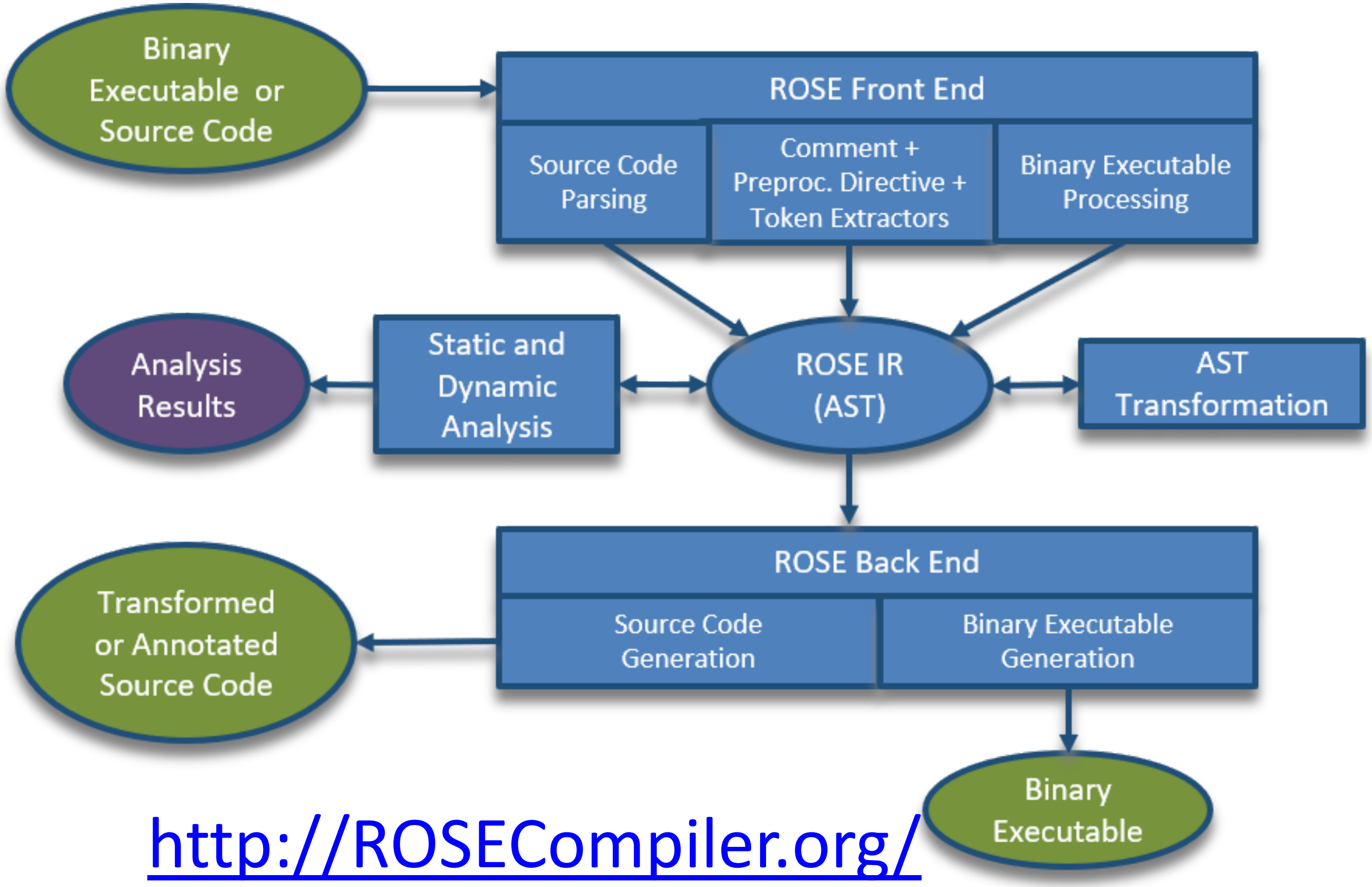
- Background
- Tools
  - Inliner
  - Outliner
  - The Move Tool
- Supportive work
  - Benchmarking
  - Tools as Services
  - FreeCompilerCamp
- Conclusion

Compilers: traditional vs. source-to-source

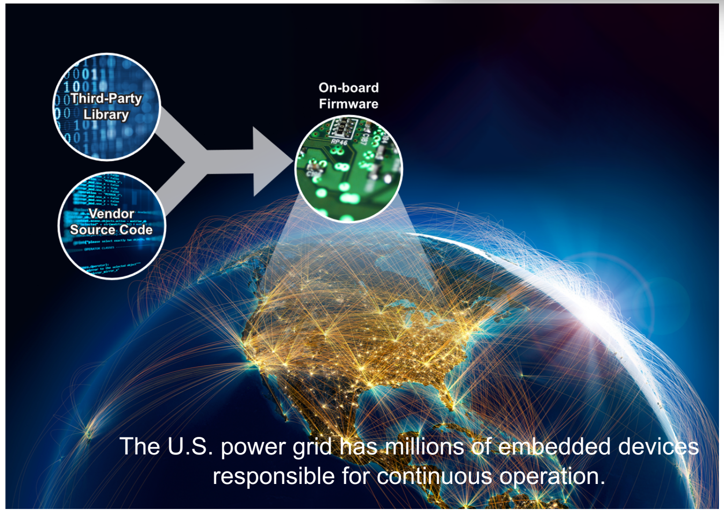


Human-Readable
Modifiable
Portable

ROSE: Enabling Compilers-based Tools for Critical Applications

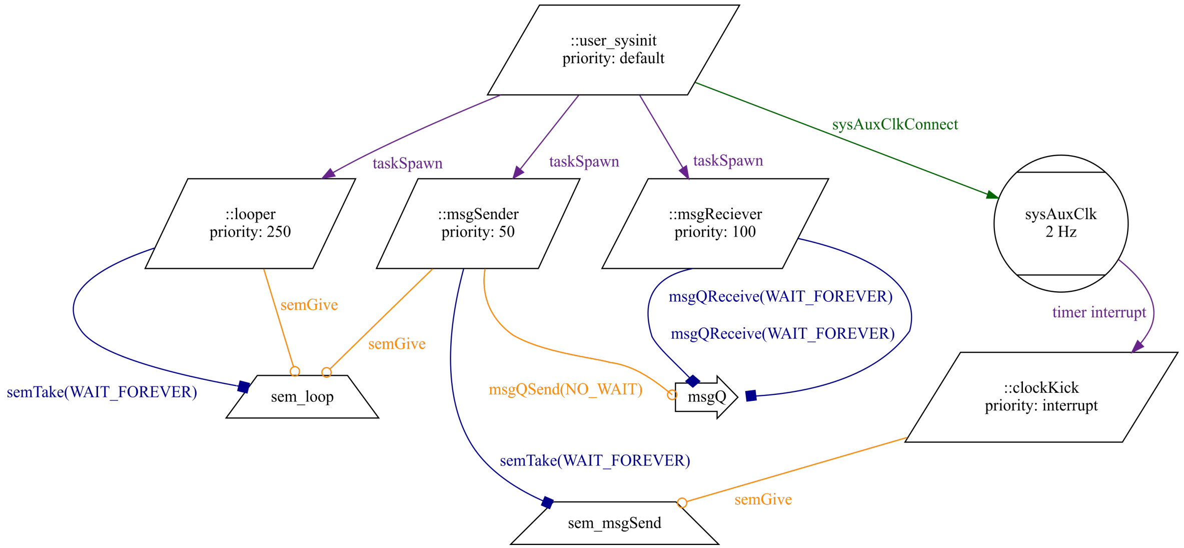


http://ROSECompiler.org/



Tools Built Using ROSE

- **Analyzers**: understanding, correctness, …
  - Visualization tools
  - Arithmetic Intensity measuring tool
  - NULL pointer analyzer
  - Data race detection tool



Automatically-Generated Machine Chart for multithreaded firmware with 110K SLOC

Tools Built Using ROSE (Cont.)

