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

1: verifier_1 := Verifier("CPAchecker")
2: verifier_2 := Verifier("ESBMC")
3: success_condition := verdict ∈ \{TRUE, FALSE\}
4: parallel_portfolio := ParallelPortfolio(
    verifier_1,
    verifier_2,
    success_condition
)
5: result := parallel_portfolio.verify(p,s)
6: **return** (result.verdict, result.witness)
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
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 ⇒ 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
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 → 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 unreachable 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

<table>
<thead>
<tr>
<th></th>
<th>CPAchecker</th>
<th>Parallel Portfolio</th>
<th>Parallel Portfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td>fallback</td>
</tr>
<tr>
<td>Score</td>
<td>9040</td>
<td>9057</td>
<td>9460</td>
</tr>
<tr>
<td>Correct</td>
<td>5652</td>
<td>6129</td>
<td>6364</td>
</tr>
<tr>
<td>True</td>
<td>3516</td>
<td>3824</td>
<td>3992</td>
</tr>
<tr>
<td>False</td>
<td>2136</td>
<td>2305</td>
<td>2372</td>
</tr>
<tr>
<td>Wrong</td>
<td>8</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>True</td>
<td>0</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>False</td>
<td>8</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

### Resource Consumption

<table>
<thead>
<tr>
<th></th>
<th>CPAchecker</th>
<th>Parallel Portfolio</th>
<th>Parallel Portfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU time (h)</td>
<td>960</td>
<td>860</td>
<td>750</td>
</tr>
<tr>
<td>Wall-time (h)</td>
<td>670</td>
<td>250</td>
<td>330</td>
</tr>
<tr>
<td>Energy (KJ)</td>
<td>37000</td>
<td>26000</td>
<td>24000</td>
</tr>
</tbody>
</table>

### 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](image1)

![Linear comparison](image2)

**Results**

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

- Lower walltime for fast tasks in Fallback
  ⇒ Setup for python processes needs less time
Validating Parallel Portfolio

**Input:** Program p, Specification s  
**Output:** Verdict

1: verifier_1 := Verifier("CPAchecker")

... 

2: validator := Validator("CPAchecker-Validator")

3: verifier_1 := SEQUENCE(verifier_1, validator)

... 

4: success_condition := (verdict == verdict_validator) ∈ \{TRUE\}

5: parallel_portfolio := ParallelPortfolio(
   verifier_1,
   verifier_2,
   verifier_3,
   success_condition
)

6: result := parallel_portfolio.verify(p,s)

7: return (result.verdict, result.witness)
## Evaluation - Validating portfolio

<table>
<thead>
<tr>
<th>Parallel Portfolio</th>
<th>MPI</th>
<th>Parallel Portfolio validating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>9057</td>
<td>5100</td>
</tr>
<tr>
<td>Correct</td>
<td>6129</td>
<td>3612</td>
</tr>
<tr>
<td>True</td>
<td>3824</td>
<td>1552</td>
</tr>
<tr>
<td>False</td>
<td>2305</td>
<td>2060</td>
</tr>
<tr>
<td>Wrong</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>True</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>False</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>

### Results

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

Special Test Setup:
- Composition of 12 Verifiers (all from SV-COMP 21 except \textit{VERIA}BS)
- Executed on a cluster of 4 machines
- \texttt{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

<table>
<thead>
<tr>
<th></th>
<th>Parallel Portfolio</th>
<th>Parallel Portfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPI</td>
<td>cluster</td>
</tr>
<tr>
<td><strong>Score</strong></td>
<td>9057</td>
<td>9227</td>
</tr>
<tr>
<td><strong>Correct</strong></td>
<td>6129</td>
<td>6681</td>
</tr>
<tr>
<td><strong>True</strong></td>
<td>3824</td>
<td>4114</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td>2305</td>
<td>2567</td>
</tr>
<tr>
<td><strong>Wrong</strong></td>
<td>35</td>
<td>66</td>
</tr>
<tr>
<td><strong>True</strong></td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td><strong>False</strong></td>
<td>14</td>
<td>34</td>
</tr>
<tr>
<td><strong>Resource Consumption</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Wall-time (h)</strong></td>
<td>250</td>
<td>240</td>
</tr>
</tbody>
</table>

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**
  - 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 \textsc{CoVeriTeam}: Shutdown of actors
- Shutdown procedure for each composition
  - Parallel $\rightarrow$ Stop both actors
  - Sequence $\rightarrow$ Stop first, then second actor
  - ITE $\rightarrow$ Invalidate condition, then stop first actor
  - Repeat $\rightarrow$ 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