[Artifact] RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning

This package contains the artifact for of *RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning*, DOI 10.1145/3582016.3582034

This artifact contains the source code for RepCut, as well as other open source projects that are required to reproduce the results in the paper. We include Verilator 4.226 as a baseline. In addition, this artifact also contains scripts and a Makefile to compile and run the generated simulators, as well as to reproduce every figure and table from experimental data.

This artifact has been validated on the platform from the paper (detailed in Table 2).

To verify or reproduce the results in the paper, this package contains 3 sets of scripts/instructions with increasing coverage:

1. Kick the Tires (section 4, 5) This setting is used to quickly prepare the environment and compile a few simulators. In this setting, you can compile simulators from RepCut or Verilator, and run them manually.
2. Quick Compilation (section 4, 6) This setting compiles design `boom21-2small` (Single core Small Boom) using RepCut with 1, 2, 4, 6, and 8 threads. This option is for a quick verification of linear/superlinear simulation scalability.
3. Full Compilation (section 4, 7) If you wish to build everything from source code, this setting is what you would like to try. It requires extra time and memory, but it will reproduce every figure and table in the paper from scratch.

More details can be find in later sections.

0. Resource Requirements

1. Multicore x86 machine to test parallel simulation performance. For detailed profiling, both Verilator and RepCut generated simulators rely on x86's RDTSC instruction to collect CPU ticks.

2. OS: Linux (Ubuntu recommended and tested)

3. Sufficient memory to build simulators

   - For a *Kick the Tires Compilation*, ~`10 GB` of memory is required.
   - For a *Quick Compilation*, ~`40 GB` of memory is required for a parallel build (`make -j11`).
   - For a *Full Compilation*, ~`500 GB` of memory is required for a parallel build (`make -j30`).

4. Sufficient disk space. We recommend using SSD for better disk performance

   - For a *Kick the Tires Compilation*, ~`60 GB` is required.
   - For a *Quick Compilation*, ~`60 GB` is required.
   - For a *Full Compilation*, ~`500 GB` is required.

5. Internet connection

6. Estimated build time:

   - For all configuration, ~ `40 minutes` is required to prepare the build environment. (section 4)
   - For a *Kick the Tires Compilation*, ~ `3 minutes` if compiling a single, small design (RepCut, rocket21-1c, 2 threads, `make -j2 compile_essent_rocket21-1c_2`).
   - For a *Quick Compilation*, ~ `30 minutes`
   - For a *Full Compilation*, ~ `42 hours` without Verilator PGO, additional `10 hours` to compile Verilator PGO simulators

7. Estimated simulation time:

   - For a *Kick the Tires Compilation*, < `1 minutes` if testing a single, small design (RepCut, rocket21-1c, 2 threads).
   - For a *Quick Compilation*, ~ `5 minutes`
   - For a *Full Compilation*, ~ `30 hours` without Verilator PGO, additional `8 hours` to run Verilator PGO simulators

1. Software Dependencies

- To compile and run Verilator (our baseline for comparison) requires `git autoconf flex bison help2man perl python3 make unzip` (Verilator dependencies can be found here)

- RepCut is built on top of ESSENT, which requires a Java environment and `sbt`. OpenJDK 11 is tested and recommended.

- `device-tree-compiler` is needed by both designs (rocket chip and Boom).

- `cmake` to build KaHyPar.

- Build tools: Please make sure `make`, `git` and `python3` are installed

- Data processing: `python3` is needed to process data and produce figures. Python package `matplotlib` and `numpy` are required. In addition, `rsync` is used to copy log files.

- `clang++`: C++ compiler. clang 10, 12 and 14 tested. We recommend clang 14 for compilation speed and performance.

