SPBench

Release 0.4-Alpha

Adriano M. Garcia

Oct 18, 2022

CONTENTS

Note: This project is under active development.

SPBench (Stream Processing Benchmark) is a framework for benchmarking C++ stream processing applications. The main goal of SPBench is to enable users to easily create custom benchmarks from real-world stream processing applications and evaluate multiple parallel programming interfaces (PPIs).

The SPBench is intended for three main audiences:

- Users who want to run performance tests with the SPBench benchmarks. It implements different real-world stream processing applications using different parallel programming interfaces (PPIs) and parallelism patterns, and makes available the main metrics used in this domain. The framework also enables users to create custom benchmarks with new PPIs.
- Researchers and developers who want to test and evaluate new technologies and solutions. The SPBench benchmarks are also highly parameterizable and its API allows for easy and fast code reuse across all applications.
- Students and teachers who want to learn/teach stream parallelism. The benchmarks implemented with the SPBench API abstract the low-level code and expose to users only the stream core of each application (in a few lines of code). Therefore, it allows users to easily identify each individual operator and data dependencies, to understand what the stream flow looks like. Then users can build parallelism on top of this highly simplified code.

CHAPTER

ONE

SPBENCH BENCHMARK SUITE

The SPBench suite comprises the following stream processing applications and will be more in the future:

- Ferret (PARSEC)
- Lane Detection
- Bzip2
- Face Recognizer (Person Recognition)

See more details in applications.

For each of them there are parallel benchmarks implemented using the following parallel programming interfaces and will be more in the future:

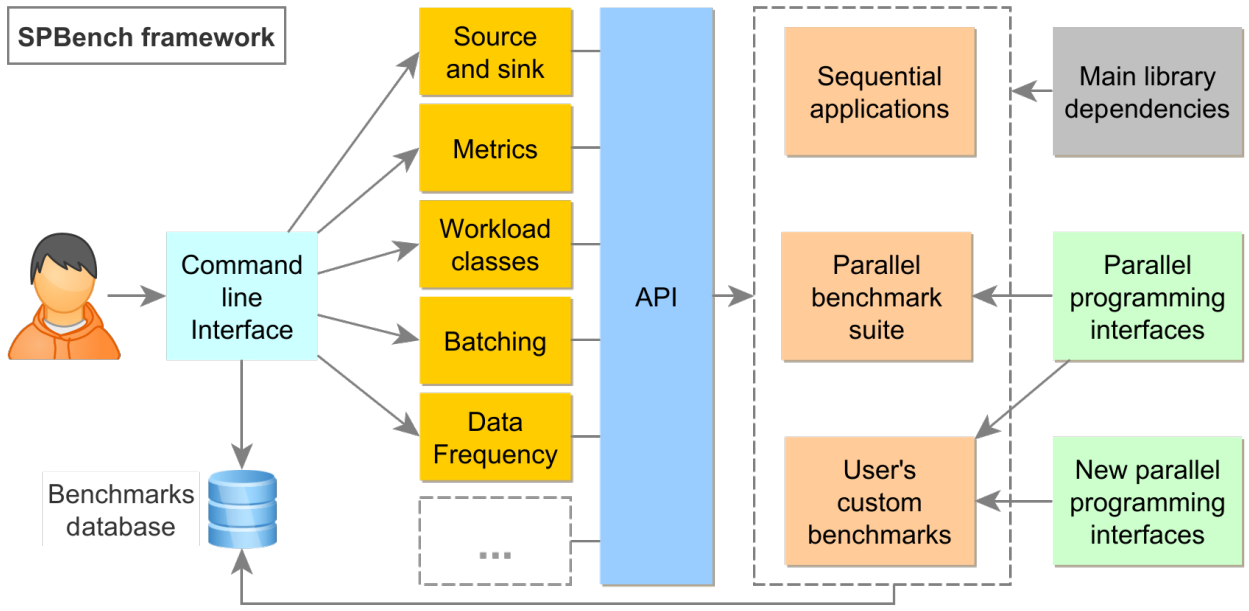
- Intel TBB
- FastFlow
- SPar
- GrPPI (with OpenMP, FastFlow, TBB, and C++11 threads backends)

CHAPTER

TWO

SPBENCH FRAMEWORK

One of the aspects that most differentiates SPBench from related solutions is how users interact with it. The kernel of SPBench is its API that allows users to access simplified versions of the applications included in our suite. Based on the sequential versions, users can implement parallelism, create new custom benchmarks with different parallelism strategies, or even explore new parallel PPIs. Users can also configure and modify parameters, such as build dependencies and metrics, through a command-line interface (CLI). This CLI accesses a database containing all benchmarks from the SPBench suite. Other secondary parameters can be tuned via the CLI with simple commands. This allows users to entirely focus on writing and tuning the parallelism rather than spending time with the non-relevant low-level aspects of each application. The figure below illustrates the SPBench framework.



