OpenSHMEM Tools Eco-system

1990s

- SHMEM library introduced by Cray Research Inc. (T3D systems)
- Adapted by SGI for products based on the Numa-Link architecture and included in the Message Passing Toolkit (MPT).
- Vendor specific SHMEM libraries emerge (Quadrics, HP, IBM, gpSHMEM, SiCortex etc.).

- Extreme Scale Systems Center (funded by DOD), located at Oak Ridge National Laboratory, along with the University of Houston come together to address the differences between various SHMEM implementations.
- **OpenSHMEM** is born.
- OpenSHMEM Specification 1.0 finalized.

2012

- Development of **OpenSHMEM Reference Library, Validation** and **Verification Suite.**
- OpenSHMEM Specification 1.0d released to community.
- Specification supported by some Vendors.
- Compiler based (**OpenSHMEM Analyzer**), debuggers (**DDT**) and performance tools (**TAU, SHMEM Tracer**) support OpenSHMEM.

2013

- Development of the OpenSHMEM programming environment gains momentum.
- Tracing Tools support OpenSHMEM (**Vampir / Vampir Trace**)
- UCCS lower level communication library integration into OpenSHMEM Reference implementation.
- OpenSHMEM Specification 1.0e released to community.
- **OpenSHMEM Specification 1.1 will be released in March 2013**

OpenSHMEM Programming Environment

UCCS

ANALYZER

Tuning and Analysis Utilities

OpenSHMEM Reference Library

Open SHMEM

Vampir

allinea DDT

Debug

- **OpenSHMEM** is a community driven effort.
- Website: www.oprnshmem.org

Introduction

- **S**tate-of-the-art tools for OpenSHMEM
- **Tools are part of the OpenSHMEM programming environment**
- **We will present tools for:**
  - **Program Development**
  - **Error Checking**
  - **Performance Analysis**
  - **Performance Modeling**
  - **Debugging**

Introduction (cont)

We will cover the following tools:

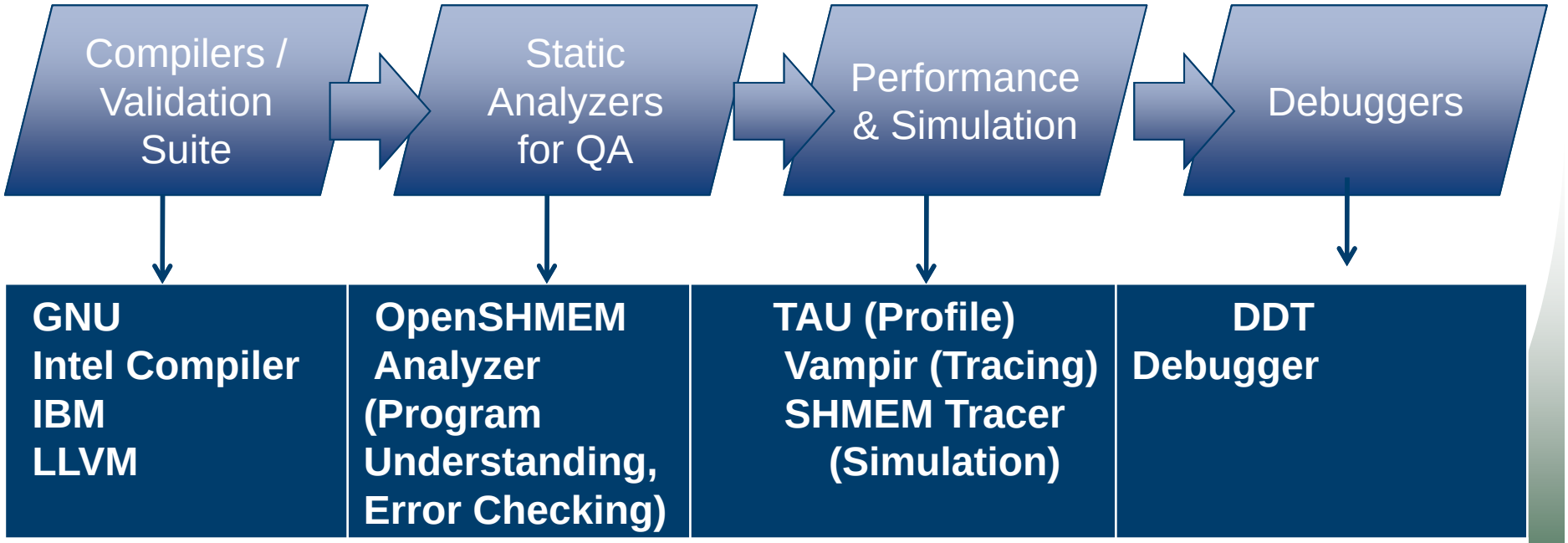
- **TAU performance system**
- Vampir / Score-P
- DDT debugger
- OpenSHMEM Analyzer

Why use OpenSHMEM tools?

