PARALLEL PORTFOLIO VERIFIER

Tobias Kleinert

Developing a Verifier Based on Parallel Portfolio with CoVeriTeam

Mentor: Sudeep Kanav
Supervisor: Prof. Dr. Dirk Beyer

LMU Munich, Germany

Motivation

- Software Verification takes a lot of time
- Software Verification tools have different strengths
- Example from SV-COMP 22:
  - `loops/while_infinite_loop_3.c`

    CPACHECKER: Solved in 5 s of CPU time

    ESBMC: timeout
  - array-examples/sanfoundry_10_ground.c

    CPACHECKER: timeout

    ESBMC: Solved in less than 1 s of CPU time

Combine tools and use their strengths

Portfolio Verifier - Idea

- Take an arbitrary number of verifiers
- Give them the same program and specification to verify
- Run them in parallel
- Take the first result which satisfies a given condition
- Terminate the still running verifiers

Portfolio Verifier - Idea

Parallel Portfolio

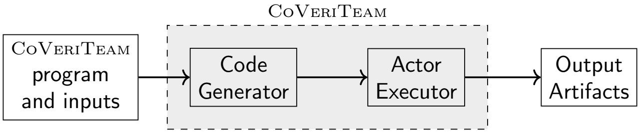
Input: Program p, Specification s
Output: Verdict

```
verifier_1 := Verifier("CPAchecker")
verifier_2 := Verifier("ESBMC")
success_condition := verdict ∈ {TRUE, FALSE}
parallel_portfolio := ParallelPortfolio(
        verifier_1,
        verifier_2,
        success_condition
    )
result := parallel_portfolio.verify(p,s)
return (result.verdict, result.witness)
```

CoVeriTeam

- Builds compositions of existing verification tools
- Main parts of CoVeriTeam
  - **Artifact:** Files and results
  - **Actor:** Uses artifacts as input and produces new ones
    - **Composition** of multiple actors
    - **Atomic actor:** External verification tools
    - **Utility actor:** Manipulates artifacts

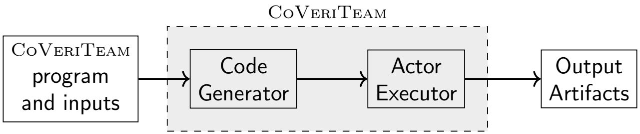
CoVeriTeam



Execution in CoVeriTeam

- Uses own lightweight language to describe compositions (in text format)
- Converts these descriptions to Python code
- Downloads (if necessary) and executes actors
- Returns the produced artifacts

Parallel Portfolio implementation



Execution in CoVeriTeam

- Extended the input language of CoVeriTeam
- Extended the code generation
- Implemented a new composition Parallel Portfolio in CoVeriTeam
  - Type check
  - Execution

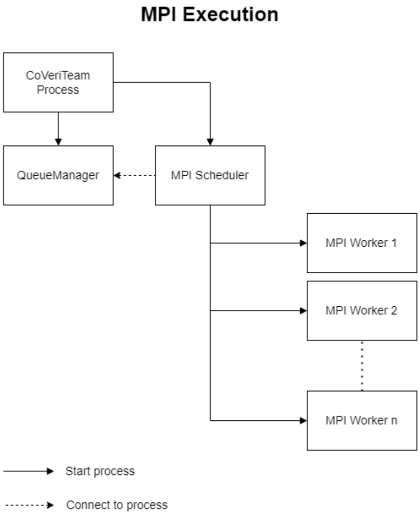
Execution

- Parallel execution $\Rightarrow$ multiple processes
- Each process executes one actor
- Sends result back to main process
- Main process evaluates `success_condition` with result
- If success condition is true stop the still running processes otherwise wait
- Spawning and communication of these processes with MPI

MPI

- Message-Passing-Interface (MPI)
- Used to exchange messages between processes
- Exchange of messages in groups of processes, so called `communicators`
- Different kinds of exchange of messages
  - One-to-one messages (e.g. `MPI_Send`, `MPI_Isend`)
  - Collective message operation (e.g. `MPI_Broadcast`)
- Available only in `C`/`C++` and FORTRAN (we used MPI4PY)