CHAPTER

THREE

CONTENTS

3.1 Download Links

- GitHub: https://github.com/GMAP/SPBench
- SPBench Website: https://gmap.pucrs.br/gmap/en/software/

3.2 Basic Concepts

Table of Contents

- **Benchmark**: Here, it means any application developed using the SPBench API. Benchmarks can be the sequential implementations, the parallel versions pre-available within SPBench, or any new custom implementations latter added by users.
- **PPI**: It stands for Parallel Programming Interface, which means any library, framework, language, etc. that leverages parallelism.
- **Frequency**: Here, it means the maximum number of items that are available per second for the source operators.
- **Latency**: In SPBench, it means the time it takes for an item to go through the entire pipeline. An item is given a timestamp when it is generated in the source, and when that item has finished processing in the sink, that timestamp is used to compute the elapsed time.
- **Throughput**: It means items processed per time unit.

3.3 Getting Started

Table of Contents

3.3.1 SPBench Dependencies

The folowing packages are required to install and use SPBench:

- build-essential
- pkg-config
- cmake
- python (tested on python3 and python2.7)

You can run the command below to install theses packages or ask a system admin.

```
sudo apt-get install -y build-essential pkg-config cmake python3
```

3.3.2 Command-Line Interface

The SPBench framework works with a command-line interface (CLI).

The CLI offers some commands. You can check them running:

```
./spbench --help
```

For each command there is also a "help" guide. You can access it using:

```
./spbench [command] -h
```

Check *Command-Line Interface* for more details.

3.3.3 Installing SPBench

The purpose of the SPBench is to provide self-contained benchmarks, i.e., it provides the specific installation dependencies for each supported application. These dependencies can be either libraries like OpenCV for the image processing applications or parallel programming interfaces like Intel TBB for the parallel benchmarks.

Therefore, SPBench has specific commands to help download and install these dependencies. See installation section for more details or run the command below:

```
./spbench install -help
```

To run the benchmarks it is not enough to install the dependencies, you also need to download and install the workloads. SPBench also offers commands to do this. See section *SPBench workloads* for more details, or run the command below.

```
./spbench download-inputs -help
```

If there were no errors during the installation of the dependencies or workloads, then you are ready to compile and run the SPBench benchmarks. See section how_to_run for more details.

3.3.4 How to use

In this framework you can simply run the benchmarks that are already provided or implement your own specific benchmark.

To run the benchmarks you first need to have installed the application dependencies and also the workloads. See pages *How to install* and *SPBench workloads* for more details.

After installing the workloads and dependencies (make sure you set the environment variables `source setup_vars.sh`), you can run the 'list' command to see all the pre-available benchmarks in SPBench.

```
./spbench list
```

Then you can select one of the benchmarks from the list and use the command below to compile it.

```
./spbench compile -bench <selected_bench_name_from_list>
```

If the compilation was successful, just run it via the 'exec' command.

```
./spbench exec -bench <selected_bench_name_from_list> -input <input_id> ... [optional_
↪commands] <optional_args>
```

Note: For GrPPI benchmarks you must select a backend using a custom user argument. Available backends: 'omp' (OpenMP), 'thr' (C++11 threads), 'ff' (FastFlow), and 'tbb'.

```
./spbench exec -bench <grppi_bench> -input <input_id> ... -user-arg <backend>
```

Check *Command-Line Interface* for other options.

3.3.5 Inputs

In the exec step you must choose a given input for the benchmark.

These inputs can be downloaded and will be stored in the inputs/ directory.

You can run the `./spbench download-inputs` to automatically download them.

You can use your own input or choose a given one.

You can check the given inputs through:

```
./spbench list-inputs
```

See the *SPBench workloads* page for more details about workloads.