- **Translators**: Optimization, modernization, refactoring, patching …
  - The AST Inliner
  - The AST Outliner
  - OpenMP Lowering for CPUs/GPUs
  - AutoPar
  - Loop Processor
  - Declaration move tool
  - Code patching tool

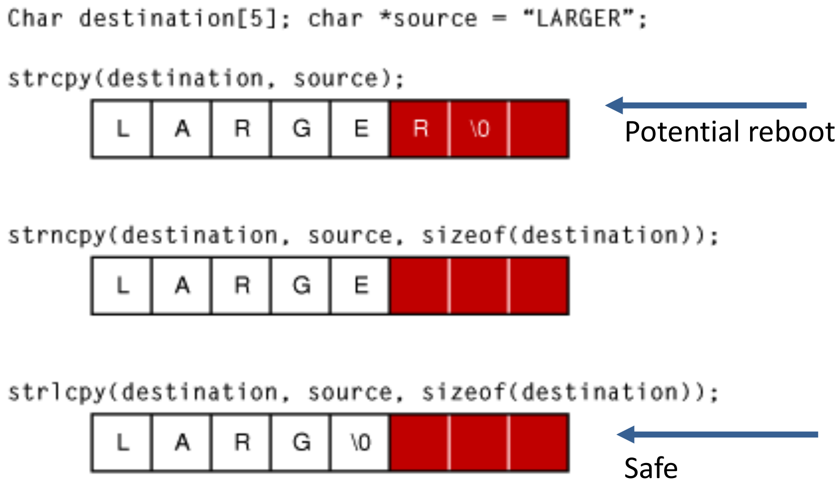


Image: developer.apple.com

Identifies unsafe functions and creates patches
e.g. detect and repair unsafe strcpy

The AST Inliner for C/C++

- Inlining: replacing a function call with the function body of the called function
  - C++: template functions, some with specialization
  - C++11: lambda expressions (anonymous functions)
- Benefits:
  - Traditional: eliminating overhead of function calls, enabling analysis and optimization which otherwise only work on single functions
  - Source-level C++ inlining: facilitating program understanding, enabling optimizations

```
template<typename T>
void swap(T& x, T& y)
{
  T tmp = x;
   x = y;
  y = tmp;
}

int foo (int a, int b)
{
   swap(a,b);
}
```

```
int foo(int a,int b)
{
    int &x__2 = a;
   int &y__3 = b;

    int tmp = x__2;
    x__2 = y__3;
    y__3 = tmp;

}
```

RAJA: C++ Abstractions Enabling Portable HPC Applications

RAJA

```
namespace RAJA
{
  // Template function
  template < typename EXE_POLICY_T, typename LOOP_BODY >
    void forall ( int begin, int end, LOOP_BODY loop_body )
    {
      forall ( EXE_POLICY_T ( ),  begin, end, loop_body );
    }

  // A sequential execution policy type
  struct seq_exec { }  ;

  // Template specialization for sequential execution
  template < typename LOOP_BODY >
    void forall ( seq_exec, int begin, int end, LOOP_BODY
loop_body )
    {
#pragma novector
      for ( int ii = begin; ii < end; ++ ii ) {
        loop_body ( ii );
      }
  }}
```

- Separating four core elements of loop execution
  1. Execution template: RAJA::forall
  2. Execution policy: RAJA::seq_exec
  3. Iteration space: RAJA::RangeSegment
  4. Loop body: lambda expressions

```
using EXEC_POLICY = RAJA::seq_exec;
RAJA::RangeSegment range(0, N);
RAJA::forall< EXEC_POLICY >( range, [=] (int i)
{
     a[i] += c * b[i];
} );
```

Inlining C++ template functions with lambda expressions

RAJA

```
namespace RAJA
{
  // Template function
  template < typename EXE_POLICY_T, typename LOOP_BODY >
    void forall ( int begin, int end, LOOP_BODY loop_body )
    {
      forall ( EXE_POLICY_T ( ),  begin, end, loop_body );
    }

  // A sequential execution policy type
  struct seq_exec { } ;

  // Template specialization for sequential execution
  template < typename LOOP_BODY >
    void forall ( seq_exec, int begin, int end, LOOP_BODY loop_body )
    {
#pragma novector
      for ( int ii = begin; ii < end; ++ ii ) {
        loop_body ( ii );
      }
  }}
```

Code Using RAJA

```
void foo()
{
   const int n=100;

   double *a = new double [100];

   RAJA::forall< class RAJA::seq_exec  >
     ( 0, n,
       [=] (int i)  {    a[i] = 0.5;     }
      );
}
```

After Inlining*

```
void foo ()
{
  const int n=100;
  double *a = new double [100];

#pragma novector
  for (int ii = 0; ii < 15; ++ii) {
      a[ii] = 0.5;
  }
}
```

* Working in progress

Inliner Algorithm

1. Eligibility check:
   a) Only allow named function, static member function, non-virtual member function, with known function body
2. Promoting function call expressions:
   a) e.g. a= func1() + b; → auto temp = func1(); a= temp + b;
3. Copy the body of the function to be inlined
   a) Create local variables for each formal argument, initialized with the actual argument
   b) Replace variable references with actual arguments
   c) Insert a label to indicate the end of the function body
   d) Convert return x to a code block
      1. E.g.: return x ; → {x; goto func_end;}
4. Postprocessing: cleanup the inlined code
   a) Remove unused labels,
   b) Remove goto to immediate next statement

The AST Outliner: Effective source-to-source Outlining

Outlining: semantically the reverse transformation of inlining

Used for kernel generation, OpenMP lowering for CPUs and GPUs, autotuning of whole programs

```c
#include <omp.h>
#include <stdio.h>

int num_steps = 10000;

int main() {
    double x = 0;
    double sum = 0.0;
    double pi;
    int i;
    double step = 1.0/(double) num_steps;

    // Run the code in parallel
    #pragma omp parallel for private(i,x) \
            reduction(+:sum) schedule(static)
    for (i=0; i<num_steps; i=i+1) {
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }

    pi=step*sum;
    printf("%f\n", pi);
}
```