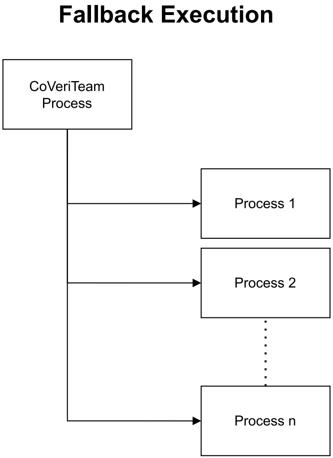
Use of MPI



- `MPI Scheduler`: Spawns worker and receives their results
- `QueueManager`: Synchronize CoVeriTeam process and `MPI Scheduler`
- `MPI Worker`: Executes actors

Fallback execution

- MPI execution needs MPI implementation and MPI4PY installed $\rightarrow$ fallback execution, if one not present
- Uses only PYTHON
- Similar, but less complicated setup
- Sharing one `Queue` for result sending



Evaluation

Experiments:

- Fallback vs MPI Execution
- Parallel Portfolio with Verifier + Validator combination
- Execution on a cluster

Tool selection:

1. CPAchecker
2. Esbmc
3. Symbiotic

Same selection as Beyer, Kanav, and Richter in "*Construction of Verifier Combinations Based on Off-the-Shelf Verifiers*"

Evaluation

- Evaluation with BenchExec
- Benchmark-set for the `unreach call` specification
  Contains 8883 tasks
- Comparison with CPAchecker
- Tools from SV-COMP 21

Resources:

- CPU: Intel Xeon E3-1230 v5 @ 3.40 GHz (apollon*) (except cluster)
- RAM: 15 GB (in most runs)
- CPU time: 15 min (in most runs)
- MPI Implementation: OpenMPI v4.0.3 (except cluster)

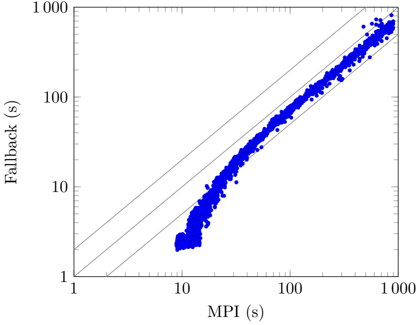
Evaluation - MPI vs. Fallback

	CPAchecker	Parallel Portfolio MPI	Parallel Portfolio fallback
Score	9040	9057	9460
Correct	5652	6129	6364
True	3516	3824	3992
False	2136	2305	2372
Wrong	8	35	35
True	0	21	21
False	8	14	14
Resource Consumption			
CPU time (h)	960	860	750
Wall-time (h)	670	250	330
Energy (KJ)	37 000	26 000	24 000

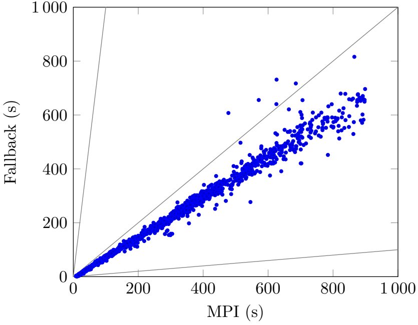
Results

- More solved tasks, but also more wrong results
  → Score of CPAchecker and Parallel Portfolio (MPI) about the same
- Fallback Parallel Portfolio performed better than MPI Parallel Portfolio

Evaluation - MPI vs. Fallback - CPU time



Log comparison

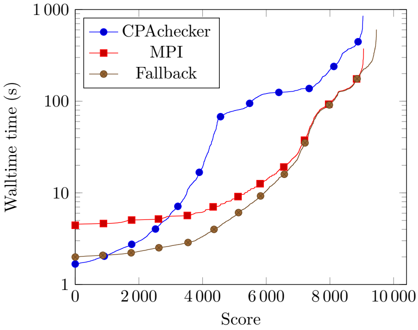


Linear comparison

Results

- Linear increase of CPU time difference due to busy waiting in `MPI Scheduler`

Evaluation - MPI vs. Fallback - Wall time



Results

- Lower walltime for fast tasks in `Fallback`
  $\Rightarrow$ Setup for python processes needs less time