Tip:

- You can run `source setup_vars.sh` to load the libraries.
- You must load the libraries before using SPBench every time you star a new session.

3.4 SPBench Applications

Table of Contents

The current applications set of SPBench comprises four real-world applications: Ferret (from PARSEC), Lane Detection, Person Recognition, and Bzip2. These applications have been studied and used as benchmarks by many works in the literature. This way, we reimplemented them to fit the SPBench API. Our plan is to include more stream processing applications in the near future.

3.4.1 Ferret (PARSEC)

Ferret is a PARSEC application intended for a content similarity search in data such as video, audio, and images. In PARSEC the parallelism is implemented with POSIX Threads using a six-stage Pipeline. The first and last ones are source and sink. The second stage performs the segmentation, a process that organizes the regions of each image into sets. The third stage extracts 14 features from the segmented regions of each image. The fourth stage (vectorization) applies an indexing method, which selects possible similar images. The fifth stage performs a refined search of the images selected in the fourth stage, ranking the most similar images.

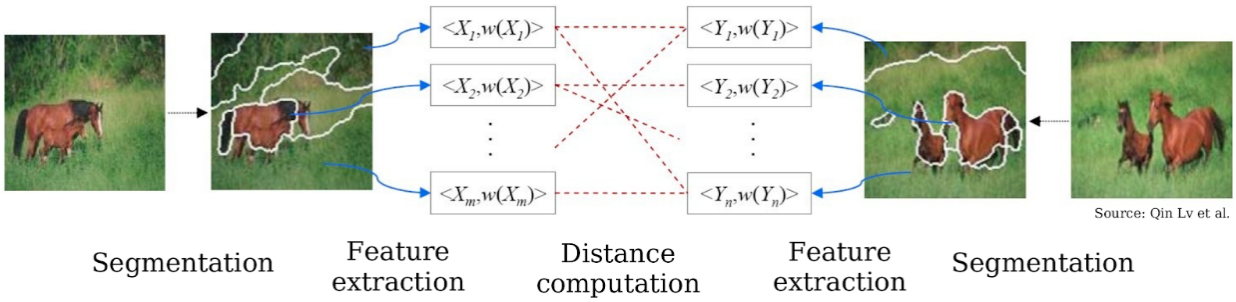


Fig. 1: Ferret illustration

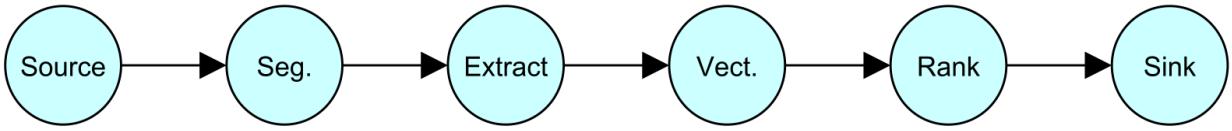


Fig. 2: Ferret Pipeline Graph

3.4.2 Lane Detection

Lane Detection is the task of detecting lanes of a road from a camera device. It can be used to keep autonomous cars on track to avoid collisions, for instance. This application captures each frame of a input video file and applies three computer vision algorithms. It can be divided into a nine-stage Pipeline, as shown in Figure 3.c. Through these stages, the detected lanes are marked with straight lines in a new frame. This new frame with the marked lanes is then overlaid on the original, and the resulting frame is written to the output file. More details about this application can be found in this link.