(a) OpenMP program to calculate PI

```c
... // omitted headers and a data structure declaration storing variable addresses
static void OUT__1__2189__(void *__out_argv);
int main(int argc, char **argv) {
  ... // omitted variable declarations
  XOMP_parallel_start(OUT__1__2189__,&__out_argv1__2189__,1,0,"demo.c",10);
  XOMP_parallel_end("demo.c",15);
  pi = step * sum;
  printf("%f\n",pi);
  XOMP_terminate(status);
}
static void OUT__1__2189__(void *__out_argv) {
  ... // omitted variable declarations
  double *sum = (double *)(((struct OUT__1__2189___data *)__out_argv) -> sum_p);
  double *step = (double *)(((struct OUT__1__2189___data *)__out_argv) -> step_p);
  XOMP_loop_default(0,num_steps - 1,1,&p_lower_,&p_upper_);
  for (p_index_ = p_lower_; p_index_ <= p_upper_; p_index_ = p_index_ + 1) {
    _p_x = (p_index_ + 0.5) * *step;
    _p_sum = _p_sum + 4.0 / (1.0 + _p_x * _p_x);
  }
  XOMP_atomic_start();
   *sum = *sum + _p_sum;
  XOMP_atomic_end(); XOMP_barrier();
}
```

(b) Transformed (or Lowered) code

The AST Outliner: Algorithm and User Interface

- Perform side-effect and liveness analysis
- Bottom up traverse the AST and process each outlining target
  - Check the eligibility of a target
  - Create an outlined function
    - Create a function skeleton with parameters
    - Handle function parameters: decide pass by value vs. reference
    - Move the target into the outlined function's body
    - Replace variable references: variable cloning to avoid pointer uses
  - Replace the target with a call to the outlined function

```
Usage: outline [OPTION]... FILENAME...
Main operation mode:
        -rose:outline:preproc-only
        -rose:outline:abstract_handle handle_string
        -rose:outline:parameter_wrapper
        -rose:outline:structure_wrapper
        -rose:outline:enable_classic
        -rose:outline:temp_variable
        -rose:outline:enable_liveness
        -rose:outline:new_file
        -rose:outline:output_path
        -rose:outline:exclude_headers
        -rose:outline:use_dlopen
        -rose:outline:enable_debug
```

outline -rose:outline:abstract_handle ”ForStatement<position,12>” -rose:outline:use_dlopen test3.cpp
// outline the for loop located at line 12 of test3.cpp, call it using dlopen

Parameter Handling & Reducing Pointer Dereferences

- Scope and linkage
  - C: global only
  - C++: global vs. class-scope , C-linkage
- Parameters: for control and data
  - Goal: a few parameters as possible
  - Rely on scope, side effect and liveness analysis

Parameters = ((AllVars − InnerVars − GlobalVars − NamespaceVars −ClassVars) ∩ (LiveInVars U LiveOutVars)) U ClassPointers

PassByRefParameters = Parameters ∩ ((ModifiedVars ∩ LiveOutVars) U ArrayVars U ClassVars)

- Variables pass-by-reference handled by classic algorithms: pointer dereferences
- We use a novel method: variable cloning
  - Check if such a variable is used by address: address-taken analysis
    - C: &x;
    - C++: T & y=x; or foo(x) when foo(T&)
  - Use a clone variable if x is NOT used by address and is assignable

CloneCandidates = PassByRefParameters ∩ PointerDereferencedVars

CloneVars = (CloneCandidates − UseByAddressVars) ∩ AssignableVars

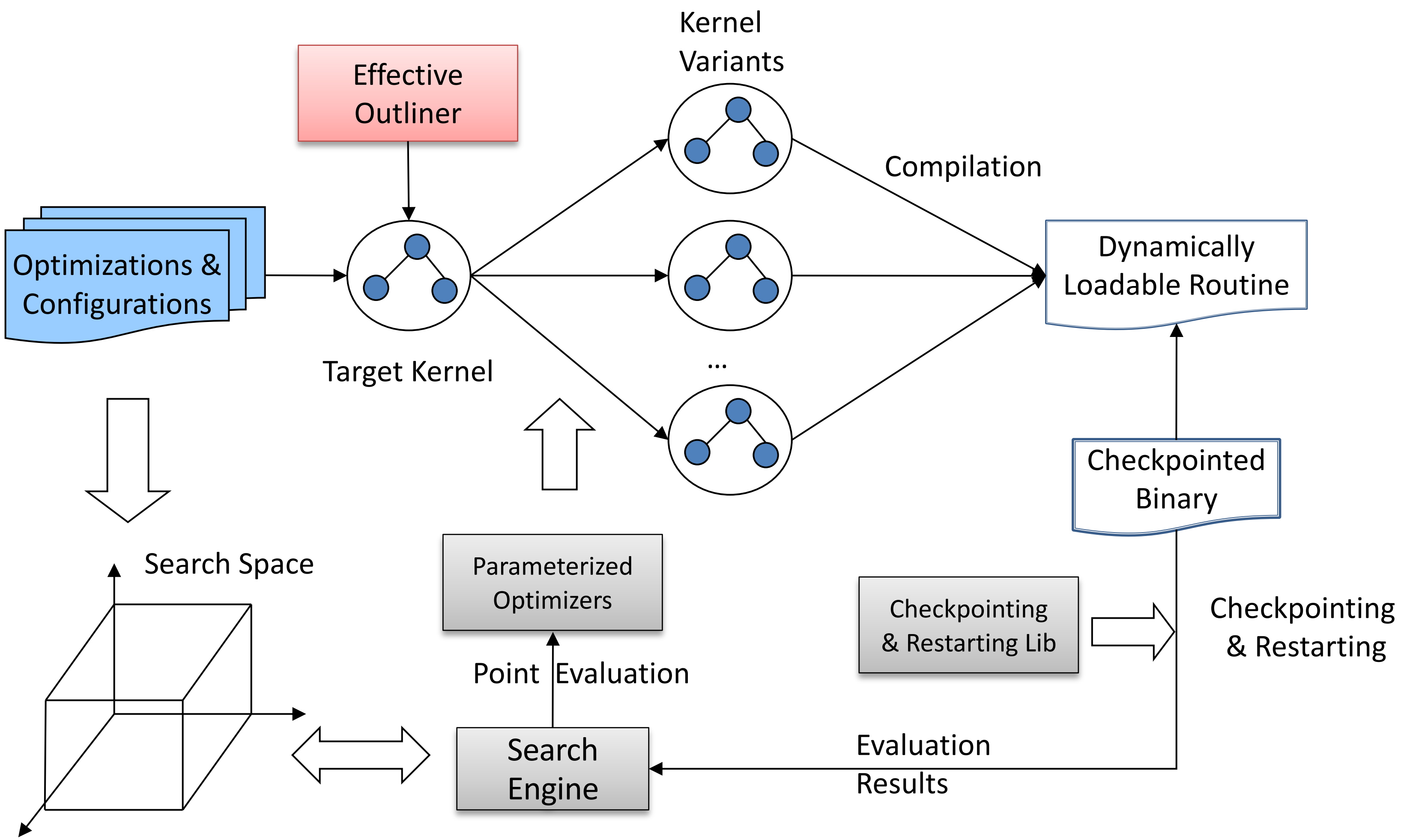
CloneVarsToInit = CloneVars ∩ LiveInVars

