

Deductive Verification of Parallel Programs Using Why3

César Santos, Vasco Vasconcelos, Francisco Martins

LaSIGE, Faculty of Sciences of the University of Lisbon

July 5, 2015

Outline

- 1 Introduction
- 2 Approach
- 3 Protocol language
- 4 Programming language
- 5 Results and conclusions

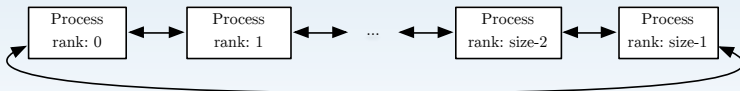
MPI



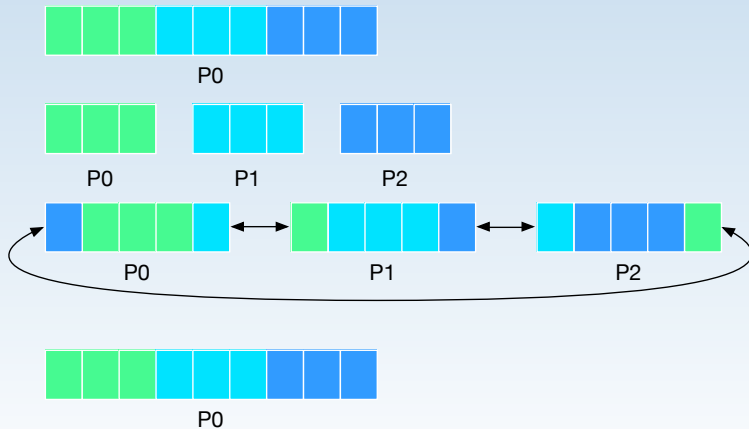
- Message-based specification for parallel computing
- Industry standard (C/Fortran libraries)
- Single-Program Multiple-Data (SPMD)
- Every process is issued a rank (process number)
- Point-to-point and collective communication
- Used for simulations that require a lot of computational power

Example: Finite differences

- Numeric method for solving differential equations
- The program starts with an initial solution X_0 , and calculates X_1, X_2, X_3, \dots iteratively until a maximum number of iterations are executed
- Processes are organized in a ring topology



Example: Finite differences



Example: Finite differences

```

1  int main(int argc, char** argv) {
2      MPI_Init(&argc, &argv);
3      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
4      MPI_Comm_size(MPI_COMM_WORLD, &size);
5      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
6      MPI_Scatter(data, n/size, MPI_FLOAT, &local[1], n/size, MPI_FLOAT, 0, MPI_COMM_WORLD);
7      int left = rank == 0 ? size - 1 : rank - 1;
8      int right = rank == size - 1 ? 0 : rank + 1;
9      for (iter = 1; i <= ITERATIONS; iter++) {
10         MPI_Send(&local[1], 1, MPI_FLOAT, left, 0, MPI_COMM_WORLD);
11         MPI_Send(&local[n/size], 1, MPI_FLOAT, right, 0, MPI_COMM_WORLD);
12         MPI_Recv(&local[n/size+1], 1, MPI_FLOAT, right, 0, MPI_COMM_WORLD, &status);
13         MPI_Recv(&local[0], 1, MPI_FLOAT, left, 0, MPI_COMM_WORLD, &status);
14         // Computation is performed here, removed for simplicity
15     }
16     MPI_Reduce(&localErr, &globalErr, 1, MPI_FLOAT, MPI_MAX, 0, MPI_COMM_WORLD);
17     MPI_Gather(&local[1], n/size, MPI_FLOAT, data, n/size, MPI_FLOAT, 0, MPI_COMM_WORLD);
18     MPI_Finalize();
19     return 0;
20 }

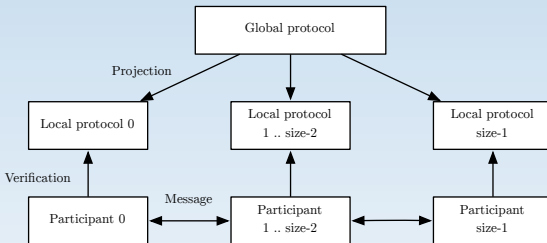
```

Does it deadlock? Is it communication type safe?

