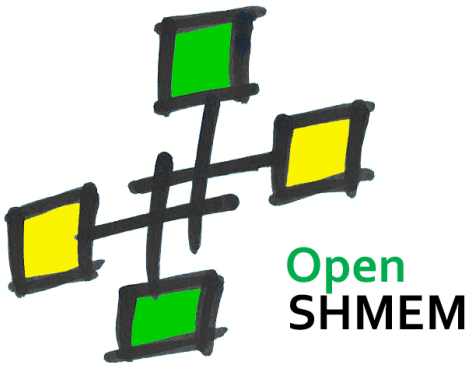


Thursday, March 6, 2014

Extending the OpenSHMEM Analyzer to perform synchronization and multi-valued analysis

Swaroop Pophale, Oscar Hernandez, Stephen Poole, and Barbara Chapman

Outline

- Motivation
  - Discovering Synchronization Phases
- Introduction
  - OpenUH compiler
  - OpenSHMEM Analyzer (OSA)
- Concurrency Analysis
- Implementation
- Results
- Limitations
- Summary

Motivation

- The OpenSHMEM Analyzer (OSA) is a compiler-based tool to help the user correctly use the OpenSHMEM library API.
  - Paper: Oscar, H., Siddhartha, J., Pophale, S., Stephen, P., Kuehn, J., Barbara, C.: The OpenSHMEM Analyzer. In: Proceedings of the Sixth Conference on Partitioned Global Address Space Programming Model. PGAS '12 (2012)

- Extending the OpenSHMEM to perform concurrent analysis
  - In OpenSHMEM you can write syntactically correct parallel programs but that are semantically wrong are possible.
  - Optimizations are often limited to portions of code between barriers.
  - Synchronization errors can lead to race conditions and deadlock.

Discovering Synchronization Phases

- Extending OSA to discover the synchronization phases of an OpenSHMEM program
  - Important for memory consistency and control points in an application
  - Two forms of collective synchronization exists in OpenSHMEM
    - shmem_barrier_all is defined over all PEs.
    - shmem_barrier is defined over an ***active set***.
      - An active set is a logical grouping of PEs based on three parameters (passed as arguments) namely, PE start, logPE_stride and PE size triplet .
  - OpenSHMEM specification allows unaligned barriers.

- Optimizations are possible only between global synchronization calls.
- Same concepts are applicable to other PGAS languages/libraries.

Textually Unaligned Barriers

- Textually Non-aligned vs. Aligned code

```
if (me%2==0){
  …
  shmem_barrier_all();
}
else{
  …
  shmem_barrier_all();
}
```

```
if (me%2==0){
  …
  …
}
else{
  …
  …
}
shmem_barrier_all();
```

- BOTH ARE VALID IN OPENSHMEM

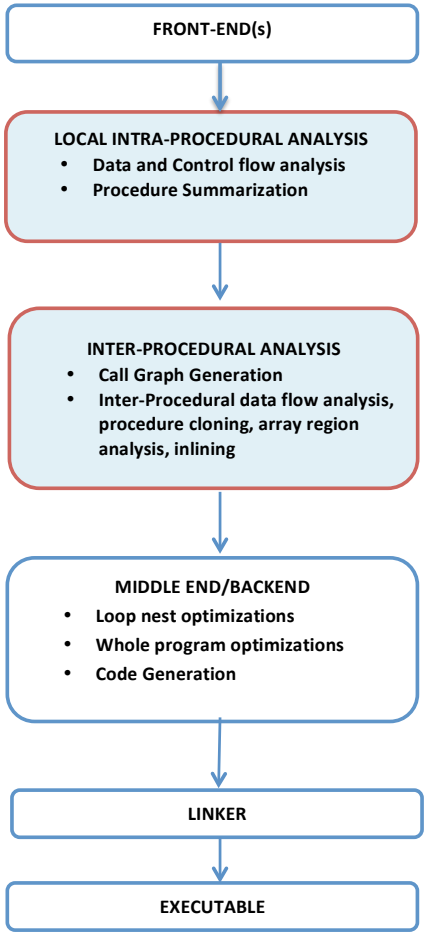
Introduction: OpenUH Compiler

- Open source research compiler based on Open64
- Complete support for OpenMP 2.5 (in C/C++ & Fortran) and CAF
- Stable, portable, modularized with complete optimization framework
- Available on most Linux platforms

Introduction: OSA

- OpenSHMEM Analyzer (OSA) extends the existing compiler technology to report errors accurately in context of C and OpenSHMEM.
- Provided basic syntactic and semantic checks.
- Verifies at compile time that all OpenSHMEM library calls are using the appropriate classes of data as required by Specification 1.0.
- Mainly focuses on IPA phase of the OpenUH compiler.