- `numactl` (for pinning processes), `time` (for measuring execution time), `perf` (package `linux-tools-generic` in Ubuntu)

To install all dependencies (except `sbt`, please follow guide here) on Ubuntu:

```
sudo apt install build-essential clang git flex bison help2man perl device-tree-compiler
sudo apt install python3-matplotlib python3-numpy rsync numactl time cmake unzip
sudo apt install openjdk-11-jdk linux-tools-generic autoconf
```

2. Open Source Projects

This artifact uses several open-source projects. Specifically, the following 2 projects are used to generate benchmark designs:

- `rocket-chip`: Rocket Chip Generator, commit `4276f17f989b99e18e0376494587fe00cd09079f`
- `boom-standalone`: BOOM is an open-source OoO RISC-V core used by this paper's evaluation. BOOM itself is not a self-running project. We run BOOM using `rocket-chip`'s IO and debug port (See this repo, commit `4276f17f989b99e18e0376494587fe00cd09079f`)

A hyper graph partitioner is used to produce high-quality hyper graph partitions:

- `KaHyPar`: KaHyPar, commit `76249c04e276a92857c8bdfae8ebfe013079b166`

Other open-source projects to run the simulator:

- `firrtl`: Convert `chisel` generated FIRRTL file to Verilog. FIRRTL, commit `a6851b8ec4044eef4af759a21887fdae6226e1cd`
- `riscv-isa-sim`: We use `fesvr` in riscv-isa-sim to create simulators. commit `ddcfa6cc3d80818140a459e590296c3079c5a3ec`, `68b3eb9bf1c04c19a66631f717163dd9ba2c923c`
- `firrtl-sig`: C++ library that provides UInt and SInt from FIRRTL spec. firrtl-sig, commit `4504848ad436c172ca997142b2744926421c4f66`

Thanks for all of the contributions from the open-source community!

3. File Structure

This AE package contains the following directories:

We use environment variable `PKGROOT` to denote the root directory of this artifact.

- `$(PKGROOT)/designs/`: compile all reference designs.
- `$(PKGROOT)/weighted/`: compile and evaluate `RepCut` and `Verilator`.
- `$(PKGROOT)/unweighted/`: compile and evaluate `RepCut` with no weights (every FIRRTL node has a weight of 1).
- `$(PKGROOT)/data_analysis/`: contains data processing scripts.

RepCut's source code is under `weighted/essent-verilator-testbed/essent/`

4. Compilation Guide: Prepare

This section describes preparation required to setup this package. Please follow this section before moving ahead.

0. Before Starting

Before compilation, please check following items:

Set compiler

Please set environment variables `CXX` `LINK` and `AR` to your designated compiler.

```
# Example: Set to clang 14

export CXX=clang++-14
export LINK=clang++-14
export AR=llvm-ar-14
```

We recommend using `clang` since we have observed **`gcc` throwing `Internal Compiler Error`** on some of the largest designs. Due to the large volume of code and impact of instruction delivery on simulation performance, `RepCut` generated simulators are sensitive to the compiler used. It is expected that different compilers and even different versions of same compiler may lead to noticeable performance differences.

Configure Number of Parallel Threads

This artifact is configured to compile and run for `1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 32, 48` threads. Reducing the number of thread counts evaluated will reduce the simulation burden. Like many parallel works, the smaller thread counts consume more time, so for overall speed, they should be the first removed. **Note: changing the thread counts may break some plot scripts**

You may change configurations in following files to adjust:

```
# Change $(NTHREAD) and $(NTHREAD_PARALLEL)
# Note: Single thread is required to calculate speedup.
# Please make sure 1 exists in $(NTHREAD)

$(PKGROOT)/weighted/essent-verilator-testbed/Makefile


# Change $(NTHREAD)
# Note: Single thread not needed

$(PKGROOT)/unweighted/essent-verilator-testbed/Makefile-essent-no-weight


# Change thread number in all run scripts:

run_*.sh


# Change plot scripts

$(PKGROOT)/data_analysis/essent-verilator-testbed/data_processing/bench.py
```

Update `gen_numactl_param.py` according to your machine configuration

This file located at `$(PKGROOT)/weighted/essent-verilator-testbed/gen_numactl_param.py` and `$(PKGROOT)/unweighted/essent-verilator-testbed/gen_numactl_param.py` (2 identical copies). It generates (print out) arguments for `numactl` to allocate emulator threads and memory. Please change this script and make it best fits your machine.