CloneVarsToSave = CloneVars ∩ LiveOutVars

Pointer-Dereferencing vs. Variable Cloning

Classic algorithm with pointer-dereferencing	Outlining with variable cloning
<pre>void OUT__1__4027__(int *ip__, int *jp__, double omega, double *errorp__, double *residp__, double ax, double ay, double b) { // Four variables becomes pointers: i,j, resid, error for (*ip__=1;*ip__<(n-1);(*ip__)++) for (*jp__=1;*jp__<(m-1);(*jp__)++) { *residp__ = (ax * (uold[*ip__-1][*jp__] + uold[*ip__+1][*jp__]) + ay * (uold[*ip__][*jp__-1] + uold[*ip__][*jp__+1]) + b * uold[*ip__][*jp__] - f[*ip__][*jp__])/b; u[*ip__][*jp__] = uold[*ip__][*jp__] - omega * (*residp__); *errorp__ = *errorp__ + (*residp__) * (*residp__); } }</pre>	<pre>void OUT__1__5058__(double omega,double *errorp__, double ax, double ay, double b) { int i, j; /* neither live-in nor live-out*/ double resid ; /* neither live-in nor live-out */ double error ; /* clone for a live-in and live-out parameter */ error = *errorp__; /* Initialize the clone*/ for (i = 1; i < (n - 1); i++) for (j = 1; j < (m - 1); j++) { resid = (ax * (uold[i - 1][j] + uold[i + 1][j]) + ay * (uold[i][j - 1] + uold[i][j + 1]) + b * uold[i][j] - f[i][j]) / b; u[i][j] = uold[i][j] - omega * resid; error = error + resid * resid; } *errorp__ = error; /* Save value of the clone*/ }</pre>

The Outliner Used for Whole Program Autotuning



SMG (semicoarsing multigrid solver) 2000
- 28k line C code, stencil computation
- 120x120x129 data set
- a kernel ~45% execution time for

Results:

- 5.55x Speedup for kernel
- 1.76x Speedup for total execution time

The Move Tool: a Code Refactoring Tool to Move Variable Declarations into Innermost Scopes

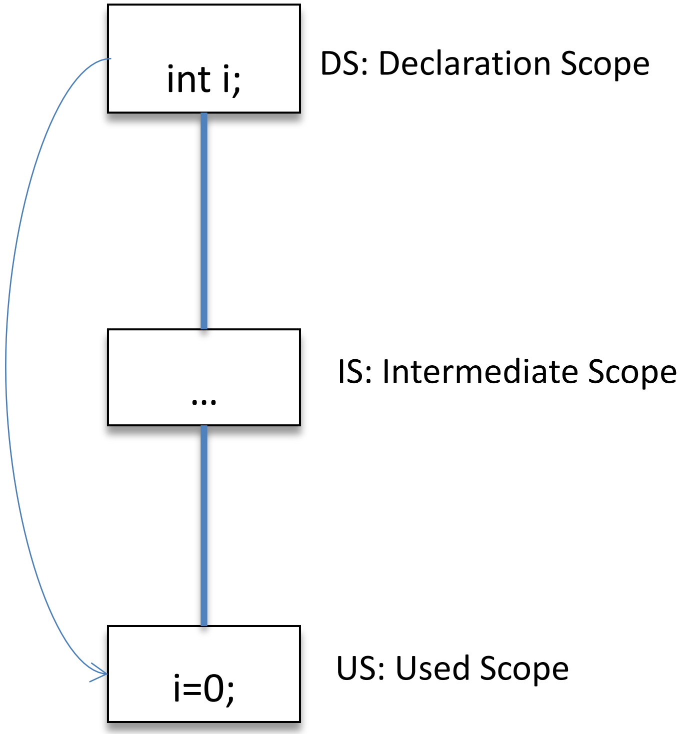
- A source-to-source refactoring tool to support ASC application teams
  - Copy-move variable declarations into innermost scopes: variable privatization
  - Benefits: facilitate code parallelization (migrating to OpenMP/RAJA)

- Algorithm went through 3 versions
  - V1: Naïve single-round move
  - V2: Iterative move using a declaration worklist
  - V3: Separated analysis and movement: much more efficient

Case 1: Single Used Scope vs. Case 2: Multiple Used Scopes

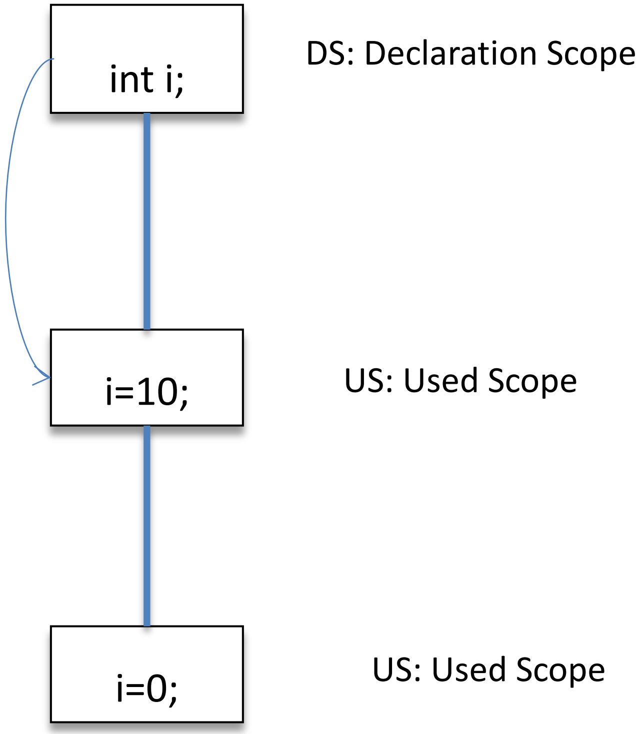
```
void foo()
{
 int i;
 {
   {
     i =0;
   }
 }
}
```

Code with a declaration



a scope tree:
three types of Scope Nodes
parent-child edges

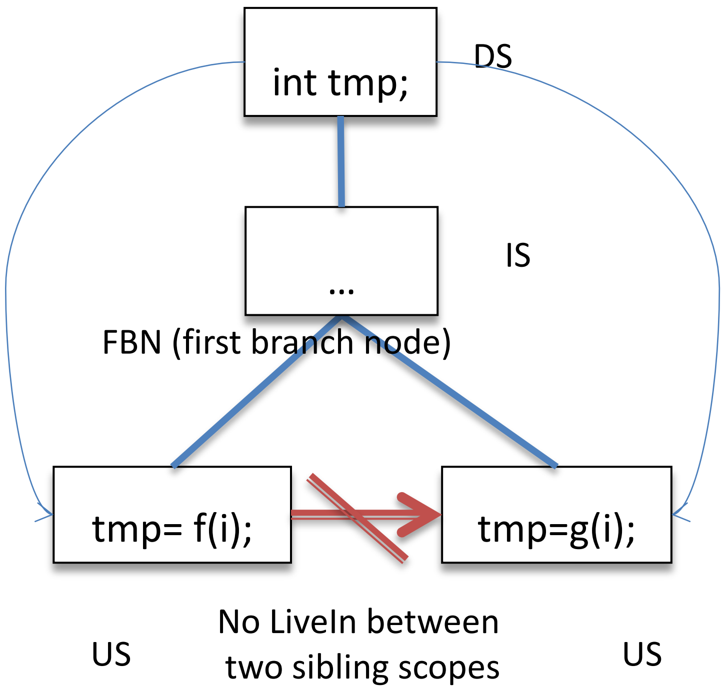
```
void foo()
{
 int i;
 {
   i = 10;
   {
     i =0;
   }
 }
}
```