OSA within OpenUH



Concurrency Analysis

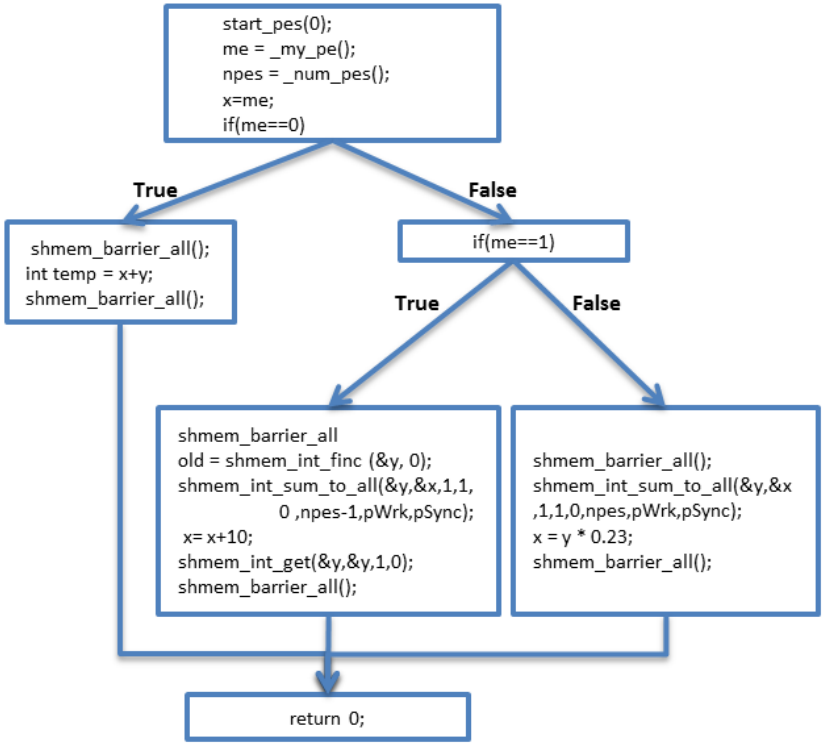
- All PEs execute the application in SPMD style
- Different PEs may take different execution paths based on the implicit or explicit conditions set by the programmer.
- Variables that have different values on different PEs.
  - **Multivalued seed**
- Multi-Valued Expressions
  - "An expression is multi-valued if it evaluates differently in different threads."

Implementation

- Step 1: Identification of multivalued seeds and multivalued expression
  - Analyze Control and Data flow
  - Construct System dependence graph
  - Analyze effect of OpenSHMEM calls on multivalued seeds
- Step 2: Detection of OpenSHMEM barriers and their ordering
- Step 3: Barrier tree generation and barrier matching