- **Help understanding your application**
- **Fast prototyping to OpenSHMEM**
- **Increases productivity**
- **Improves performance and quality of software**
- **Facilitates the maintenance of the code**
- **Use leading technologies in the field**

Software Development Cycle

- **Tools are part a the programming environment for OpenSHMEM**



Static Analyzers: OpenSHMEM Analyzer

- **Understanding of an OpenSHMEM application**
  - **Callgraph, Control flow graphs, Def-Use info**
- **Helps to produce high quality code:**
  - **Prevents common errors in OpenSHMEM**
    - **Type checks, out-of-bounds, pointer and alias analysis**
  - **Provides information for optimization**
- **Provides Feedback to the user**
  - **Command based, User Based**
- **We will cover this tool in the lab session**

The OpenSHMEM Analyzer

- **Analysis that the tool can provide:**
  - **SHMEM-aware callgraphs, callsites**
    - **Visualization of synchronizations, atomics, memory I/O,**
  - **Mappings to the source code, source code browser.**
- **Errors and Semantic Checks:**
  - **Initialization / finalization calls are present in the code**
    - **PEs are initialized**
    - **Redundant PE initializations**
  - **Correct type information of OpenSHMEM calls**
    - **Data types and sizes check in put/get operations**
    - **Enforcing accesses to symmetric data structures**
  - **Data/Memory Management, advanced analysis**
    - **Buffer allocations / de-allocations**
    - **Check the memory allocation library used.**

Implementation

- **Based on a state-of-the-art compiler, that work on large applications.**
  - **Open64 AMD 4.2.5.2 compiler release.**
  - **Handles C/C++/Fortran + OpenSHMEM 1.0a Spec**
- **Relies heavily on Inter-procedural analysis.**
- **OpenSHMEM library is recognized by compiler and its semantics is used to perform checks**
  - **Analysis is exported to different formants:**
    - **Command line messages**
    - **Hyper-texted graphs/images that can be visualized with a web browser.**
- **Can be easily integrated to application Makefiles.**

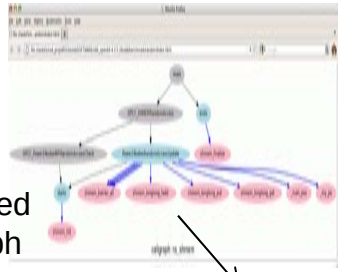
Evaluation of the OpenSHMEM

- **We validated the tool with the OpenSHMEM validation suite 1.0 provided by openshmem.org**
- **This includes the validation tests that comes with 1.0a**
- **We were able to reproduce the bugs intended for the validation suite.**
- **In one case we found an unintended bug in one of the cases.**

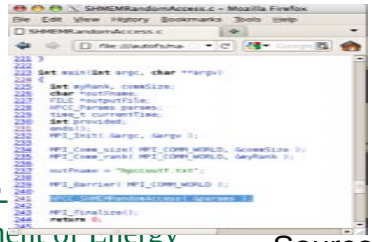
OpenSHMEM Analyzer Implementation

Compilers:

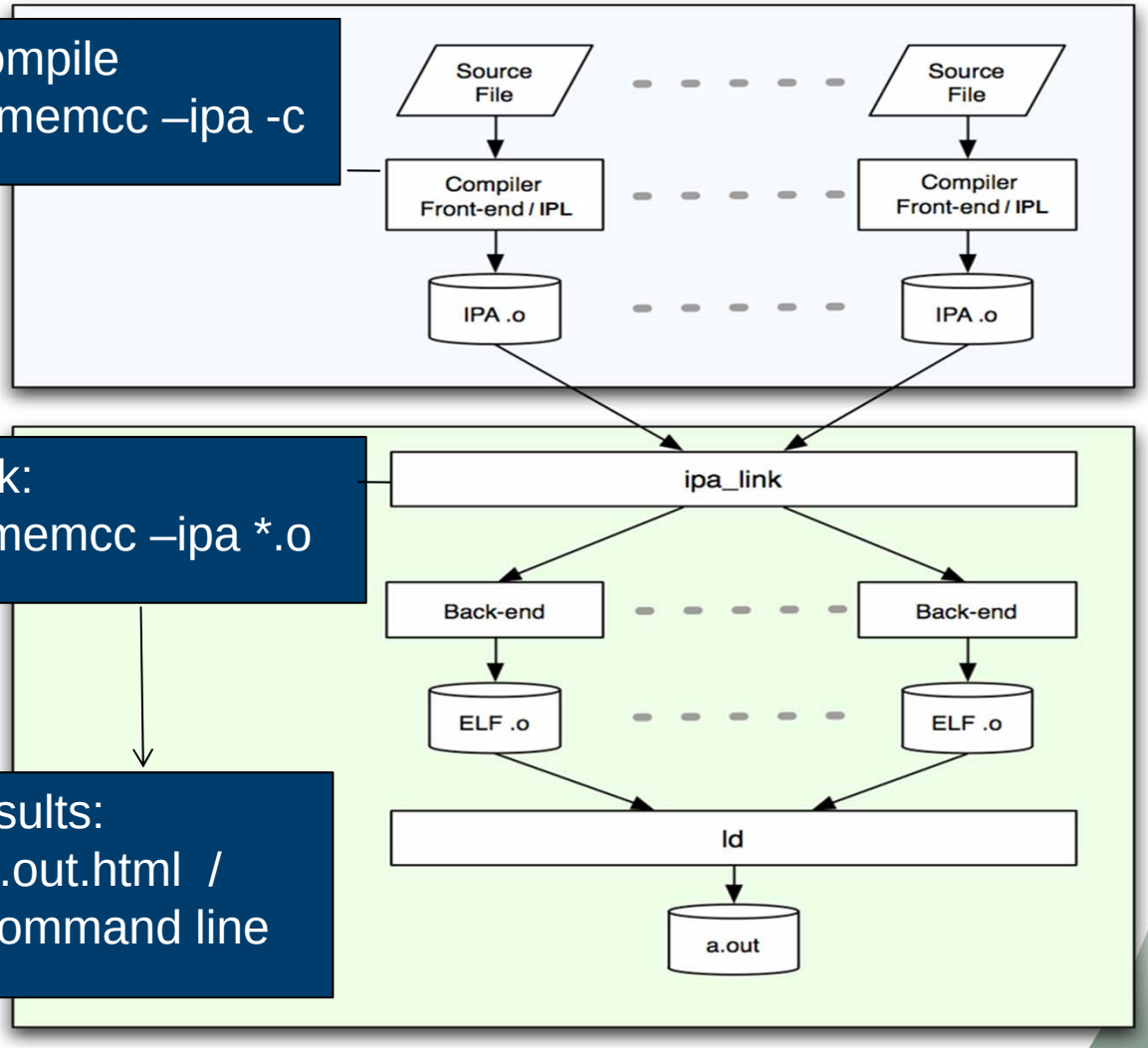
- Summarization & local analysis
- Analysis
- Detection / Optimization
- Visualization of results:



Annotated Callgraph

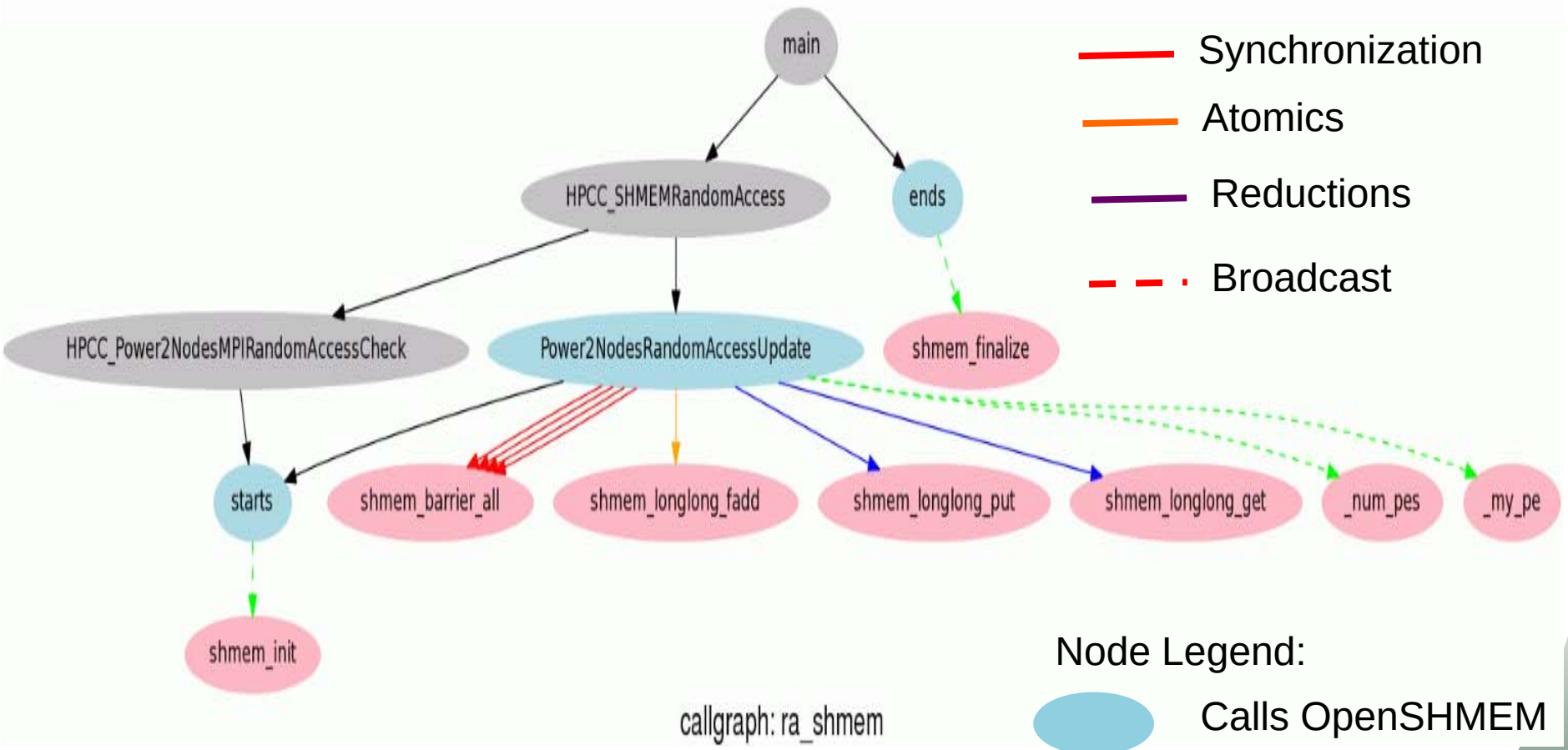


Source Code



OpenSHMEM analysis

- **Callgraph generated:**



Callsites Legend:

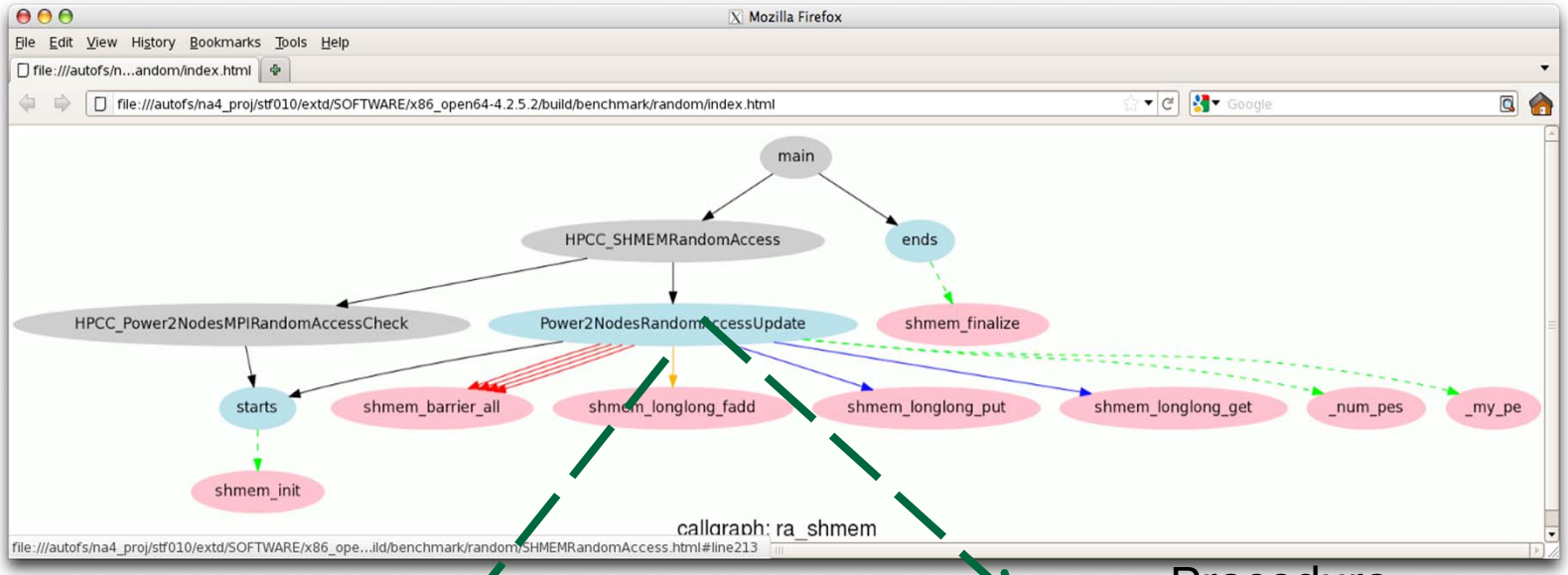
- Init & RTL
- I/O: put/get
- Synchronization
- Atomics
- Reductions
- Broadcast

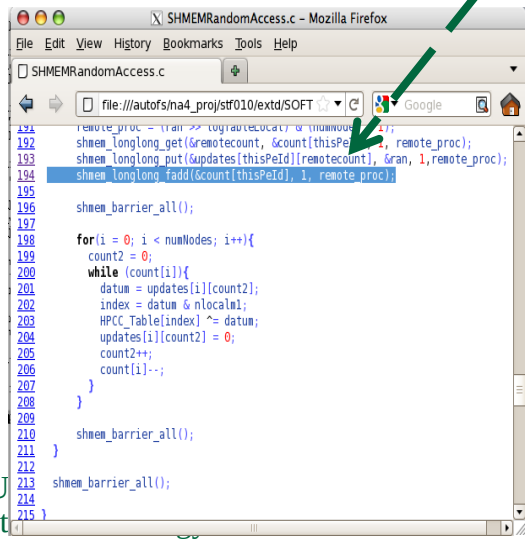
Node Legend:

- Calls OpenSHMEM
- OpenSHMEM
- Other

OpenSHMEM analysis

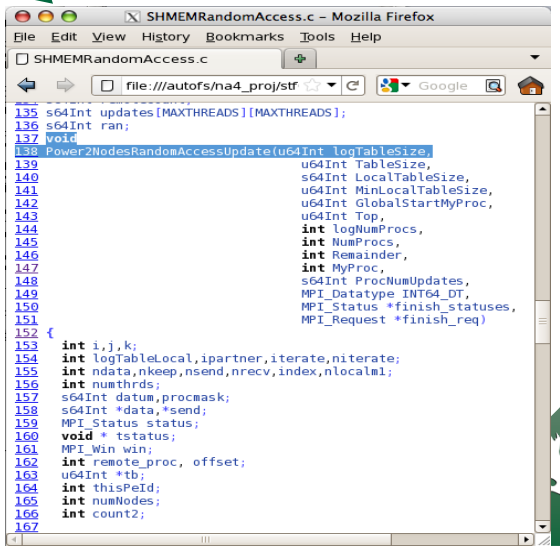
- **Hypertexted OpenSHMEM callgraph**





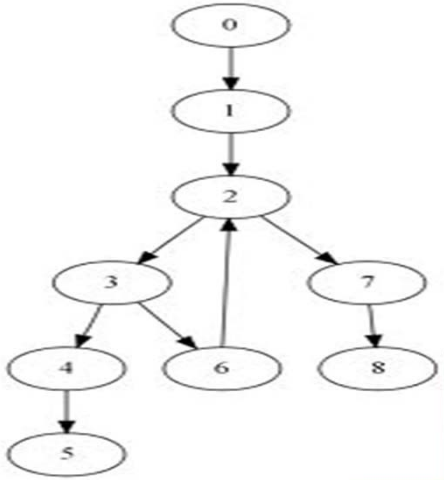
Callsite

Procedure



SHMEM Callgraph Analysis

- **Is start_pes() / shmem_init present in the application?**
- **If yes, is it present before other OpenSHMEM calls?**
- **start_pes should be a dominator of other OpenSHMEM calls.**
- **What about inter-procedural files?**
  - **Flow insensitive analysis**
  - **Traverse callgraph nodes in pre-order**