This file takes 2 arguments. The first is `<num of parallel threads>` and second is `cross/local`.

Examples:

```
# Run 4 thread simulator within a local NUMA node
# Result: Allocate memory on NUMA node 1 and run simulator using core 24, 25, 26, 27

$> python3 gen_numactl_param.py 4 local
-m 1 -C 24,25,26,27
```

```
# Run 4 thread simulator cross NUMA nodes
# Result: Allocate memory on NUMA node 1 and run simulator using core 0, 24, 1, 25

$> python3 gen_numactl_param.py 4 cross
-m 1 -C 0,24,1,25
```

Currently this file is written for a server with 2x Intel Xeon Platinum 8260 processor (24 cores per socket/NUMA node) (Table 2)

Check Perf Events

This artifact relies on the following perf events, please check if your test platform supports all of them (by `perf list`)

```
cycles
instructions

L1-icache-load-misses
L1-dcache-load-misses
L1-dcache-loads
L1-dcache-stores
l2_rqsts.code_rd_hit
l2_rqsts.code_rd_miss
l2_rqsts.miss
l2_rqsts.pf_hit
l2_rqsts.pf_miss
l2_rqsts.all_demand_data_rd
l2_rqsts.all_demand_miss

LLC-load-misses
LLC-loads
LLC-store-misses
LLC-stores

branches
branch-misses

topdown-fetch-bubbles

icache_16b.ifdata_stall
icache_64b.iftag_stall
```

If not, you are unable to reproduce Table 3. Please remove `line 135, 138` in `Makefile` to avoid encounter errors.

1. Compile Hardware Designs

Compile designs from source code

```
# Generate FIRRTL file and copy to workspace
cd $(PKGROOT)/
make design_firrtl
```

Suggestion of parallelism `<N>`: Single thread compilation.

Max. memory requirement: ~`10 GB`

Typical time: 20 minutes

2. Prepare environment

This step creates build directory, compiles necessary dependencies and also compiles FIRRTL into Verilog for Verilator. An internet connection is required to download `boost` when compiling `KaHyPar`.

```
cd $(PKGROOT)/
make -j<N> prepare
```

Suggestion of parallelism `<N>`: 20 or more

Max. memory requirement: <`100 GB` using `make -j24`

Typical time: 13 mins using `make -j24`

5. Kick the Tires Test

1. Compile a single design

This artifact contains following designs and supports following number of threads:

Name	Description
rocket21-1c	Single core Rocket Chip
rocket21-2c	Dual core Rocket Chip
rocket21-4c	Quad core Rocket Chip
boom21-small	Single core Small Boom
boom21-2small	Dual core Small Boom
boom21-4small	Quad core Small Boom
boom21-large	Single core Large Boom
boom21-2large	Dual core Large Boom

boom21-4large	Quad core Large Boom
boom21-mega	Single core Mega Boom
boom21-2mega	Dual core Mega Boom
boom21-4mega	Quad core Mega Boom

```
NTHREADS = 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 32, 48
```

A legal compile target name has following format:

```
# <simulatorName>: essent(RepCut), verilator(baseline)
#                   verilator_prof, verilator_pgo1, verilator_pgo2
#         Note: verilator_prof, verilator_pgo1 and verilator_pgo2 generates or
requires
#              data that need special care. We recommend build them in a full build
#              (see later)
# <designName>: Shown in previous table
# <numThreads> pick one from NTHREADS

compile_<simulatorName>_<designName>_<numThreads>
```

