Combining Static and Dynamic Analysis for Top-down Communication Trace Compression

Jidong Zhai
Tsinghua University
Communication Traces

- Communication Traces highly useful:
  - Communication performance analysis
    - Bottleneck/Bug detection
    - Platform selection
    - Communication optimization
  - Design future HPC systems

Segment of MPI communication trace
Analyzing Communication Patterns

Communication patterns extracted from MPI communication traces.

Predicting Future HPC Systems


MUG’18-Combining Static and 26 Analysis for Top-down Communication Trace Compression
Tracing Communication: Challenges

• Problem: Traces $\rightarrow$ Huge
  • Caused by growing
    • Problem size
    • job scale (# of processes)
  • Example:
    • ASCI SMG2000, 64*64*32 Problem size,
      22,538 Processes $\rightarrow$ 5 TB Traces$^1$

• Huge traces bring
  • Pressure on storage system
  • Interference with user application execution

Solution: Trace Compression

• Significant **redundancy**
  - Within a process: iterative timesteps (loops)
  - Across processes: similar behavior (SPMD)

• **Existing tools: Bottom-up approach**
  - ScalaTrace\(^1\), ScalaTrace-2\(^2\)
  - Examine traces at runtime to discover repeated patterns
  - **Two-phase**: intra-process + inter-process

---


Bottom-up 1: Intra-Process Compression

- Identify repeated pattern within one process

Repeated pattern

Communication traces after intra-process compression

Communication traces for each process.
Bottom-up 2: *Inter*-Process Compression

- Compare traces in horizontal dimension to identify repeated patterns

**Communication traces after inter-process compression**
Limitations of Bottom-Up Compression

- **Intra-Process:**
  - Expensive with complex communication patterns
  - Nested loop structures[1]

MPI Program snippet

```c
1 for (i=0; i<10; i++)
2  for (j=0; j<=i; ++j) {
3    if (j%2 == 0)
4      MPI_Isend(...);
5    else
6      MPI_Irecv(...);
7  }
8  MPI_Waitall(...);
9  MPI_Reduce(...);
10 }
```

Communication trace segment

```
1  MPI_Isend(...)  
2  MPI_Waitall(...) 
3  MPI_Reduce(...)  
4  MPI_Isend(...)  
5  MPI_Irecv(...)  
6  MPI_Waitall(...) 
7  MPI_Reduce(...)  
8  MPI_Isend(...)  
9  MPI_Irecv(...)  
10 MPI_Isend(...) 
11 MPI_Waitall(...) 
12 MPI_Reduce(...) 
```

Limitations of Bottom-Up Compression

- **Inter-Process:**
  - Even more expensive, not scalable
  - Complexity for two processes $\rightarrow O(n^2)$

```
4*MPI_Send
3*MPI_Recv
5*MPI_Bcast
3*MPI_Recv
......
```

```
3*MPI_Isend
4*MPI_Irecv
5*MPI_Wait
3*MPI_Barrier
......
```

```
4*MPI_Send
3*MPI_Recv
5*MPI_Bcast
3*MPI_Recv
......
```

```
3*MPI_Isend
4*MPI_Irecv
5*MPI_Wait
3*MPI_Barrier
......
```

```
Compressed traces of P_0 and P_1
```
Our Approach: CYPRESS

• *Top-down* technique
  
  ● Combines *static* and *dynamic analysis*
  
  ● **Static analysis phase**
    
    • Extract program communication structure tree
      – Loops, branches etc.

  ● **Dynamic analysis phase**
    
    • Use this tree as a template
    
    • “Fill in” runtime information
      – Loop count
Overview of CYPRESS

An MPI Program

Intra-Procedural Analysis

Inter-Procedural Analysis

A Compiler Plug-in

Static Analysis Module

Program communication structure tree (CST)

A HPC system

Dynamic Analysis Module

Compressed communication traces

Program CST

An MPI Binary
CYPRESS: Advantages

• Much faster
  • Top-down “Big picture”
    • Redundancy expected
    • Cut the guessing
  • More efficient
    • Knows “where to stop”

• Scalable
  • Significant improvement to Inter-process compression
  • A better worst-case complexity
  • More affordable for large-scale analysis
CYPRESS: Static Analysis