Fig. 3: Lane Detection illustration

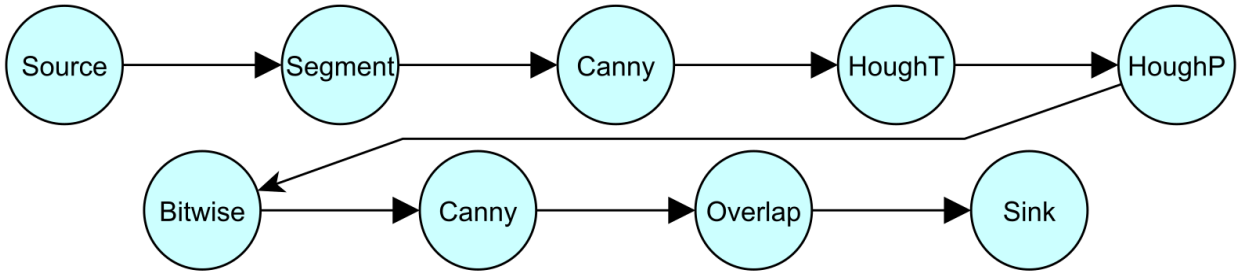


Fig. 4: Lane Detection Pipeline Graph

3.4.3 Person Recognition

Person Recognition (or Face Recognizer) tries to match human faces from a video frame against a database of faces. It could be used to recognize faces in a crowd and compare against a police database to identify wanted criminals, for instance. For each video frame it applies a detection algorithm to detect all the faces in it. Then, it uses a set of face images and compares each of the detected faces in the frame with the faces on that set. The recognized faces are marked with a circle, and then the frames are written to the output file. Therefore, this application can be divided into a four-stage Pipeline. More details about this application can be found in this link.

Fig. 5: Person Recognition illustration

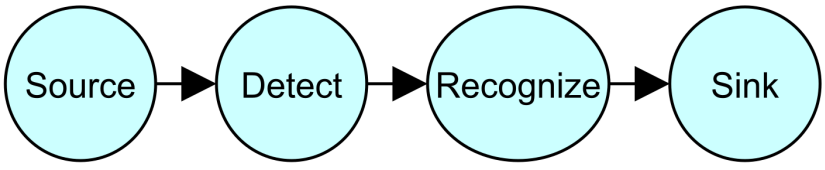


Fig. 6: Person Recognition Pipeline Graph

3.4.4 Bzip2

Bzip2 is a free and open-source data compression application. It compresses data more effectively than the older LZW (.Z) and Deflate (.zip and .gz) algorithms, but it is considerably slower than them. Therefore, parallel implementations that increase compression speed are useful. This application can be divided into a three-stage Pipeline (source, compress/decompress, sink) and has two operation modes: compress and decompress.

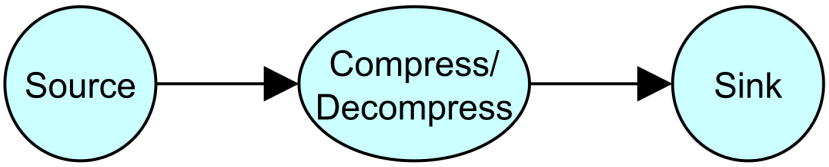


Fig. 7: Bzip2 Pipeline Graph

3.5 SPBench Benchmarks

Table of Contents

3.5.1 Single source benchmarks

These are the commom benchmarks. They read data from a single input and generate a single stream of input and output data. The source operator runs inside the stream region, as shown in the example below.

```
int main (int argc, char* argv[]){

   /* Init of stream region */
   while(1){
      spb::Item item;
      if(!spb::Source::op(item)) break;
      spb::First::op(item);
      spb::Second::op(item);
      spb::Nth::op(item);
      spb::Sink::op(item);
   } // End of stream region

   return 0;
}
```

3.5.2 Multiple source benchmarks

These are benchmarks that can read data from different inputs and generate multiple input/output streams. Here, the n source operators are run each by a exclusive thread outside the stream region. Each source has its own queue with items that can be accessed by a getItem() method inside the stream region, as shown in the example below.

```
int main (int argc, char* argv[]){

   spb::Source source1;
   spb::Source source2;

   source1.init();
   source2.init();

   /* Stream region */
   while(!(source1.depleted() && source2.depleted())){

      spb::Item item1 = source1.getItem();
      spb::Item item2 = source2.getItem();

      spb::First::op(item1);
      spb::First::op(item2);

      spb::Nth::op(item1);
      spb::Nth::op(item2);

      spb::Sink::op(item1);
      spb::Sink::op(item2);
   }
   return 0;
}
```

3.6 How to install

Table of Contents

3.6.1 SPBench Dependencies

The folowing packages are required to install and use SPBench:

- build-essential
- pkg-config
- cmake
- python (tested on python3 and python2.7)

You can run the command below to install theses packages or ask a system admin.

```
sudo apt-get install -y build-essential pkg-config cmake python3
```

3.6.2 Applications Installation

To install SPBench you can choose full installation (all applications):

```
./spbench install
```

Or choose a specific application:

```
./spbench install -app <application_name>
```

Run `./spbench install -h` for more info.

To install specific components, see `libs/README.md` or `ppis/README.md`.