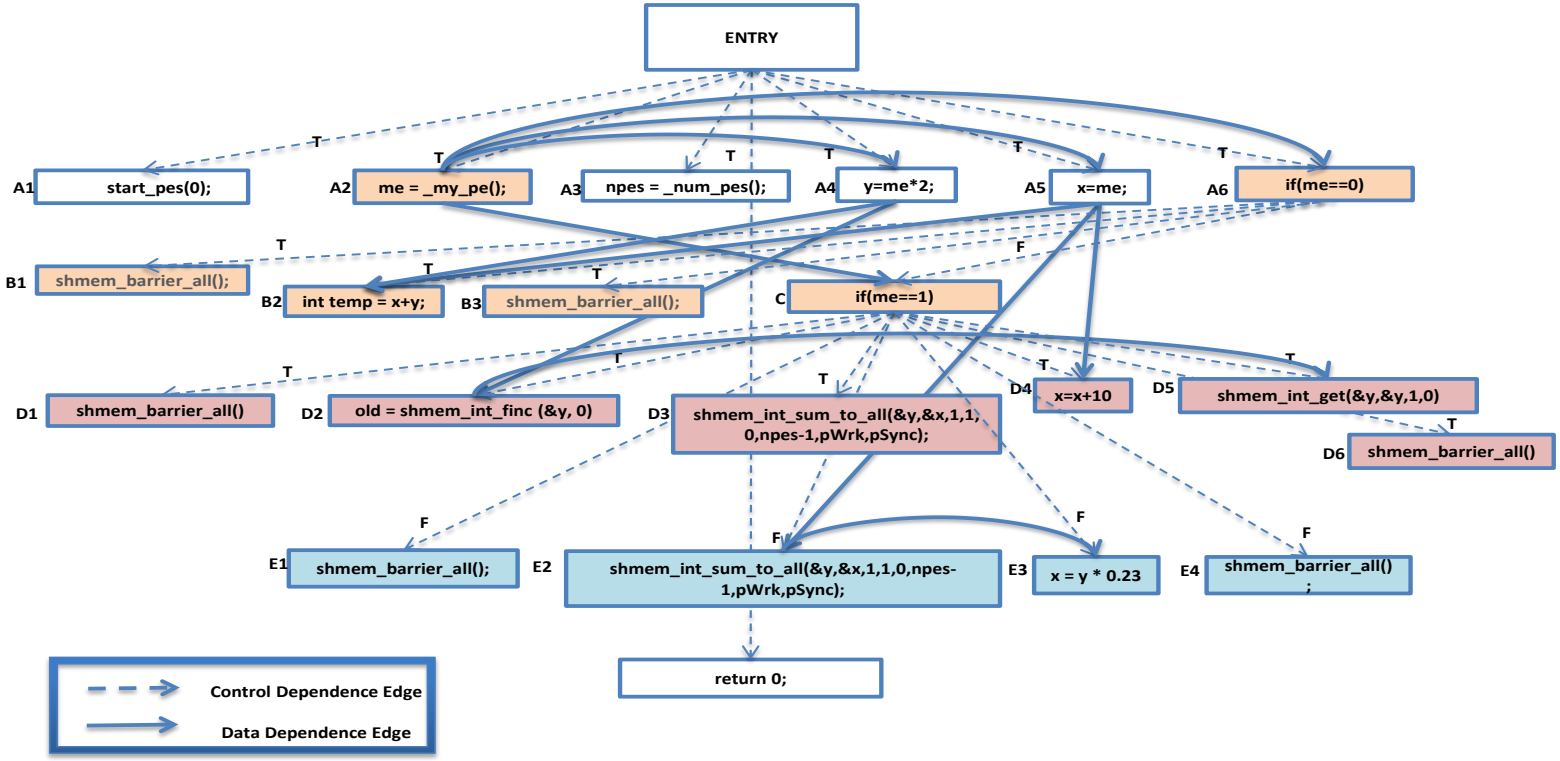
Step 1: Control Flow and Data Flow

```
int main(int argc, char *argv[]){
 …
   if(me==0){
      shmem_barrier_all();
      int temp = x+y;
      shmem_barrier_all();
   }
   else {
     if(me==1){
         shmem_barrier_all();
         old = shmem_int_finc (&y, 0);
         shmem_int_sum_to_all(&y,&x,
                1,1,0,npes-1,pWrk,pSync);
          x= x+10;
         shmem_int_get(&y,&y,1,0);
         shmem_barrier_all();
        }
     else{
         shmem_barrier_all();
         shmem_int_sum_to_all(&y,&x,
                1,1,0,npes-1,pWrk,pSync);
         x=y*0.23
         shmem_barrier_all();
        }
   }
  return 0;
}
```



Control Flow Graph

Step 1: System Dependence Graph



Step 1: Multi-valued and OpenSHMEM

- *Multi-valued-ness* depends on the semantics of OpenSHMEM and its treatment of different program variables.