An MPI Program

Intra-Procedural Analysis

Inter-Procedural Analysis

A Compiler Plug-in

Static Analysis Module

Program communication structure tree (CST)

A HPC system

Dynamic Analysis Module

Compressed communication traces

Program CST

MUG’18-Combining Static and 26 Analysis for Top-down Communication Trace Compression
Communication Structure Tree (CST)

- Ordered tree
  - Program communication structure
  - Non-leaf nodes:
    - Loop
    - Branch
  - Leaf nodes:
    - MPI communication calls
      » MPI_Send, MPI_Recv
1 int main(){
2   for (i=0; i<k; i++){  
3       if (myid % 2 == 0)
4           MPI_Send(buf, size, MPI_INT,  
5               myid+1, 0, MPI_COMM_WORLD);
6       else
7           MPI_Recv(buf, size, MPI_INT,  
8               myid-1, 0, MPI_COMM_WORLD, &status);
9       bar();
10   }
11   foo();
12   if (myid % 2 == 0)
13       MPI_Reduce(sbuf, rbuf, 1, MPI_INT,  
14               MPI_SUM, root, comm);
15 }
16 int bar(){
17   for(k=0; k<n; k++)
18       MPI_Bcast(buf, size, MPI_INT, root, 18  
19           MPI_COMM_WORLD);
20 }
21 int foo(){
22   for(j=0; j<m; j++)
23       sum += j;
24 }
CYPRESS: Dynamic Analysis

An MPI Program

Intra-Procedural Analysis

Inter-Procedural Analysis

A Compiler Plug-in

Static Analysis Module

Program communication structure tree (CST)

A HPC system

Dynamic Analysis Module

Compressed communication traces

Program CST
Intra-Process Compression

• “Fill in” dynamic information into CST
  ● Record loop count and branch taken status
  ● Per-node linked list for MPI event details

Fill in dynamic information into the CST.
Inter-Process Compression

• Compare corresponding nodes of the trees

A compressed trace tree for P₀

A compressed trace tree for P₁
Experiments

• **Benchmarks**
  - 8 NPB programs: BT, CG, DT, EP, FT, LU, MG, SP
  - Real Application: LESlie3d

• **Platform**
  - Explorer-100 HPC cluster
    - 2*Intel Xeon X5670/48GB Memory
    - Infiniband network, MVAPICH-2

• **Alternative systems**
  - Gzip (offline compression)
  - ScalaTrace (online, bottom-up)
  - ScalaTrace-2 (online, bottom-up)

• **Metrics**
  - Compression effectiveness (trace size reduction)
  - Compression overhead
Compressed Trace Size

Compressed communication trace size in log scale (KB).

MUG'18-Combining Static and 26 Analysis for Top-down Communication Trace Compression
Intra-Process Compression Overhead

ScalaTrace: 51.05%
ScalaTrace-2: 9.1%
CYPRESS: 1.58% ↓5X over ScalaTrace-2

MUG'18-Combining Static and 26 Analysis for Top-down Communication Trace Compression
Inter-Process Compression Overhead

**Average inter-process compression overhead:**

- **ScalaTrace:** 170.69%
- **ScalaTrace-2:** 30.3%
- **CYPRESS:** 3.29% ↓9X over ScalaTrace-2
Overall Compression Overhead

- Overhead breakdown

<table>
<thead>
<tr>
<th>Programs</th>
<th>BT</th>
<th>CG</th>
<th>DT</th>
<th>EP</th>
<th>FT</th>
<th>LU</th>
<th>MG</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>0.25</td>
<td>0.11</td>
<td>0.05</td>
<td>0.09</td>
<td>0.05</td>
<td>0.16</td>
<td>0.09</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Extra compilation time incurred by Cypress for each program (second)
Conclusion

• **CYPRESS: Top-Down trace compression**
  - Leverage *static information* to facilitate more effective and efficient trace compression
  - Reduce compression overhead
    - Intra-process by 5X, inter-process by 9X
  - Strong lossless compression performance

• **Take-away Message: static information useful**
  - Provides reliable insight
  - Relatively cheap, with scale-independent cost

• Analyze memory access patterns:
Thanks!

Contributors: Jianfei Hu, Liwei Chen, Jidong Zhai, Xiongchao Tang, Xiaosong Ma, Wenguang Chen