Challenges

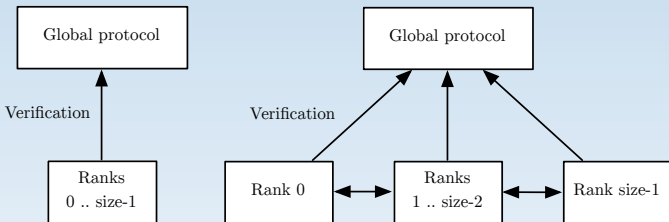
- Verifying parallel programs is difficult
 - State explosion problem
- Many verification tools only work at runtime
 - Dependent on the quality of the test set
 - Performance costs

Multi-party session types



- Theory for communication protocols
- Global protocol is projected for each process
- Properties guaranteed for program: absence of deadlocks and of communication errors
- Does not suffer from the state explosion problem

ParTypes



- Inspired by multi-party session types
- Point-to-point and collective communication, for loops and collective choices
- No separate projection step
- Can be used with both SPMD (like MPI) and MPMD programs

Protocol language

```
1 protocol FiniteDifferences (size >= 2) {
2     val iterations: natural
3     broadcast 0 n: {x: natural | x % size = 0}
4     scatter 0 float[n]
5     foreach iter: 0 .. iterations {
6         foreach i: 0 .. size-1 {
7             message i (size+i-1)%size float
8             message i (i+1)%size float
9         }
10    }
11    reduce 0 max float
12    gather 0 float[n]
13 }
```

Protocol compiler (Eclipse plugin)

Java - Fdiff/src/fdiff.prot - Eclipse SDK

fdiff.prot

```

1 protocol FiniteDifferences (size >= 2) {
2   val iterations: natural
3   broadcast 0 n: {x: natural | x % size = 0}
4   scatter 0 float[n]
5   foreach iter: 0 .. iterations {
6     foreach i: 0 .. size-1 {
7       message i (size+i-1)%size float
8       message i (i+1)%size float
9     }
10  }
11  reduce 0 max float gather 0 float[n]
12 }

```

Problems @ Javadoc Declaration

0 items

Description	Resource	Path	Location	Type

Writable Insert 3 : 20

Foreach expansion

i	Loop body	Rank 0	Rank 1	...	Rank size-2	Rank size-1
0	message 0 size-1 message 0 1	send size-1 send 1	recv 0			recv 0
1	message 1 0 message 1 2	recv 1	send 0 send 2			
2	message 2 1 message 2 3		recv 2			
...						
size-2	message size-2 size-3 message size-2 size-1				send size-3 send size-1	recv size-1
size-1	message size-1 size-2 message size-1 0	recv size-1			recv size-1	send size-2 send 0

Example: Finite differences

```
1 int main(int argc, char** argv) {
2   MPI_Init(&argc, &argv);
3   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
4   MPI_Comm_size(MPI_COMM_WORLD, &size);
5   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
6   MPI_Scatter(data, n/size, MPI_FLOAT, &local[1], n/size, MPI_FLOAT, 0, MPI_COMM_WORLD);
7   int left = rank == 0 ? size - 1 : rank - 1;
8   int right = rank == size - 1 ? 0 : rank + 1;
9   for (iter = 1; i <= ITERATIONS; iter++) {
10    MPI_Send(&local[1], 1, MPI_FLOAT, left, 0, MPI_COMM_WORLD);
11    MPI_Send(&local[n/size], 1, MPI_FLOAT, right, 0, MPI_COMM_WORLD);
12    MPI_Recv(&local[n/size+1], 1, MPI_FLOAT, right, 0, MPI_COMM_WORLD, &status);
13    MPI_Recv(&local[0], 1, MPI_FLOAT, left, 0, MPI_COMM_WORLD, &status);
14    // Computation is performed here, removed for simplicity
15  }
16  MPI_Reduce(&localErr, &globalErr, 1, MPI_FLOAT, MPI_MAX, 0, MPI_COMM_WORLD);
17  MPI_Gather(&local[1], n/size, MPI_FLOAT, data, n/size, MPI_FLOAT, 0, MPI_COMM_WORLD);
18  MPI_Finalize();
19  return 0;
20 }
```