Evaluation - Validating portfolio

Validating Parallel Portfolio

Input: Program p, Specification s

Output: Verdict

```
verifier_1 := Verifier("CPAchecker")
...
validator := Validator("CPAchecker-Validator")
verifier_1 := SEQUENCE(verifier_1, validator)
...
success_condition := (verdict == verdict_validator) ∈ {TRUE}
parallel_portfolio := ParallelPortfolio(
        verifier_1,
        verifier_2,
        verifier_3,
        success_condition
)
result := parallel_portfolio.verify(p,s)
return (result.verdict, result.witness)
```

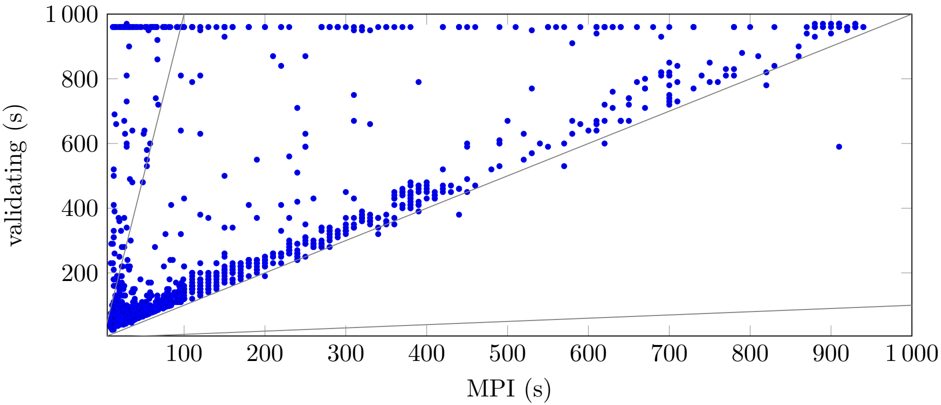
Evaluation - Validating portfolio

	Parallel Portfolio MPI	Parallel Portfolio validating
Score	9057	5100
Correct	6129	3612
True	3824	1552
False	2305	2060
Wrong	35	4
True	21	0
False	14	4

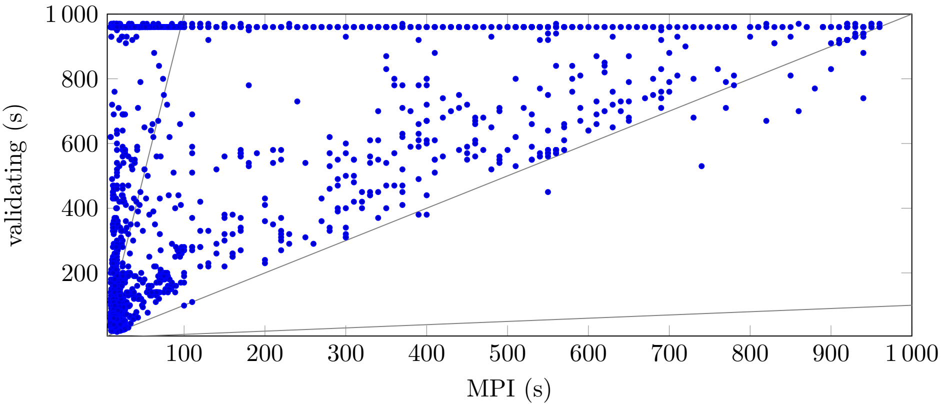
Results

- Minimized the wrong results
- Total number of results reduced drastically

Evaluation - Validating portfolio - Tasks False



Evaluation - Validating portfolio - Tasks True



Evaluation - Cluster

Special Test Setup:

- Composition of 12 Verifiers (all from SV-COMP 21 except VeriAbs)
- Executed on a cluster of 4 machines
- BenchExec as benchmark framework (no CPU time and energy measurement)

Resources:

- CPU: Intel Core i7-6700 @ 3.40 GHz (ws*)
- RAM: 30 GB
- CPU time: 30 min
- MPI Implementation: MPICH