3.6.3 Workload installation

The workloads (applications inputs) natively provided by SPBench to run the applications are not automatically obtained during installation. So at this point, you may also want to download and install the workloads.

```
./spbench download-inputs
```

Or choose a specific application:

```
./spbench download-inputs -app <application_name>
```

Run `./spbench download-inputs -h` to see more options.

And see the *SPBench workloads* page for more details.

3.7 Command-Line Interface

Table of Contents

The SPBench framework works with a command-line interface (CLI).

The CLI offers some commands. You can check them running:

```
./spbench --help
```

For each command there is also a "help" guide. You can access it using:

```
./spbench [command] -h
```

3.7.1 Install

`./spbench install -h`

Description:
It install the dependencies for the SPBench benchmarks. See installation for more information.

Optional arguments:

- `-app <some_application>`
  Install dependencies only for the seleted application.

3.7.2 Update

Description:

It updates the Makefiles with new compiling configurations or create a new Makefile if there is no one.

You should run it every time you change the SPBench path or SPBench version, after a SPBench update, or after running the `configure` command or any similar case. If in doubt, run it!

Note: Do not forget to recompile the benchmarks for the command `update` to take effect.

Usage examples:

For all applications:

- `./spbench update`

For a specific benchmark:

- `./spbench update -bench <benchmark_name>`

For all benchmarks based on a specific application:

- `./spbench update -app <application_name>`

For all benchmarks implemented with a specific PPI:

- `./spbench update -ppi <ppi_name>`

For more info:

- `./spbench update -h`

3.7.3 New

`./spbench new -h`

Description:
It allows for creating a new benchmark based on a SPBench application. This command will create a fresh copy of the selected application and add it to the database.

Required arguments:

- `-bench <new_benchmark_id>`
  Set a name for your new benchmark.

- `-app <application_name>`
  You must insert an application ID. See (**applications page link**) for more details.
- `-ppi <any_ppi>`
  We recommend inserting a name for the PPI you will use here. But you are free to insert any identification word.

Optional arguments:

- `-from <other_benchmark_id>` **(Optional)**
  Use this option to create your new benchmark as a copy of an existent benchmark. Insert the name of the benchmark you want to copy from
- `-nsources`
  Use this option to create a new benchmark with support for multiple sources

Usage example:

- `./spbench new -bench <new_bench_name> -app <application_name> -ppi <ppi_name>`

3.7.4 Edit

`./spbench edit -h`

Description:
It opens a choosen benchmark for editing the source code.

Required arguments:

- `-bench <benchmark_id>`

Optional arguments:

- `-editor <editor_name>` **(Default: nano)**
  You can select a different text editor available in your machine.

Usage example:

- `./spbench edit -bench <benchmark_name> -editor <text_editor_name>`

3.7.5 Configure

`./spbench configure -h`

Description:
It allows users to add compiling dependencies for a benchmark. It opens a JSON configuration file for users to write the required dependencies.

Required arguments:

- `-bench <benchmark_id>`

Optional arguments:

- `-editor <editor_name>` **(Default: nano)**
  You can select a different text editor available in your machine.

Usage example:

- `./spbench configure -bench <benchmark_name> -editor <text_editor_name>`

3.7.6 Compile

`./spbench compile -h`

Description:
It allows users to compile the benchmarks.

Required arguments:

- **`-bench <benchmark_id>` or `-ppi <ppi_id>` or `-app <application_id>`**
  Name ID of the benchmark(s) to compile. These arguments can be combined. Example:
  - `-bench my_awesome_bench` compiles the benchmark called 'my_awesome_bench'.
  - `-bench all` compiles all benchmarks.
  - `-app ferret` compiles all ferret benchmarks.
  - `-ppi tbb` compiles all benchmarks parallelized with TBB.
  - `-app ferret -ppi tbb` compiles all ferret benchmarks parallelized with TBB.

Optional arguments:

- **`-editor <editor_name>` (Default: nano)**
  You can select a different text editor available in your machine.
- **`-clean`**
  Removes all the object files that had been created for the selected benchmark(s) before compile it.

Usage example:

- `./spbench compile -bench <benchmark_id> -editor <text_editor_name>`

3.7.7 Exec

`./spbench exec -h`

Description:
This command runs the given benchmarks.