This is going to deadlock

Why3

- Deductive software verification platform
- Specification language (Why) + programming language (WhyML)
- Verification conditions can be split in parts and proven with different SMT solvers (or with proof assistants such as Coq)
- Chosen to avoid the annotation overhead required for static verification of C/Fortran programs
- Experiment in programming methodology

Why3 IDE

The screenshot displays the Why3 Interactive Proof Session IDE. The interface is divided into several sections:

- Left Panel (Project Tree):** Shows a hierarchy of files and goals. Under 'Theories/Goals', there is a folder 'fdiff.mlw' containing 'VC for main', which in turn contains 'split_goal_wp'. Below this, a list of 30 goals is shown, each with a status icon (e.g., a yellow triangle with a red circle) and a label like '1. precondition'.
- Middle Panel (Tools):** Contains various tool buttons such as 'Split', 'Inline', 'Edit', 'Replay', 'Remove', and 'Clean'. Below these are 'Proof monitoring' options: 'Waiting: 0', 'Scheduled: 0', 'Running: 0', and an 'Interrupt' button.
- Right Panel (Code Editor):** Displays the source code for 'fdiff.mlw'. The code includes:
 - Header: `forall init. (fc @ n) = Scatter @ any_array_float`
 - Loop: `(Loop (ForEach @ [p - 1] >= 0 then i - 1 else p - 1) => protocol. forall i init. (forall i init. (Message i (if (i - 1) >= 0 then i - 1 else p - 1) any_float (Message i (if (i + 1) <= (p - 1) then i + 1 else 0) any_float Skip))) (AllReduce Max (CFloat (epsilon fcl:real -> bool. forall i:real. (fcl @ x) = True) (epsilon fcl:real -> protocol. forall usx4:real. (fcl @ usx4) = Skip))))))`
 - Constants: `constant max_size : int = 10000`, `constant max_iter : int = 10000`, `constant max_error : real = 0.0001`, `constant np : int = size`, `constant o : int = max_size`
 - Goal: `goal NP_parameter_main ; match head (head fdiff_protocol) with Val d -> matches o d -> false end`
 - Main Function: `let main () = let np = size in let s = init fdiff_protocol in let psize = apply max_size in let work = ref (make 0.0 max_size) in if rank = 0 then for i = 0 to [max_size-1] do work := (!work)[i] <- (from_int (random_int 100000)) /. 10000.0 done ; let lszie = div pszie np in let local = ref (make 0.0 max_size) in scatter @ !work ; let globalex = ref 999.0 in let iter = ref 0 in let left = (if rank = 1 >= 0 then rank-1 else np-1) in let right = (if rank = 1 <= np-1 then rank-1 else 0) in let lbody = inloop s in let rbody = copy lbody in while !globalex >= max_error at iter < max_iter do invariant { inbody = lbody } variant { max_iter - iter } let body = foreach inbody in if (rank = 0) then let f1 = project body 0 in send left (!local)[!f1] ; send right (!local)[!size] f1 ;`

Why3 theory for protocols

```
1  type protocol =
2      Val datatype
3      | Broadcast int datatype continuation
4      | Scatter int datatype protocol
5      | Gather int datatype protocol
6      | Message int int datatype protocol
7      | Reduce int op datatype protocol
8      | Skip
9      | AllGather datatype continuation
10     | Foreach int int (cont int) protocol
11     | AllReduce op datatype continuation
12 with
13     op = Max | Min | Sum | ...
```