For example:

```
cd $(PKGROOT)/weighted/essent-verilator-testbed

# build RepCut simulator, rocket21-1c, 2 threads:
make -j2 compile_essent_rocket21-1c_2

# build Verilator simulator, rocket21-1c, 2 threads:
make -j<N> compile_verilator_rocket21-1c_2
```

Generated simulators can be found at `$(PKGROOT)/weighted/essent-verilator-testbed/emulator` and is named as:

```
emulator_<simulatorName>_<designName>_<numThreads>t
```

Suggestion of parallelism `<N>`: For RepCut, we recommend `N=2`. RepCut doesn't support parallel compilation, and compiles both a regular simulator and simulator with thread profiling. For Verilator, we recommend a larger number depending on your test platform.

Max. memory requirement: < `100 GB`

Typical time: Depends on the design. We do not recommend testing large design like LargeBoom and MegaBoom for a quick test. As mentioned earlier, RepCut doesn't support parallel compilation, so large designs may lead to long compilation time. We recommend `rocket21-1c_2` for time-saving purpose.

2. Run a simulator

To run a simulator:

```
cd $(PKGROOT)/weighted/essent-verilator-testbed/

/usr/bin/time numactl <numactlParameters> ./emulator/<simulatorBinaryName> -c ./riscv-benchmarks-bin/dhrystone.riscv

# Example: run RepCut, rocket21-1c, 2t on CPU 1,2 (2 threads), socket 0:

$> /usr/bin/time numactl -m 0 -C 1,2 ./emulator/emulator_essent_rocket21-1c_2t -c ./riscv-benchmarks-bin/dhrystone.riscv
Microseconds for one run through Dhrystone: 1843
Dhrystones per Second:  542
mcycle = 922049
minstret = 571969
Completed after 1439263 cycles
25.07user 0.47system 0:13.12elapsed 194%CPU (0avgtext+0avgdata 791884maxresident)k
0inputs+0outputs (0major+196962minor)pagefaults 0swaps
```

6. Quick Compilation

This setting compiles design `boom21-2small` (Dual core Small Boom) using RepCut with 1, 2, 4, 6, and 8 threads. This option is for a quick verify of linear/superlinear simulation scalability.

1. Compile Simulators

```
cd $(PKGROOT)/
make -j<N> emulator_essent_quick
```

Suggestion of parallelism `<N>`: 11 is sufficient

Max. memory requirement: ~`40GB` when `make -j11`

Typical time: ~ `30 minutes` when `make -j11`.

2. Run simulators

```
cd $(PKGROOT)/
make run_quick
```

Suggestion of parallelism `<N>`: Single thread

Max. memory requirement: < `1GB`

Typical time: ~ `3 minutes`

3. Print result

Show speedup and simulation performance for the quick test.

```
cd $(PKGROOT)/
make result_quick
```

Example output:

```
$> make result_quick
cd /lscratch/hwang/ae/ae-pkg/weighted/essent-verilator-testbed && python3
./print_quick_result.py
boom21-2small: 1 threads: 160.01s elapsed, 1.00x speedup, simulation speed at 10.14kHz
boom21-2small: 2 threads: 59.98s elapsed, 2.67x speedup, simulation speed at 27.04kHz
boom21-2small: 4 threads: 25.93s elapsed, 6.17x speedup, simulation speed at 62.55kHz
boom21-2small: 6 threads: 17.92s elapsed, 8.93x speedup, simulation speed at 90.50kHz
boom21-2small: 8 threads: 16.06s elapsed, 9.96x speedup, simulation speed at 100.99kHz
```

7. Full Compilation

If you wish to build everything from scratch, this is the section you should check.