OpenSHMEM Initializations checks

- **main (..) {**

  …

  sub();

  …

  }

- **sub (..) {**

  sub1();

  sub2();

  }

sub1 (..) {

shmem_finalize();

}

sub2(..) {

shmem_init()

}

- **Tool will detect out of order calls inter-procedural**
- **Multiple instances of calls**
- **OpenSHMEM called outside**

How-to-use: multi-init.c

```
int main (int argc, char **argv)
{
  start_pes (0);
  start_pes (0);
      printf ("Hello from multi-init test\n");

   return 0;
}
```

- Multiple start_pes()
- No shmem_finalize()

How-to-use:

$cd startpes-ipa

$ shmemcc -ipa test-startpes-ipa.c

***** OpenSHMEM Warning: more than one OpenSHMEM initialization call found *****

Examples: Symmetric Errors

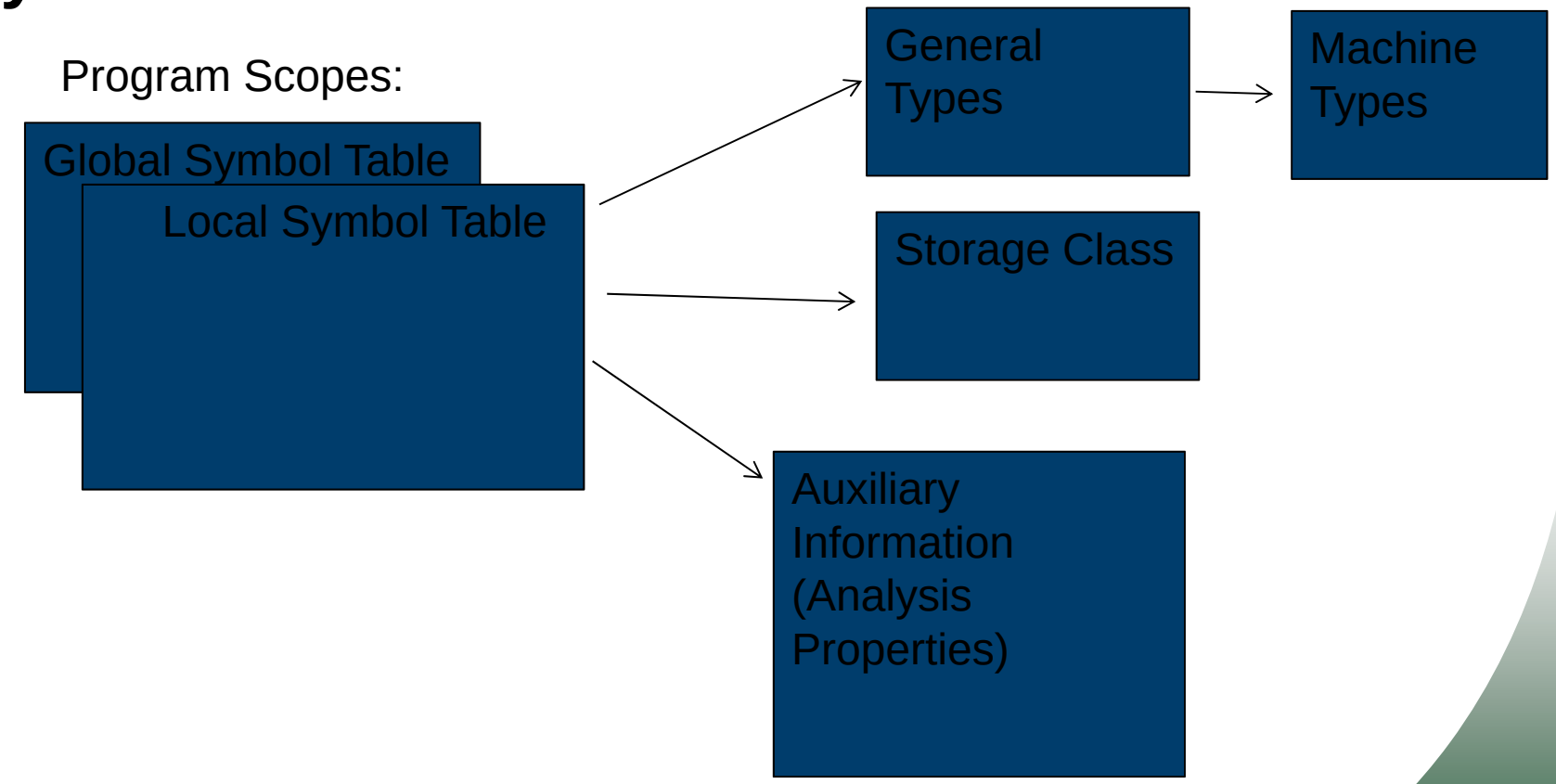
- **Storage checking:**

```
#ifdef N 64

int main ()  {

    long dest[N], src[N]; //error: dest non-
  symetric

 <….>

    shmem_long_put(dest, src, 64, 1);
```