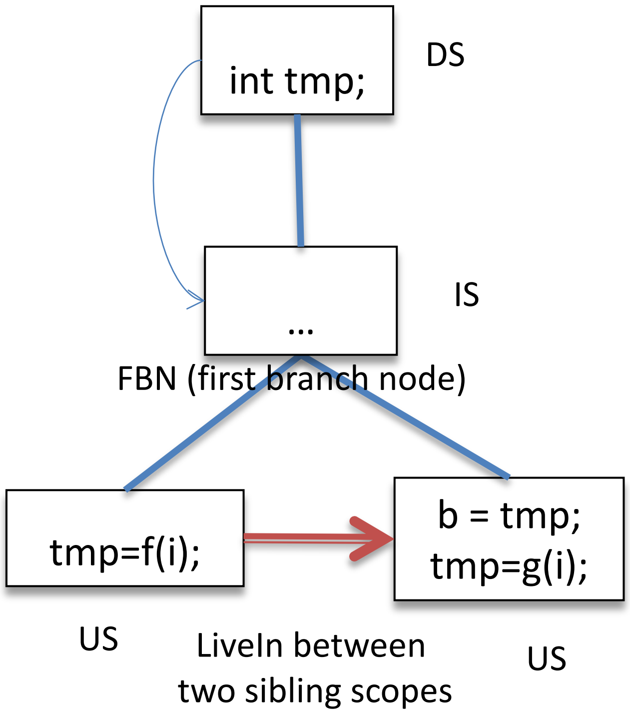
scope tree with multiple used scopes
* trim shadowed used scope

Case 3: Multiple Used Scope Branches of the Same Length

```
{
  int tmp ;
  {
   {
      tmp = f(i) ;
   }
   /*… */
   {
     tmp = g(i) ;
   }
  }
}
```



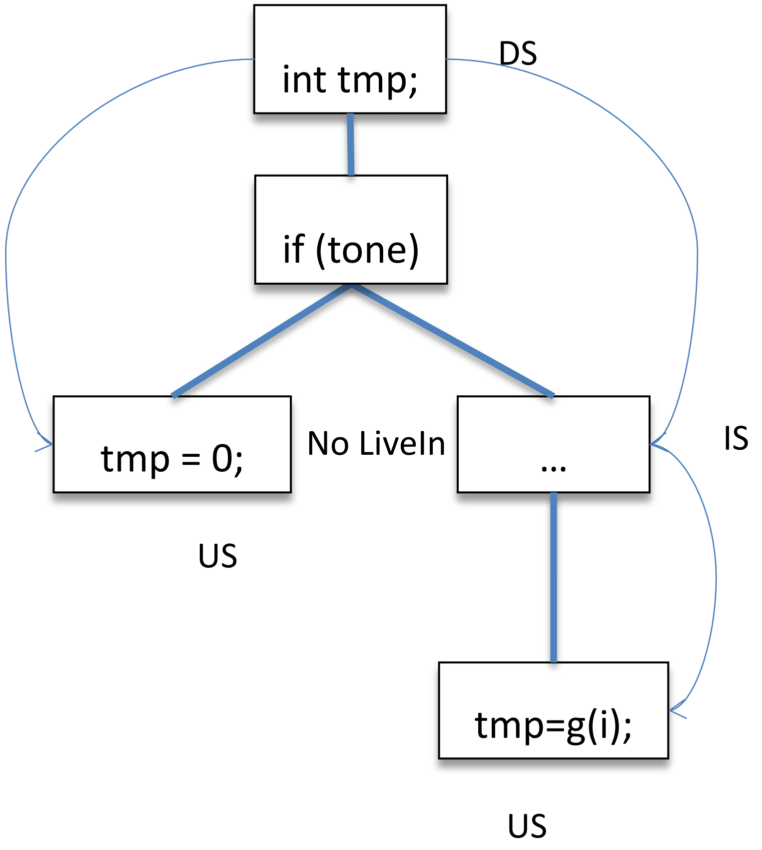

```
{
  int tmp ;
  {
   {
      tmp = f(i) ;
   }
   /*  … */
   {
      b = tmp;
      tmp = g(i) ;
   }
  }
}
```



Baseline algorithm V1: handles case 1,2 and 3

Case 4: Multiple Branches with Different Lengths

```
int  tmp;
if (tone)
{
    tmp = 0;
}
else
{
  {
     {
       tmp = 0;
     }
  }
}
```

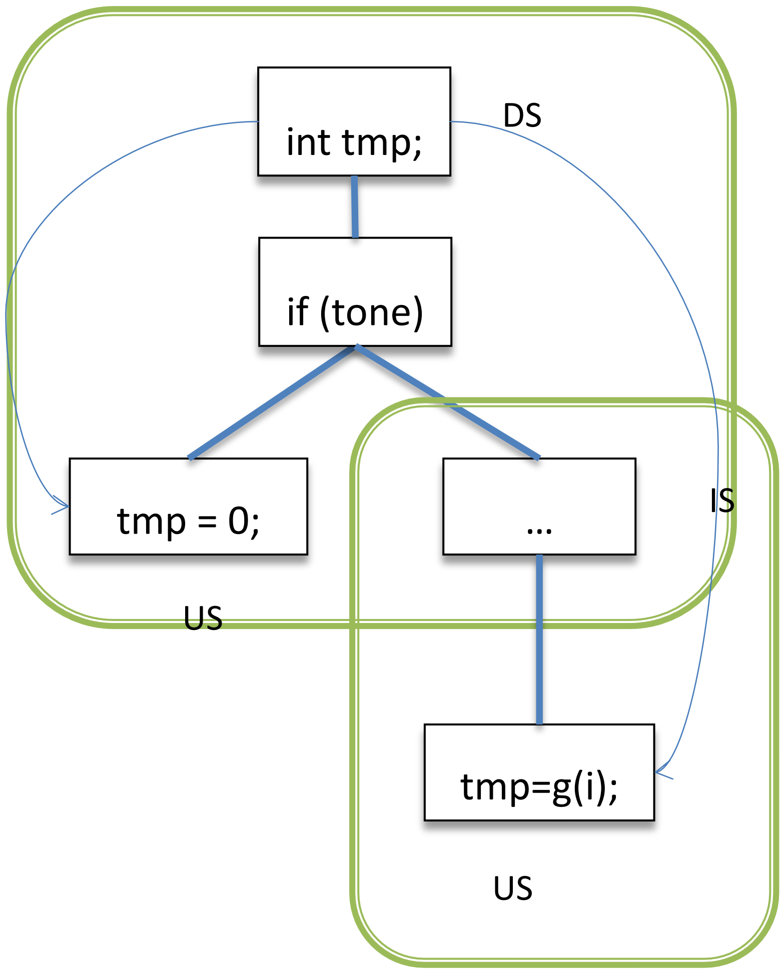


Need further moves

Algorithm V2: iteratively move declarations

- A declaration copy-moved to a new location
  - the newly inserted declaration should be considered for further movements
  - Focus on declarations
- An iterative algorithm using a worklist
  - initial worklist = original declarations in the function
  - while (!worklist.empty())
    - decl = worklist.front(); worklist.pop();
    - moveDeclarationToInnermostScope(decl, inserted_decls);
    - worklist.push_back(each of inserted_decls)