1. Compile simulators

Compile RepCut, RepCut (no weight) and Verilator simulators:

Compile everything, include RepCut simulator (weighted), RepCut simulator (unweighted), RepCut simulator with thread profiling, Verilator simulator, Verilator simulator with thread profiling.

```
cd $(PKGROOT)/
make -j<N> emulator_no_pgo
```

Suggestion of parallelism `<N>`: 20 or more. **Don't use all available cores! Leave some cores for Java GC.**

Max. memory requirement: ~`500GB` when `make -j30` (load average 60 ~ 70), ~`400GB` when `make -j24`,

Typical time: ~ 42 hours when `make -j30`.

Compile Verilator with PGO (Profile Guided Optimization)

Verilator with PGO (Profile Guided Optimization) uses profile data collected at run time to guide thread scheduling. To achieve this, Verilator with PGO requires the following steps:

1. Compile the simulator with thread profile code (denote as `pgo1`)
2. Run generated binary and acquire thread profile data (Cost 8~9 hours in our test)
3. Compile again using generated data (`pgo2`)
4. Acquire binary (and need run again to observe its performance, another 8~9 hours)

Compiling Verilator with PGO and obtaining performance data needs ~`20`hours. If you wish to check Verilator PGO data, please compile as following:

```
cd $(PKGROOT)/
make -j<N> emulator_verilator_pgo2
```

Suggestion of parallelism `<N>`: 40 or more.

Max. memory requirement: ~ `200GB` when `make -j40`

Typical time: `1 hour` for compile `pgo1` and `pgo2`, ~`8.5 hours`for collecting profile information. ~`10 hours` in total.

2. Run simulators

Before starting execution, please update `gen_numactl_param.py` according to your machine NUMA topology

Run RepCut, RepCut (no weight) and Verilator simulators:

Run RepCut simulators and Verilator simulators (No PGO):

```
cd $(PKGROOT)/
make run_no_pgo
```

Following item will be executed:

Task	Duration (Est.)
RepCut unweighted	3 hours
RepCut weighted	3 hours
RepCut Cross Socket perf	10 mins
RepCut Cross Socket	1 hour
RepCut perf	30 mins
RepCut thread profile	5.5 hours
Verilator	8.5 hours
Verilator thread profile	8 hours
Total	~30 hours

Suggestion of parallelism `<N>`: No parallelism.

Max. memory requirement: < `1GB`

Run Verilator PGO simulators

```
cd $(PKGROOT)/
make run_pgo2
```

Following item will be executed:

Task	Duration (Est.)

Verilator PGO2	8 hours

Suggestion of parallelism `<N>`: No parallelism.

Max. memory requirement: < `1GB`

3. Obtain Figures and Tables

Generate Figures

Generate figures without Verilator PGO data:

```
cd $(PKGROOT)/
make figures_no_pgo
```

Alternatively, if you have run Verilator PGO, generate figures with Verilator PGO:

```
cd $(PKGROOT)/
make figures_with_pgo
```

Figures are under `$(PKGROOT)/data_analysis/essent-verilator-testbed/`

Figure	File Name
Figure 2a	verilator_gantt_heatmap_combined.pdf
Figure 2b	essent_gantt_heatmap_combined.pdf
Figure 6	replication_cost_4by1.pdf
Figure 7	speedup_4by3.pdf
Figure 8	peak_speedup_node_count.pdf
Figure 9	sim_speed.pdf
Figure 10	cross_socket_speedup_2.pdf
Figure 11	exec_profile.pdf
Figure 12	ib_speedup.pdf
Figure 13	imbalance_4by3.pdf

RepCut profile is processed by `python3` in parallel (using half of available cores, 16 core max)

Max. memory requirement: ~`100GB`

Typical time: ~`18mins`

Generate Tables

The following command generates the LaTeX code of tables in this paper:

Table 1: Evaluated Design

```
cd $(PKGROOT)/data_analysis/essent-verilator-testbed/
python3 ./data_processing/table_design.py
```

Table 3: Performance Counters

```
cd $(PKGROOT)/data_analysis/essent-verilator-testbed/
python3 ./data_processing/table_perf.py
```