- **Check if variables are symmetric.**
  - **Check within storage tables of compiler**
  - **Differentiate if variables are pointers or arrays**
    - **Querying compiler's symbol tables.**

SHMEM storage/type checks

- **A variable is represented in a compiler this way:**



Example: badget.c

```
Int main (int argc, char **argv) {
  long dest, src;
  int me, npes;
  start_pes (0);

  me = _my_pe ();
  npes = _nu_pes ();
  src = 42;
  shmem_barrier_all ();
  if (me == 0)  {
     shmem_long_get (&dest, &src, 1, 1);
   }
  shmem_barrier_all ();
  return 0;
}
```

Evaluation

$ cd badget

$ shmemcc -ipa badget.c

***** OpenSHMEM Warning: non-symetric variable in arg2 of shmem_long_get (line=65, file=badget.o) *****

Example: Out of bounds checking

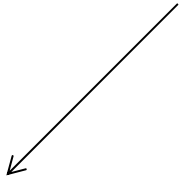
```
#ifdef N 64

int main ()  {

    long dest[N], src[N]; //error: dest non-
  symetric

 <….>

    shmem_long_put(dest, src, N+2, 1);
```

- **Check if variables are accessed within bounds.**
  - **Works for array accesses for both dest and src**
  - **Depends if constant propagation is known for length of accesses**

Evaluation: test-bounds.c

```
int main(void) {
     int i, src[N];
     long lget[N];
     static int targ_static[N];

     start_pes(0);

     for(i=0; i< N; i++) {
       src[i] = my_pe() + i;
       froml[i] = my_pe() + i*i;
     }

     shmem_int_put(targg, srcg, N+M, 2);
     shmem_int_put(targ_static, src, N+M, 3);
     shmem_long_get(lget,froml,N+M,4);

     shmem_barrier_all();  /* sync sender and receiver */
     return 1;

}
```

Out of bounds accesses of arrays of size N



Evaluation

- **%cd bounds**
- **%shmemcc -ipa test-bounds.c**

***** OpenSHMEM Warning: out of bounds access of shmem_int_put arg1 of 8 elements with access of 18 elements (line=20, file=test-bounds.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_int_put arg2 of 8 elements with access of 18 elements (line=20, file=test-bounds.o) *****

- **…..**

Example: Out of bounds checking with constant propagation

- **Since out of bounds checks relies on constants being propagated correctly.**
- **The OpenSHMEM analyzer can perform:**
  - **Constant Folding**
  - **Inter-procedural constant propagation**
  - **Common expression elimination that evaluate to constants**

Evaluation: test-bounds-const.c

```
int main(void) {
    int i, src[N], targ[N], len=N;
    long lget[N];
    static int targ_static[N];
    start_pes(0);

    for(i=0; i< N; i++) {
      src[i] = my_pe() + i;
      froml[i] = my_pe() + i*i;
     len++;
   }
   len++;
    shmem_int_put(targg, srcg, len, 2);
    shmem_int_put(targ_static, src, len, 3);
    shmem_long_get(lget,froml,len,4);
.....
}
```

- len is a variable that is initialized to N
- len is increased by N
- len is increased by 1

Out of bounds access with len= 2N+1

Evaluation: test-bounds-const.c

- **%cd bounds-constprog**
- **%shmemcc –ipa test-bounds-const.c**

***** OpenSHMEM Warning: out of bounds access of shmem_int_put arg1 of 16 elements with access of 33 elements (line=23, file=test-bounds-constprog.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_int_put arg2 of 16 elements with access of 33 elements (line=23, file=test-bounds-constprog.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_int_put arg1 of 16 elements with access of 33 elements (line=24, file=test-bounds-constprog.o) *****

Examples: Out of bounds in strided data accesses

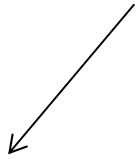
```
#ifdef N 10

int main ()  {

    static long dest[N],

    long src[N];

    shmem_long_iput(dest, src, 100, 2, 5 1);
```

- **Check if variables are within range**
  - **Check if access to variables are out of range**
  - **Perform Constant propagation and check if len is constant**

Evaluation

```
#define N 7
short src1[N];
int src2[N];
long src3[N];
…

Int main () {
….
    shmem_barrier_all();

     shmem_iget32(dest2, src2, 1, 2, N, npes-1);
     shmem_iget64(dest3, src3, 1, 2, N, npes-1);
     shmem_iget128(dest4, src4, 1, 2, N, npes-1);

….
```

Out of bounds accesses of for strided accesses



Evaluation (Bug in Validation Suite)

- **%cd iget-global**
- **%shmemcc -ipa test_shmem_get_globals.c**

***** OpenSHMEM Warning: out of bounds access of shmem_iget32 arg2 of 7 elements with access of 14 elements (line=297, file=test_shmem_get_globals.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_iget64 arg2 of 7 elements with access of 14 elements (line=298, file=test_shmem_get_globals.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_iget128 arg2 of 7 elements with access of 14 elements (line=299, file=test_shmem_get_globals.o) *****

***** OpenSHMEM Warning: out of bounds access of shmem_short_iget arg2 of 7 elements with access of 14 elements (line=395, file=test_shmem_get_globals.o) *****

……..