Required arguments:

- **`-bench <benchmark_id>` or `-ppi <ppi_id>` or `-app <application_id>`**
  Name ID of the benchmark(s) to execute. These arguments can be combined. Example:
  - `-bench my_awesome_bench` executes the benchmark called 'my_awesome_bench'.
  - `-bench all` executes all benchmarks.
  - `-app ferret` executes all ferret benchmarks.
  - `-ppi tbb` executes all benchmarks parallelized with TBB.
  - `-app ferret -ppi tbb` executes all ferret benchmarks parallelized with TBB.

- `-input <input_id>`
  Insert an input ID (run `./spbench list-inputs`). You can also insert multiple inputs for multiple source benchmarks.

Optional arguments:

- `-batch <batch_size>`
  It can be used to change the batch size (default: 1).
- `-batch-interval <time_interval_sec>`
  It can be used to change the batch time interval. Default = 0 (disabled).
- `-frequency <items_per_second>`
  A limited number of items will be available for the source per second. By default, no limit is aplied, which is equivalent to maximum frequency. The maximum frequency is defined by your system's memory or disk access speed.
- `-freq-pattern <frequency_pattern>`
  (pattern,period,max,min). This argument overwrites the -frequency command and sets a varying frequency for the input stream. For this argument users need to supply a tuple string: <pattern,period,max,min>, where pattern can be "wave", "spike", "binary", "increasing" or "decreasing". Period is the duration in seconds of one cycle of the pattern. And min is the minimum frequency and max the maximum frequency, defined in items per second. For the spike pattern users can also set the duration of the spikes as percentage of the period (0-100) (e.g. <pattern,period,max,min,spike>). The default spike duration is 10 percent of the period. Usage example: `-freq-pattern wave,10,5,80`
- `-nthreads <number_of_threads>`
  Number of threads. It can be used inside the benchmarks as a global variable. You can insert a single number or a range using the ':' character (e.g. `-nthreads 1:10`). You can also define iteration step of the range (e.g. `-nthreads 4:4:16` would run the benchmarks with 4, 8, 12, and 16 threads). Default value: 1.
- `-in-memory`
  It runs the application in in-memory mode.
- `-latency`
  Print average latency results at the end of the execution.
- `-throughput`
  Print the average throughput at the end of the execution.
- `-resource-usage`
  It prints the global memory and CPU usage at the end of the execution. To use it is required to run the benchmark as root or adjust paranoid value.
- `-monitor <time_interval>` **(millisecond)**
  Use this option to monitor latency, throughput, CPU and memory usage. You must indicate a monitoring time interval in milliseconds (e.g. 250).
- `-monitor-thread <time_interval>` **(millisecond)**
  Use this option to monitor latency, throughput, CPU and memory usage. It works the same as the `-monitor` command above, but it runs the monitoring computations in an individual thread. It may be used for decreased performance overhead in the application when monitoring with low time intervals.
- `-latency-monitor`
  It monitors latency per stage and per item and generates a log file.
- `-user-arg <user_custom_argument>`
  User custom argument. Allow users to pass one or more arguments to use inside the benchmarks. Single argument: [-user-arg single_argument] or [-user-arg "single argument"]. Multiple arguments: [-user-arg two arguments] or [-user-arg two -user-args arguments].

- `-test-result`
  This option will run a correctness test on the output (when available) after the benchmark execution.
- `-repeat <number_of_repetitions>`
  Use this argument to repeat this execution n times. It will compute and show a summary of the selected metrics at the end.
- `-print`
  When this parameter is added, the exec command will only print on the screen the execution command it would use to run the benchmark. You may use it to manually change some specific aspect of the execution that is not supported by SPBench.
- `-quiet`
  Run it in quiet mode. If any metrics are selected, it shows the results in a short format.
- `executor <executor_name_and_parameters>`
  You can use this argument to change the way the benchmarks are executed. Usage example: `./spbench exec -executor "-my_executor my_executor_parameters" -bench ...`
- **`-d` (Bzip2 only)**
  Enables decompress mode for Bzip2 benchmarks.