Only Need to Find Final Scopes and Move Once: Algorithm V3



- Find final scopes first
  - scope_tree_worklist.push(scope_tree);
  - while (!scope_tree_worklist.empty())
    - — current_scope_tree = scope_tree_worklist.front(); …
    - — collectCandidateTargetScopes(decl, current_scope_tree);
      - if (is a bottom scope?)
        - target_scopes.push_back(candidate)
      - else
        - scope_tree_worklist.push_back(candiate)
- Then copy&move in one shot
  - if (target_scopes.size()>0)
    - — copyMoveVariableDeclaration(decl, target_scopes);

Results

- 230+ regression tests, with correctness verification (diff-based)
- Applied to large-scale X,Y apps, very positive user feedback
- Users kept requesting more features once previous requests were met
  - merge moved declarations with immediately followed assignments
  - transformation tracking, debugging support
  - aggressive mode, keep-going mode, no-op mode, …

Rethinking the Success Metric of HPC

Success(HPC) = f(FLOPs, Watt)

HPC = Highly Painful Computing

Sacrificing hours of hard human cycles for a few reduced machine cycles

Would some application teams really want to use the HPC software/hardware systems we dump on them every 3-5 years, if they had choices??

A New Holistic Success Metric for HPC as a Service

Success(HPC) =

f (Total_time, Quality_of_results, Total_cost, Context)

Total_time = the entire end-to-end, machine-human interaction time to get results
- Human_time = training, thinking_steps, keystrokes, mouse_clicks, cursor_travel_distance, ~~hairs_pulled_off~~, …

Quality_of_results:
- Correctness, accuracy, certainty/confidence, up-to-date …

Total_cost = Machine_cost + Human_cost
- Human cost tied to hourly rates: make HPC operable/usable by even cavemen

Context: under which conditions can HPC serve users (including cavemen)?
- Access devices (smartphones), Locations (AOE), Time (24x7), …

Benchmarking to Understand Quality of Tools

If You Can't Measure it Correctly,
You Can't Improve it

Regression positive/negative tests

Metric	Formula
Precision	Confidence of true positive P = TP/(TP + FP)
Recall	Completeness of true positive R = TP/(TP + FN)
Accuracy	Chance of having a correct report A = (TP +TN)/(TP + FP +TN + FN)

```
...
int i,x;
#pragma omp parallel for
 for (i=0;i<100;i++)
 {   x=i;  }
 printf("x=%d",x);
...
```

one data race pair
x@5 vs. x@5

lastprivatemissing-orig-yes.c

```
...
int i,x;
#pragma omp parallel for lastprivate (x)
for (i=0;i<100;i++)
 {   x=i; }
printf("x=%d",x);
...
```

lastprivate-orig-no.c

https://github.com/LLNL/dataracebench

Evaluation Report

Tool-Compiler	Tests	Test Results				Metrics			Testing Error				Test Time (hh:mm:ss)
		TP	FN	TN	FP	Prec.	Recall	Acc.	CSF	CUN	RSF	RTO	
Archer1.0-Clang3.9.1	376	187	24	145	0	1.00	0.89	0.93	5	5	10	0	00:06:11
Archer2.0-Clang6.0.0	386	202	20	156	3	0.99	0.91	0.94	0	5	0	0	00:06:17
Inspector2008-Intel17.0.2	392	195	30	156	9	0.96	0.87	0.90	2	0	0	0	01:32:50
Inspector2008-Intel18.0.2	396	198	27	160	8	0.96	0.88	0.91	0	0	0	3	02:04:34
Inspector2018-Intel19.0.0	396	213	12	60	108	0.66	0.95	0.69	0	0	0	3	03:41:17
Inspector2018-Intel19.0.4	396	198	27	160	11	0.95	0.88	0.90	0	0	0	0	01:33:54
Inspector2019-Intel17.0.2	392	195	30	159	6	0.97	0.87	0.91	2	0	0	0	01:37:08
Inspector2019-Intel18.0.2	396	195	30	162	6	0.97	0.87	0.91	0	0	0	3	02:04:49
Inspector2019-Intel19.0.0	396	214	11	61	107	0.67	0.95	0.70	0	0	0	3	03:32:55
Inspector2019-Intel19.0.4	396	195	30	164	7	0.97	0.87	0.91	0	0	0	0	01:37:27
ROMP-Clang8.0.0	384	198	18	144	6	0.97	0.92	0.93	0	6	9	3	00:59:20
Tsan5.0.2-Clang5.0.2	386	192	30	153	3	0.98	0.86	0.91	0	5	0	3	00:36:28
Tsan6.0.1-Clang6.0.1	386	195	27	156	3	0.98	0.88	0.92	0	5	0	0	00:07:34
Tsan7.1.0-Clang7.1.0	386	193	29	154	5	0.97	0.87	0.91	0	5	0	0	00:07:19
Tsan8.0.1-Clang8.0.1	384	184	38	152	4	0.98	0.83	0.89	0	6	0	0	00:07:03

Compile-time seg. fault (CSF), Unsupported feature (CUN) Runtime seg. Fault (RSF), Runtime timeout (RTO)

Regression of Tools

ID	R	Tool-Compiler														
		Arch.1-Cl.391	Arch.2-Cl.600	Ins.18-In.1702	Ins.18-In.1802	Ins.18-In.1900	Ins.18-In.1904	Ins.19-In.1702	Ins.19-In.1802	Ins.19-In.1900	Ins.19-In.1904	ROMP-Cl.800	Tsan5-Cl.502	Tsan6-Cl.601	Tsan7-Cl.710	Tsan8-Cl.801
5	Y	✓	✓	✗	✗	✓	✓/✗	✗	✗	✓	✗	✓	✓	✓	✓	✗
6	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓/✗	✓/✗	✓/✗	✗
7	Y	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
8	Y	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
13	Y	✓	✗	✗	✓/✗	✗	✓/✗	✗	✗	✓/✗	✗	✓	✓/✗	✓	✓/✗	✗
23	Y	✓/✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
24	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
25	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
27	Y	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
29	Y	✓	✓	✓/✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
34	Y	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓/✗	✓/✗	✓/✗	✓
37	Y	RSF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
39	Y	✓	✓	✗	✓/✗	✓	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓
40	Y	✓	✓	✓/✗	✗	✓	✓/✗	✗	✗	✓	✗	✓	✓/✗	✓/✗	✓/✗	✓/✗
41	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
42	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
43	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
44	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
45	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
46	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
47	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓
48	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
49	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
50	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
52	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
53	N	RSF	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
54	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
55	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
56	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
57	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
59	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
60	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
61	N	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓

IDs not shown are benchmarks that are correctly evaluated with every tool.