Evaluation - Cluster

	Parallel Portfolio MPI	Parallel Portfolio cluster
Score	9057	9227
Correct	6129	6681
True	3824	4114
False	2305	2567
Wrong	35	66
True	21	32
False	14	34
Resource Consumption		
Wall-time (h)	250	240

Results

- A lot of solved tasks
- Also a lot of incorrect results
- Only slight increase of the score for the available resources (Used 4 machines instead of one)

Conclusion

- Implemented Parallel Portfolio composition in CoVeriTeam
- Fast (in terms of wall-time) and energy efficient execution
- Relatively high amount of wrong results (due to the nature of the Parallel Portfolio)
- Fast fallback execution
- Cluster capability

Conclusion

Special achievements:

- The Parallel Portfolio participated in the SV-COMP 22 (not as competitor)
  - Second place in `ReachSafety`
  - Even first place in `NoOverflows`
- The Parallel Portfolio was used in the paper "*Construction of Verifier Combinations Based on Off-the-Shelf Verifiers*" from Beyer, Kanav, and Richter.

Future work

- Investigate the validating Parallel Portfolio
  - Run on a cluster
  - Use of different validators
- Parallel Portfolio of testers

Thank you!

MPI Environment

- MPI programs need to be started with `mpiexec`
- Spawning new process with `mpiexec`
- Process executes Python interpreter
  Command: "`mpiexec -n 1 python mpi-scheduler.py`"

Process spawning

Static

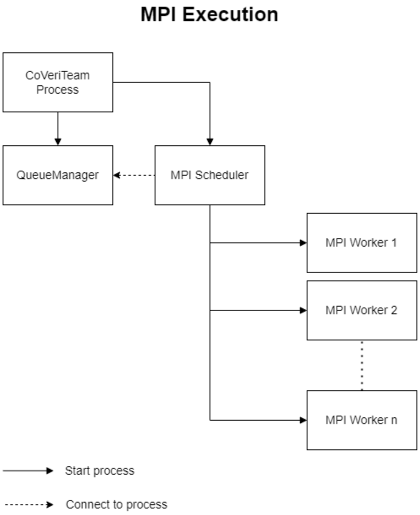
- Need to calculate the needed processes
- Grouping processes difficult
- Easy to use with SLURM

Dynamic

- Processes are spawned when needed
- Processes are grouped by default
- `MPI_Comm_Spawn` not supported with OpenMPI in combination with SLURM

Busy waiting

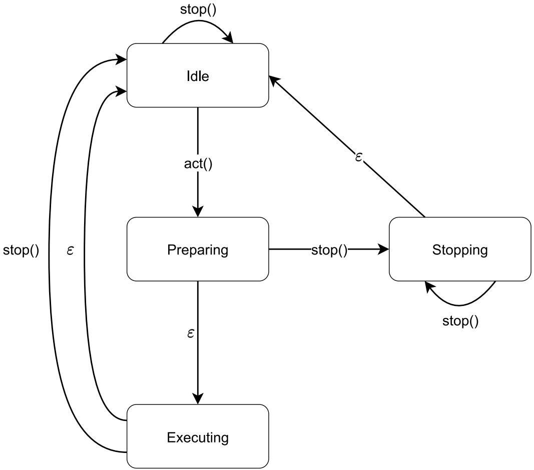
- MPI uses busy waiting as waiting strategy
  $\Rightarrow$ A lot of CPU time is wasted
- Solution: Manual pause of the check for new messages in `MPI Worker`
- `MPI Scheduler` still uses busy waiting



Shutdown of actors

- New Feature in CoVeriTeam: Shutdown of actors
- Shutdown procedure for each composition
  - Parallel → Stop both actors
  - Sequence → Stop first, then second actor
  - ITE → Invalidate condition, then stop first actor
  - Repeat → Invalidate repeat condition, then stop actor
- Shutdown for external tools (atomic actors)

Stopping of atomic actors



- First, only boolean flag
- Bug occurred with some tools
- $\Rightarrow$ more complex shutdown procedure

Exiting the MPI environment

- Result are sent back with the `QueueManager`
- `MPI Scheduler` shuts down every `MPI Worker`
- `MPI Scheduler` catches signals to terminate the workers