Usage example:

- `./spbench exec -bench ferret_tbb_farm -input small -nthreads 8`
  Simple run.
- `./spbench exec -bench lane_sequential -input medium -latency -throughput -resource-usage -test-result -repeat 5`
  Running with basic metrics, result testing, and repeating the execution 5 times.
- `./spbench exec -bench ferret_grppi_farm -input test -latency -throughput -nthreads 8:2:16 -user-arg omp`
  Running a Ferret GrPPI benchmark with 8, 10, 12, 14, and 16 threads, and selecting the OpenMP backend through a custom user argument.
- `./spbench exec -bench ferret_sequential -input test -executor "gdb"`
  Running a benchmark with GDB debugger as executor.
- `./spbench exec -bench bzip2_ff_farm -d -input large -nth 6 -latency-monitor -monitor 300`
  Running Bzip2 in decompress mode with monitoring metrics enabled.
- `./spbench exec -bench person_seq_ns -input large -input my_custom_input -in-memory`
  Running multiple source Person Recognition benchmark with two inputs and in-memory enabled.
- `./spbench exec -bench my_custom_bench -input small -batch 4 -frequency 50 -user-arg "My awesome argument"`
  Running a custom benchmark with batch size = 4 and maximum of 50 input items per second, plus a custom argument.
- `./spbench exec -bench my_custom_bench -input huge -batch-time 1.2 -freq-patt binary,7,10.5,70.8`
  Running a custom benchmark with the workload class huge, batch time interval = 1.2 sec., and a binary frequency pattern that oscilates the frequency between 10.5 and 70.8 every 7 seconds.

Tip: If you use repetitions (`-repeat <n>`) and `-nthreads <start:step:end>` in range mode together in the exec command, a specific resulting log will be automatically generated inside the log folder. This log file will store average latency, throughput, and exec. time, with their respective std. deviation side by side. It may be handy for running experiments and saving time gathering resulting data. The resulting log file is ready for users to draw charts over the results.

Usage example:

- `./spbench exec -bench ferret_ff_farm -input test -latency -throughput -nthreads 1:2:10 -repeat 3`

You can find the resulting log file for this usage case in the `log/ferret_ff_farm_1-2-10.dat` file.

3.7.8 List

`./spbench list -h`

Description:
This command lists all SPBench benchmarks, including user's custom benchmarks.

Optional arguments:

- **`-app <application_name>`**
  Lists only benchmarks based on the given application.

Usage example:

- `./spbench list`

3.7.9 Delete

`./spbench delete -h`

Description:
It completely deletes a given benchmark.

Required arguments:

- **`-bench <benchmark_id>`**
  Name of the benchmark to delete.

Usage example:

- `./spbench delete -bench my_not_so_awesome_bench`

3.7.10 Rename

`./spbench rename -h`

Description:
This command can be used to change the name of a given benchmark inside SPBench.

Required arguments:

- **`-old-name <old_bench_id>`**
  Insert the old and existent benchmark ID.
- **`-new-name <new_bench_id>`**
  Insert the new ID.

Usage example:

- `./spbench -old my_certainly_working_bench -new my_doubtfully_working_bench`

3.7.11 New-input

`./spbench new-input -h`

Description:
It allows for adding an alias for a custom input. The chosen alias will be registered for future uses. You must use this option to run the benchmarks using other workloads than the ones provided by SPBench.

Required arguments:

- **`-id <new_input_id>`**
  Set a name for your new input (existent entries will be replaced)
- **`-app <application_name>`**
  Insert an application from the SPBench suite.
- **`-input "<input_string>"`**
  You must insert the input string (e.g., "/home/user/my_awesome_input_file ...")

Optional arguments:

- **`-md5 <md5_hash_for_result_testing>`**
  SPBench uses the associated md5 hash to check the correctness of the result for this input (Optional). ATTENTION: do not insert the input file resulting md5 hash here. You must insert the md5 hash regarding the resulting output file. We strongly recommend that you run your new input with a sequential benchmark to compute this hash of the resulting output file.

Usage example:

- `./spbench new-input -id immensely_huge -app bzip2 -input "/home/me/awesome_songs/never_gonna_give_you_up-remix.mp3"`

3.7.12 List-inputs

`./spbench list-inputs -h`

Description:
This command lists all application's inputs provided by SPBench, including users custom ones.

Optional arguments:

- **`-app <application_name>`**
  Lists only inputs based on the given application.

Usage example:

- `./spbench list-inputs`

3.7.13 Delete-input

`./spbench delete-input -h`

Description:
Delete a given input ID from the SPBench inputs registry. It does not delete the associated files.

Required arguments:

- `-id <new_input_id>`
  Insert the ID of the input to delete.
- `-app <application_name>`
  Insert the associated app for the given input.