OpenSHMEM Data Flow problems

- **The tool will provides static analysis to respond questions:**
  - **How data is being used and initialized?**
  - **Framework for data flow information.**
    - **Intra and inter-procedural data flow**
  - **How to construct use-def chains that are OpenSHMEM aware?**
    - **For example locate uninitialized variables in OpenSHMEM calls**
    - **Pointers that get propagated to constants or invalid addresses (i.e. Null)**
    - **How to bind use-define of variables with correct memory allocators?**
  - **How to make the analysis accurate to be flow sensitive, process sensitive, etc.**

Check for Allocation of Data

```
int main () {
   src = (long *) shmalloc (N * sizeof (long));
 <…>
   shmem_long_get(targ, src, N, 1);
<…>
}
```

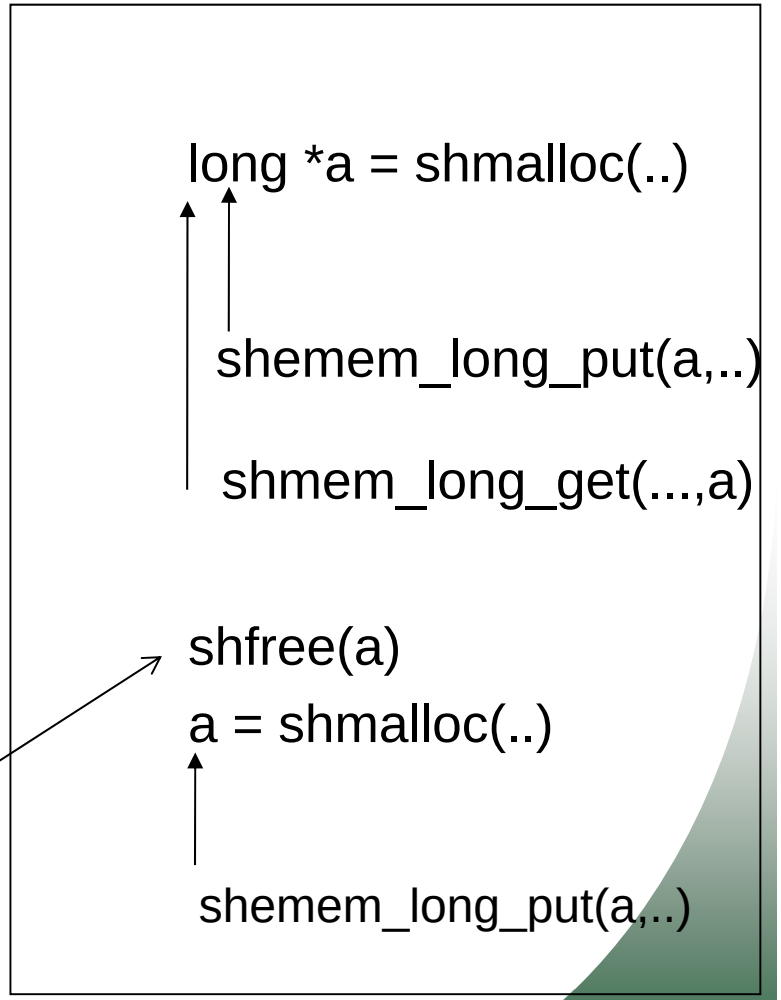
- Have data been allocated?
- Have data been initialized?
- Static vs. Dynamic testing

IPA Dataflow analysis

- **Traverse intermediate representations of each compilation unit**
- **Mark variables accessed via OpenSHMEM calls with extra field in symbol tables**
- **Aggregate global tables and symbols procedures at “ipa_link” time**
- **Perform Constant Propagation**
- **Check for allocation calls of variables and mark them**
- **Use data flow analysis to check use-def chains**

Constructing Use-Def Chains for OpenSHMEM calls

- Each use must be defined by 1 and only 1 def
- Straight-line code trivially single-assignment
- Uses-to-defs: many-to-1 mapping
- Each def dominates all its uses



Challenges for Pointer Analysis

- **Pointers are represented by scalars in compilers**
- **Accesses in OpenSHMEM calls for pointers can be complex expressions with address calculations**
- **Common expression elimination might be needed to simplify analysis**
  - **This is sometimes used for re-computation of addresses**
- **Pure scalars variables can affect the address calculation and might be uninitialized**

Example: Complex pointer usage

```
....
float *x, *y;
static float xs[100],ys[100];
  x = (float*)shmalloc((n_local1 - n_local0 + 2)*sizeof(float));
  y = (float*)malloc((n_local1 - n_local0 + 2)*sizeof(float));
  x -= (n_local0 - 1);
  y -= (n_local0 - 1);
shmem_barrier_all();

//... // fill x, y

// fill ghost zone
if (_my_pe() > 0)
 shmem_float_get(ys,xs,n1,1); // extra code
 .....
if (/*_my_pe() < _num_pes()-1 */(int)y[2])
 shmem_float_put(&y[n_local0-1], &y[n_local1-1], 1, _my_pe()+1);
shmem_barrier_all();
```

Initialization of pointers
*Wrong initialization of y

Address calculation, address is substracted by n_local0 -1

Address accessed by adding n_local0 -1

We need to check also if n_local0 has been initialized and if it is part of a pointer expression.