OpenSHMEM Library Operations	Variable Affected	Effect
_num_pes	*npes*	Single-valued
_my_pe	*me*	Multi-valued
PUT (elemental, block, strided)	*target*	Multi-valued
GET (elemental, block, strided)	*target*	Multi-valued
ATOMICS (fetch and operate)	*target*	Multi-valued
BROADCAST	*target*	Multi-valued
COLLECTS (fixed and variable length)	*target* array	Single-valued if *active set* = npes else Multi-valued

TABLE I
EFFECT OF OPENSHMEM LIBRARY CALLS ON PROGRAM VARIABLES

Step 2: Barrier Detection

- We analyze the CFG of the program to determine the presence and relative ordering of the barriers.
- Based on the multivalued analysis we mark points at which the execution paths diverge.
- Mark conditionals with appropriate operator:

Placement of barriers	Operator used	Result
b1 followed by b2	·	b1 · b2
if((*single-valued*) *conditional*) b1 else b2;	|	b1 | b2
if((*multivalued-valued*) *conditional*) b1 else b2;	$\|^c$	b1 $\|^c$ b2

TABLE II
RULES FOR BUILDING A BARRIER EXPRESSION

Step 3: Barrier Tree Generation and Barrier Matching

- Generating a barrier tree
  - Iterate over the AST
  - Analyze conditionals
  - Determine relationship
  - Output using graphviz

- Use barrier tree to match unaligned barriers
  - Traverse barrier tree to match barriers
  - Gives line number information where potential unmatched barrier is found.

Results

- Application: Matrix Multiplication

- The application consists of three 2-D arrays A, B, and C, where C is used to store the product of two matrices A and B.

Output (1)

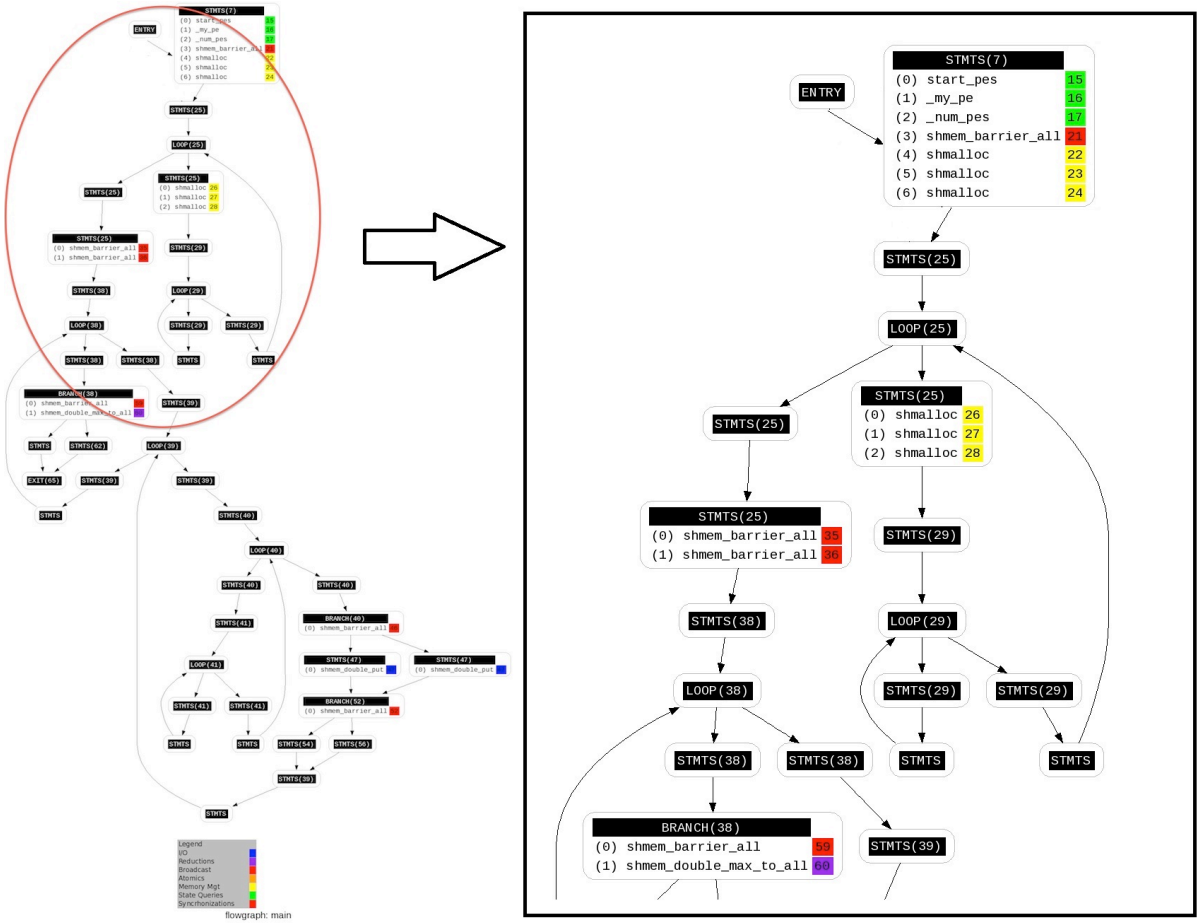


Figure: CFG with OpenSHMEM calls

Output (2)