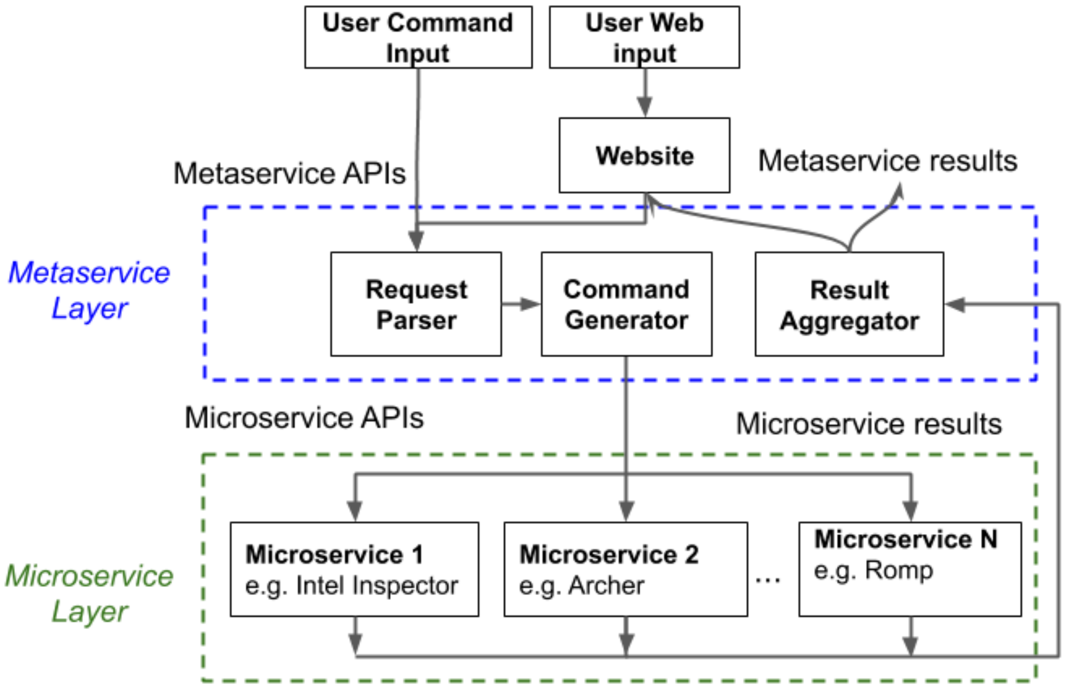
Tools as Services to Reduce Human Costs and to allow more user context

Motivation

- Hard to use individual tools
- All individual tools have limitations

Solution

- Compose tools as cloud-based services
- Define APIs
- Define JSON formats



RaceDetectionService: A Cloud-Based Metaservice for Detecting Data Races

Data Race Detection Service: RESTful API and JSON

HTTP Method	URL	Parameters	Description
GET	/RDS	N/A	List available race detection services' IDs, such as meta, micro-archer, micro-romp, etc.
POST	/RDS/service-id	SyncFlag, file, options	Send a file to a race detection service specified by its id, with extra options. The service will finish all the work and return a report if the synchronous flag is set to true. Otherwise, the service will immediately return a request ID and a key to authenticate possible HTTP DELETE requests. The actual race detection work will be executed in background.
GET	/requests	N/A	List all requests submitted to all services
GET	/requests/request-id	N/A	Check the status of a specific request, return a status of nonexist, finished, pending, running.
DELETE	/requests/request-id	key	Cancel an ongoing request, return a status of nonexist, success or failure.

```json
{
  "program": "a.out",
  "data_races": [
    {
      "read": {
        "location": ["file1.c",64,12],
        "symbol": "A[i]"
      },
      "write": {
        "location": ["file1.c",64,11],
        "symbol": "A[i]"
      },
      "microservices": [
        {"Archer": true},
        {"ROMP" : true},
        {"ThreadSanitizer": true},
        {"Inspector": false}
      ]
    },
    {
      "read": {
        "location": ["file2.c",132,7],
        "symbol": "b"
      },
      "write": {
        "location": ["file2.c",246,31],
        "symbol": "b"
      },
      "microservices": [
        {"Archer": true},
        {"ROMP" : false},
        {"ThreadSanitizer": true},
        {"Inspector": true}
      ],
      "raw_output": [
        {"Archer": "shorturl.at/uzJR7"},
        {"ROMP" : "shorturl.at/enwH4"},
        {"ThreadSanitizer": "shorturl.at/otO67"},
        {"Inspector": "shorturl.at/dH389"}
      ]
    }
  ],
  "aggregate_policy": "Union"
}
```


28

Preliminary results of RDS

Tool	Aggregate Policy	TP	FP	TN	FN	Recall	Specificity	Precision	Accuracy	Adjusted F1
Intel Inspector	Union	51	30	27	8	0.864	0.474	0.629	0.672	0.729
ThreadSanitizer		**56**	31	26	**3**	**0.949**	0.456	0.644	0.707	0.767
Archer		51	3	50	7	0.879	0.943	0.944	0.910	0.872
ROMP		51	**1**	**56**	8	0.864	**0.982**	**0.981**	**0.922**	**0.919**
RDS	Union	**57**	53	4	**2**	**0.967**	0.070	0.518	0.526	0.675
	Intersection	45	**0**	**57**	14	0.763	**1.0**	**1.0**	0.879	0.865
	Random	53	8	49	6	0.898	0.860	0.869	0.860	0.869
	Majority (Positive tie breaker)	56	18	39	3	0.949	0.684	0.757	0.819	0.842
	Majority (Negative tie breaker)	53	3	54	6	0.898	0.947	0.946	0.922	0.922
	Weighted Vote	53	3	54	6	0.898	0.947	0.946	0.922	0.922
	Directive-Specific Weighted Vote	55	2	55	4	0.932	0.965	0.965	**0.948**	**0.948**