WhyML library

- Inspired by MPI
- Point-to-point and collective communication
- Primitives annotated with pre and post-conditions
- Verification guided by the protocol
- Requires program annotations for loops and choices

Finite differences in WhyML

```
1  let main () =
2    let s = init fdiff_protocol in
3    let iterations = apply 100000 s in
4    let n = broadcast 0 input s in
5    let local = scatter 0 work s in
6    let left = (if rank > 0 then rank-1 else size-1) in
7    let right = (if rank < size-1 then rank+1 else 0) in
8    let inbody = expand (foreach s) rank in (* Annotation *)
9    for iter = 1 to iterations do
10      (* Loop body *)
11    done;
12    globalerror := reduce 0 Max !localerror s;
13    gather 0 local s;
14    isSkip s; (* Annotation *)
```

Fixed communication

```

1  let body = foreach inbody in (* Annotation *)
2  if (rank = 0) then (
3    let f1 = expand body 0 in (* Annotation *)
4    send left local[1] f1;
5    send right local[n/size] f1; isSkip f1; (* Annotation *)
6    let f2 = expand body 1 in (* Annotation *)
7    local[n/size+1] <- recv right f2; isSkip f2; (* Annotation *)
8    let f3 = expand body (np-1) in (* Annotation *)
9    local[0] <- recv left f3; isSkip f3); (* Annotation *)
10 else if (rank = size-1) then (
11   let f1 = expand body 0 in (* Annotation *)
12   local[lsize+1] <- recv right f1; isSkip f1; (* Annotation *)
13   let f2 = expand body (np-2) in (* Annotation *)
14   local[0] <- recv left f2; isSkip f2; (* Annotation *)
15   let f3 = expand body (np-1) in (* Annotation *)
16   send left local[1] f3;
17   send right local[lsize] f3; isSkip f3); (* Annotation *)
18 else (
19   ...
20 isSkip inbody; (* Annotation *)
21 (* Computation is performed here, removed for simplicity *)

```

Fixed communication

```
1 let body = foreach inbody in (* Annotation *)
2 if (rank = 0) then (
3     let f = expand body [size-1, 0, 1] in (* Annotation *)
4     send left local[1] f;
5     send right local[n/size] f;
6     local[n/size+1] <- recv right f;
7     local[0] <- recv left f;
8     isSkip f); (* Annotation *)
9 else if (rank = size-1) then (
10    let f = expand body [size-2, size-1, 0] in (* Annotation *)
11    local[lsize+1] <- recv right f;
12    local[0] <- recv left f;
13    send left local[1] f;
14    send right local[lsize] f;
15    isSkip f); (* Annotation *)
16 else (
17     ...
18 isSkip inbody; (* Annotation *)
19 (* Computation is performed here, removed for simplicity *)
```

Results: Annotation effort and verification time

Program	Why3 LOC	Why3 Anot	Ratio	VCC LOC	VCC Anot	Why3/VCC
Pi	33	6	18%	42	10	23%
Finite differences	86	29	33%	128	49	38%
Parallel dot	61	11	18%	99	30	30%

Program	Why3 Sub-Proofs	Why3 Time (s)	VCC Time (s)	Why3/VCC
Pi	27	1,6	2,4	66,7%
Finite differences	374	14,9	16,1	92,5%
Parallel dot	298	7,9	7,4	106,7%

Conclusions

- Our approach does not suffer from the state-explosion problem, typical of model checking
- Nor does it require any sort of runtime verification
- Results comparable with the closest tool, with less annotations and more verification options
- Many program annotations still required
- ... but can be simplified
- Protocols have the advantage of also serving as documentation

Future work

- Larger subset of MPI primitives (e.g. non-blocking, wildcard receive)
- Reduce annotation effort
- Adapt work for industry use
- Cover real-world applications

Thank you!

Questions?