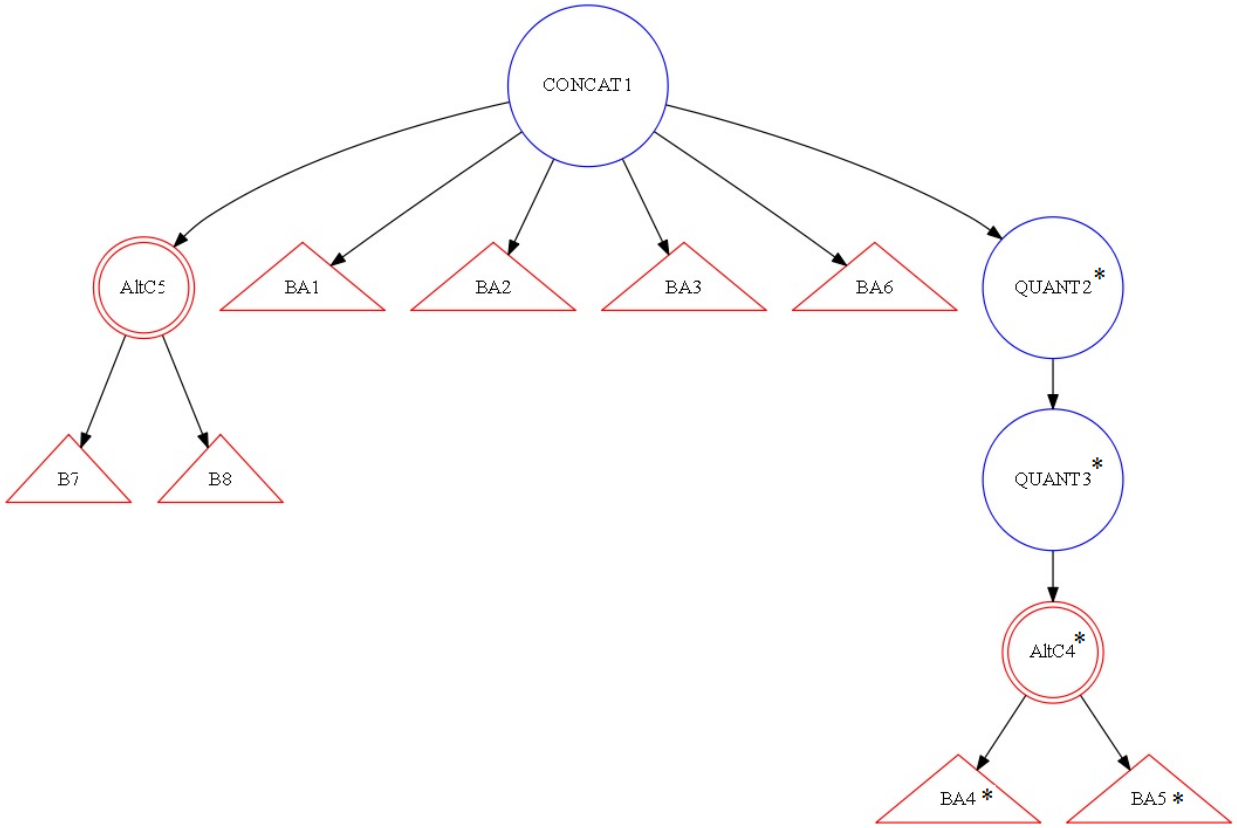


Figure: Barrier Tree generated for Matrix Multiplication benchmark

Limitations

- Static analysis tool
- shmem_barrier statements are tricky
  - active set parameters
  - Solution: explicit active sets

Summary

- OSA combines data and control flow information to determine concurrent paths in OpenSHMEM application.
- Barriers are points in OpenSHMEM applications that guarantee memory consistency.
- Unaligned barriers are a challenge for application programmers.
- OSA provides visual feedback of analysis to application programmer to better understand the program structure and synchronization structure of the program.

Acknowledgement

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