How pointer is represented in IR

- **shmem_float_put(&y[n_local0-1], …);**
- **IR representation:**

INTCONST(-1)

LOAD n_local

Need to check:
If variable is part of a pointer address calculation
If variable has been properly initialized

SUB

LOAD y

ADD

Need to check if this
Variable is of type pointer

PARAM 0

CALL shmem_float_put

Example: Complex pointer usage

```
....
float *x, *y;
static float xs[100],ys[100];

  y = (float*)malloc((n_local1 - n_local0 + 2)*sizeof(float));
  x -= (n_local0 - 1);
  y -= (n_local0 - 1);
shmem_barrier_all();


//... // fill x, y


// fill ghost zone
if (_my_pe() > 0)
 shmem_float_get(ys,xs,n1,1);
 .....
if (/*_my_pe() < _num_pes()-1 */(int)y[2])
 shmem_float_put(&y[n_local0-1], &y[n_local1-1], 1, _my_pe()+1);
shmem_barrier_all();
```

UD n_local1
UD n_local0

UD y1 (SSA)
UD n_local0

UD y2 (SSA)

UD n_local0

Evaluation:

- **%cd heap**
- **%shmemcc –ipa test-shmem_heap.c**

***** OpenSHMEM Warning: Local pointer in arg2 of OpenSHMEM call (line=24, file=test-shmem_heap.c) initialized with malloc *****

***** OpenSHMEM Warning: Local pointer in arg1 of OpenSHMEM call (line=26, file=test-shmem_heap.c) initialized with malloc *****

Global pointers

- **For the case of global pointers, we keep track of all global pointer intializations per procedures**
- **We summarize their local access in a table and mark them initialized.**
- **If global pointer is not in the table and accessed in an OpenSHMEM call a warning is generated.**

Evaluation:

```
#include <shmem.h>
#include <stdlib.h>
int n1=101;
float *x, *y;
void variable_allocation(int n) {
 int nn = (n-1) / _num_pes();
  int n_local0 = 1 + _my_pe() * nn;
  int n_local1 = 1 + (_my_pe()+1) * nn;
//  y = (float*) shmalloc((n_local1 - n_local0 + 2)*sizeof(float));  // forgot to initialize pointer

}

int main(….) {
m = ..
variable_allocation(m)
shmem_float_put(y,x, 1, _my_pe()-1);

}
```

Evaluation:

- **%cd heap-global**
- **%shmemcc –ipa test-shmem_heap-global.c**

***** OpenSHMEM Warning: global variable arg1 of call shmem_float_put is uninitialized line=20, file=shmem_heap-global.0) *****

Other Data Flow problems: going beyond scalars or array single accesses

- **How can we summarize data accesses at different granularity levels:**
  - **Loop data accesses**
  - **Basic Block Level at the Control Flow**
  - **Procedure level**
- **Alias analysis information: The tool is able to check if a symmetric variable is aliased.**

How to use OpenSHMEM Analyzer

- **Text-Based Analysis:**
  - **Use like another compiler with the option –ipa**
    - **If more aggressive analysis is require use –ipa –O3**
  - **shmemcc –ipa mytest.c**
  - **Use the default shmem.h provided by the Analyzer**
- **Callgraph Analysis**
  - **shmemcc –ipa mytest.c -o test**
  - **firefox test.html**

Makefile Example:

•Driver: shmemcc

```
%cat Makefile
CC =  shmemcc
CXX = shmemCC
CFLAGS = -ipa
LDFLAGS = $(CFLAGS)
TARGET  = ra_shmem
OBJECTS = RandomAccess.o SHMEMRandomAccess.o verification.o

.SUFFIXES: .c
.c.o:
        $(CC) $(INCDIR) $(CFLAGS) -c $<
        code2html -l C -n -N $< $*.html

all:  RASHMEM
RASHMEM: $(OBJECTS)
        $(CC) $(INCDIR) $(CFLAGS) $(OBJECTS) -o $(TARGET) $(LDFLAGS)
        callgraph ra_shmem &> index.html
        firefox index.html
```

- Makefile modifications
- Use the tool as a regular compiler
- It includes OpenSHMEM (shmem.h)
- Uses a special version of SHMEM library that can used for IPA.
- It will not generate executables

•Converts sources to HTLM format

•Provides error analysis

•Builds graphical callgrph

How to Install

- **Installation instructions:**

- **Install code2html 0.9.1**
- **http://www.palfrader.org/code2html/**
- **Use current .tar.gz file)**

- **Note: Make sure to add the path to 'code2html' to your $PATH**

- **Install graphviz 2.28.0**
- **http://www.graphviz.org/**
- **Use current .tar.gz file**

- **Recommended Configuration: ./configure --prefix=<install directory> --enable-python=no**
- **Note: Make sure to add the path to 'dot' to your $PATH**

How to Install (2)

- **Go to www.openshmem.org/OSA**
- **tar -xzvf openshmem-analyzer-1.0.tar.gz**
- **cd openshmem-analyzer-1.0/build**
- **../configure --prefix=<openshmem analyzer install directory> --disable-host_bdver1-support --with-build-optimize=DEBUG**
- **gmake**
- **gmake install**
- **Add <openshmem analyzer install directory>/bin to your $PATH**

Supported Environments

- **Processors: IA32, X86_64 with**
- **Supports 32/64-bits builds**
- **SLES 11 SP1, RHEL 6 (GLIBC 2.11 and above) [Development]**
- **SLES 10 SP2, SLES 10 SP3, RHEL 5.5**

Acknowledgements

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.