Usage Guide: SimpleScalar Cache Experiments

This document describes how to build, configure, and run the cache experiments used in this project, and how to reproduce the aggregated results from scratch.

The workflow is designed so that all experiment logic lives in scripts and configuration files.

1. Repository Structure

```
.
├── config/
│   ├── ss_cache.sh            # Cache + mechanism configuration
│   └── ss_benchmarks.txt      # Benchmark definitions
├── docs/
├── scripts/
│   ├── run_ss_cache.sh        # Run one benchmark + config
│   ├── run_ss_sweep.sh
│   ├── run_all_ss_experiments.sh
│   └── summarize_ss_results.py
├── results/
│   └── ss/                    # All SimpleScalar outputs
├── simplesim-3.0/              # SimpleScalar source + sim-cache
└── traces/                    # Benchmark binaries
```

Note: there may be more files within these directories, but this is the bare minimum needed for running the experiments

2. Prerequisites and Assumptions

Software

- Linux environment
- `bash`, `python3`
- Python packages
  - `pandas`
  - `numpy`
  - `matplotlib` A working build of SimpleScalar with the modified sim-cache (located in the `simplesim-3.0/` directory)

Assumptions

- SimpleScalar binaries are already compiled as PISA
- `sim-cache` reflects the modified version described in `simulator_modifications.md`
- Benchmarks under `traces/` are executable

3. Benchmark Configuration

Benchmarks are defined in `config/ss_benchmarks.txt`

Format (one per line): `<benchmark_name> <relative_path_to_binary> [args...]`

Example:

```
test-fmath traces/test-fmath
test-llong traces/test-llong
test-lswlr traces/test-lswlr
test-math  traces/test-math
```

Comments in the `config/ss_benchmarks.txt` file are ignored.

4. Cache and Mechanism Configuration

All cache geometry and mechanism selection is handled in: `config/ss_cache.sh`

Cache Geometry

Defaults (can be overridden by scripts):

```
export SS_IL1_CONFIG="il1:32768:64:1:l"
export SS_DL1_CONFIG="dl1:16:32:1:l"
export SS_UL2_CONFIG="ul2:8192:64:4:l"
```

Experimental Modes

Valid values for `SS_MODE`:

- `baseline`
- `victim`
- `miss`
- `stream`
- `victim_stream`
- `miss_stream`

Each mode automatically enables/disables the correct structures.

Mechanism Parameters

- Victim Cache: `SS_VC_ENTRIES`
- Miss Cache: `SS_MC_ENTRIES`
- Stream Buffer:
  - `SS_SB_COUNT`
  - `SS_SB_DEPTH`
  - `SS_SB_DEGREE`

Parameters are exported by the sweep script. Manual editing is not required for standard runs.

5. Running a Single Experiment (Manual)

To run a single benchmark under the current configuration: `scripts/run_ss_cache.sh <benchmark_name> <program_path> [args...]`

Example:

```
SS_MODE=stream \
SS_SB_DEPTH=8 \
scripts/run_ss_cache.sh test-fmath traces/test-fmath
```

Outputs:

- Text States:
  - `results/ss/<run_tag>/<benchmark>_<mode>.txt`
- CSV stats:
  - `results/ss/<run_tag>/ss_<mode>.csv`

6. Running All Experiments (Recommended)

To run the full experiment matrix:

```
bash scripts/run_all_ss_experiments.sh
```

To start from a clean state:

```
bash scripts/run_all_ss_experiments.sh --clean
```

This script:

- Sweeps DL1 size
- Sweeps UL2 size
- Runs all cache mechanisms
- Sweeps mechanism-specific knobs
- Organizes results by a structured RUN_TAG

Note: This script may launch hundreds of simulations. Runtime depends on benchmark count and machine speed.

7. Results Aggregation

After all runs complete, merge results into a single CSV:

```
python3 scripts/summarize_ss_results.py
```

This generates:

```
results/ss/ss_summary_all.csv
```

Each row corresponds to:

- One benchmark
- One cache geometry
- One mechanism mode
- One parameter configuration

8. Normalization

If normalization is wanted:

```
python3 scripts/normalize_data.py
```

Output:

```
results/ss/analysis/ss_normalized.csv
```