Automatic Online Training: FreeCompilerCamp.org

Problem:
- Many tools requested by app teams
- But only limited FTEs available

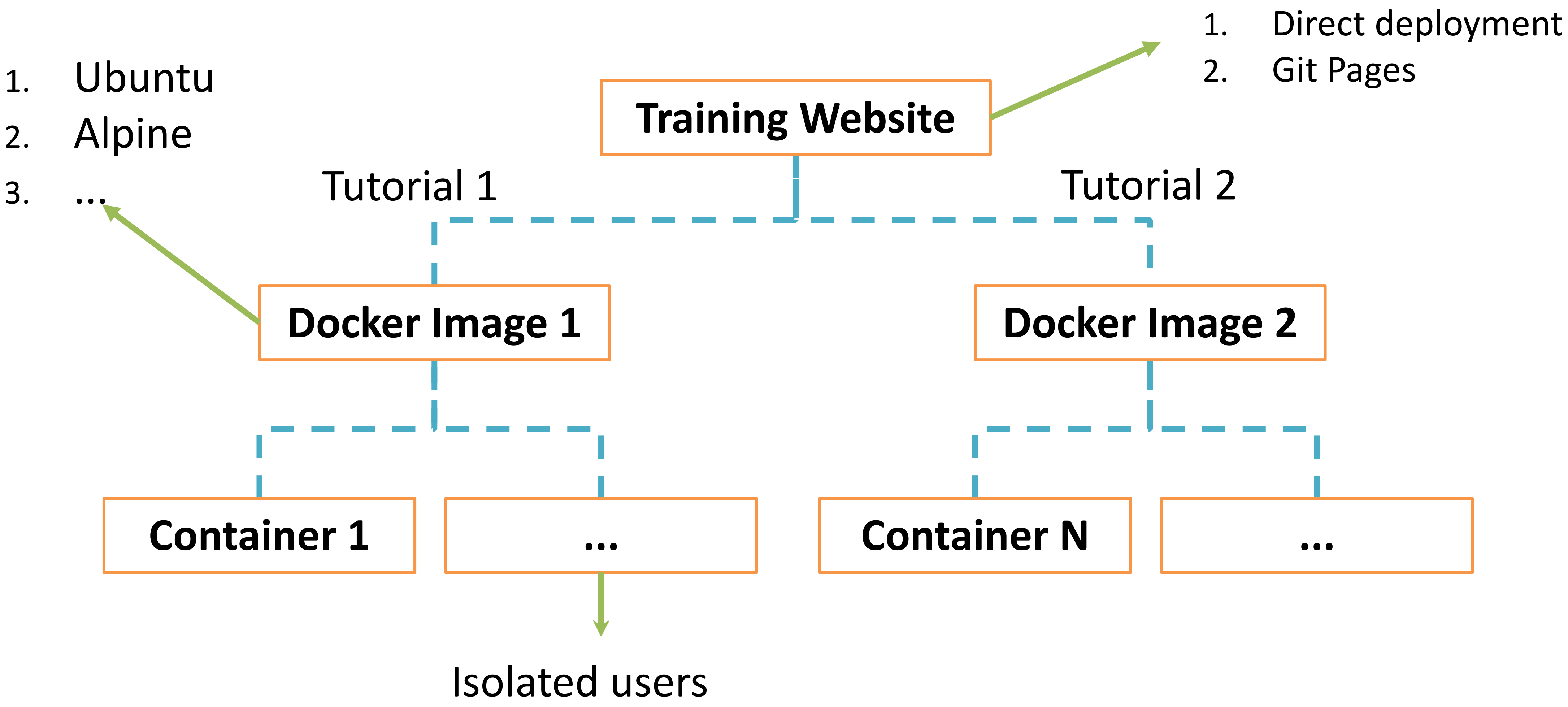
Solution: automatic training and certifying developers
- Modern Learning Management Systems (LMS) + Adaptive Learning/Assessement
- FreeCodeComp → FreeCompilerComp
- Interactive cloud-based playground for learners: Play-with-Docker

Pain Point	Description	Solution
Accessible	Paperwork to get accounts on suitable machines	Online sandbox terminal open to anyone
Installation	Many software packages are needed	Docker images
Effectiveness	Traditional text tutorials are not effective	Learning by doing, testing, certification
Content	No single person/group knows all details of compiler development	Self-made tutorials + crowd-sourcing to accept external contributions
Design trade-offs	One compiler cannot demonstrate all options	Hosting tutorials for multiple compilers
Costs	Hosting websites with containers costs money	Open-source, self-deployable framework
Security	Online websites have inherent risks	Containers + VM + AWS

Challenges and Solutions

Compilers : Parsing -> AST/IR -> Traversal (analysis) -> Transform (optimization) -> Runtime

FreeCompilerCamp Design



User interface

Free Compiler Camp Classroom

Essentially, we can see the following content:

```
// goal 1. generate
//  foo(p_sum);
// goal 2. generate
//  foo(0.5);
//   after inserting its header

// how parameter is used
void foo(int x);

int main (void)
{
  int p_sum=0;
  return p_sum;
}
```

C. Run sample program to insert a function call

After building the demo, there is executable file named `buildFunctionCalls` under the current directory:

```
ls buildFunctionCalls
```

Finally, run the demo tool to insert the function call to the sample input code:

```
./buildFunctionCalls -c inputbuildFunctionCalls.C
```

The generated source code still has the same name but wiht a prefix `rose_`. It's unparsed from updated AST. Be checking the new source code, it clearly shows that `foo()` is called with parameter `p_sum` now.

```
cat rose_inputbuildFunctionCalls.C
```

The line 13 and 14 verified that new function calls have been added to the AST.

```
...
10 int main()
11 {
12   int p_sum = 0;
13   foo(p_sum);
14   bar(0.500000);
15   return p_sum;
16 }
...
```

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window. If you are behind firewall, please open the firewall of your computer. Time remaining in this session : 1h 58m

```
2019-09-26 13:32:51 (31.3 MB/s) - 'inputbuildFunctionCalls.h' saved [123/123]

freecc@node1:astInterfaceTests$ cat inputbuildFunctionCalls.C
/// goal 1. generate
//  foo(p_sum);
// goal 2. generate
//  foo(0.5);
//   after inserting its header

// how parameter is used
void foo(int x);

int main (void)
{
  int p_sum=0;
  return p_sum;
}

freecc@node1:astInterfaceTests$ ls buildFunctionCalls
buildFunctionCalls
freecc@node1:astInterfaceTests$ ./buildFunctionCalls -c inputbuildFunctionCalls.C
freecc@node1:astInterfaceTests$ cat rose_inputbuildFunctionCalls.C
/// goal 1. generate
//  foo(p_sum);
// goal 2. generate
//  foo(0.5);
//   after inserting its header
// how parameter is used
#include "inputbuildFunctionCalls.h"
void foo(int x);

int main()
{
  int p_sum = 0;
  foo(p_sum);
  bar(0.500000);
  return p_sum;
}
freecc@node1:astInterfaceTests$
```

Takeaway Messages

ROSE: A source-to-source compiler framework for building tools for national security applications

- tools for source code and binary: inliner, outliner, autopar, move tool, loop processor …
- http://roseCompiler.org/

Tool Development

- Regression tests to reflect what users want (positive tests) and don't want (negative tests)
- Standard metrics to communicate incremental progress with sponsors and users
  - Precision, Recall, Accuracy
- Commenting on issues of apps == commenting on issues of children in front of their parents!

Takeaway Messages (Cont.)

- Success(HPC)=f (Flops, Watt) ➔ Highly Painful Computing for people
  - Let's refine the metric to include human factors together and make it Highly Pleasant Computing
- Doing my part to make HPC Highly Pleasant Computing
  - Benchmarks: people love and hate benchmarks
    - Best qualified people may not want to develop/release the best benchmarks for their work
  - Microservice design, docker, cloud,
  - Online learning/certifying frameworks, ….

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

Sponsored by DOE Office of Cybersecurity, Energy Security, and Emergency Response, Department of Defense, and LLNL.