Usage example:

- `./spbench delete-input -id my_super_extra_medium_input -app lane_detection`

3.7.14 Download-inputs

`./spbench download-inputs -h`

Description:
Download the inputs for the SPBench benchmarks.

Optional arguments:

- `-app <application_name>`
  Use this option for downloading inputs for a specific application.
- `-class <workload_class>`
  You can insert the class of workload you want to download (leave it empty to download all classes). Possible options: all, test, small, medium, large, and huge.
- `-force`
  Use this option to delete any respective input file and download it again.

Usage example:

- `./spbench download-inputs` **(recommended)**
  It will download everything.
- `./spbench download-inputs -app bzip2 -class small -force`
  Bzip2 has some large files to download, so users can use this option to avoid downloading everything.

3.7.15 Edit-operator

`./spbench edit-op -h`

Description:
Open an operator of a given benchmark to edit its source code.

Required arguments:

- `-benchmark <benchmark_id>`
  Enter the name of an application benchmark to edit (mandatory)

- `-operator <operator_id>`
  Enter the name of the benchmark's operator to edit (mandatory)

Optional arguments:

- `-editor <editor_name>`
  Enter the name of a text editor available in your machine, e.g. "vim", "nano", etc. (default: nano)

Usage example:

- `./spbench edit-op -bench ferret_sequential -op segmentation -editor vi`

3.7.16 List-operators

`./spbench edit-op -h`

Description:
List all available operators for a given benchmark.

Required arguments:

- `-benchmark <benchmark_id>`
  You must give the name of the respective benchmark you want to see the operators list (mandatory).

Usage example:

- `./spbench list-op -bench my_awesome_bench`

3.7.17 Reset-operators

`./spbench reset-op -h`

Description:
Reset to default the source code of all operators for a given benchmark (it still keeps the old operators as backup until this command is executed again).

Required arguments (at least one of them must be used):

- `-benchmark <benchmark_id>`
  Insert the name of a benchmark to reset its operators. Caution: It will remove any current modifications on the operators. You can use [-bench all] to reset the operators for all benchmarks.
- `-app <app_id>`
  Reset operators for all benchmarks based on a specific given application.
- `-ppi <ppi_id>`
  Reset operators for all benchmarks implemented using a specific given PPI.

Usage example:

- `./spbench reset-op -bench my_awesome_bench`

3.7.18 Clean

`./spbench clean -h`

Description:
Clean a benchmark (similar to make clean). It also removes old operators.

Required arguments (at least one of them must be used):

- `-benchmark <benchmark_id>`
  Insert the name of a benchmark to clean (similar to make clean). You can use 'all' to clean all benchmarks.
- `-app <app_id>`
  Clean all benchmarks based on a specific given application.
- `-ppi <ppi_id>`
  Clean all benchmarks implemented using a specific given PPI.
- `-logs <ppi_id>`
  Delete all execution logs of SPBench.
- `-outputs <ppi_id>`
  Delete all the outputs generated by the benchmarks

Usage example:

- `./spbench clean -bench all`
  Clean all benchmarks.
- `./spbench clean -app lane_detection -logs`
  Clean all lane detection benchmarks and all SPBench logs
- `./spbench clean -ppi <my_awesome_ppi> -outputs`
  Clean all benchmarks using a specific PPI and the outputs from all benchmarks.
- `./spbench clean -logs -outputs`
  Delete all SPBench logs and the outputs from all benchmarks.

3.8 Compiling Configuration

Table of Contents

Each SPBench benchmark has a configuration file that includes its parameters and build dependencies. Also, SPBEnch provides a global configuration file, that enables adding extra compiling parameters or override the benchmark-specific ones. SPBench reads these files and generates a makefile specific to the respective benchmark. Some triggers activate the generation of a new makefile. It happens whenever one of the following situations occurs:

- When the `./spbench compile` command is executed and there is no makefile yet.
- When the user runs the `./spbench configure` command and modifies the JSON file (if nothing is changed, a new makefile will not be generated).
- When the user runs the `./spbench update` command.
- When the user runs the `./spbench global-config` command and modifies the JSON file.

Run `./spbench -h` for more info about the commands.

> **Warning:** In the `./spbench configure` command, the trigger for generating a new makefile is to exit the text editor. So realize that if you use a text editor that will not exit or was not opened via the `./spbench configure ... -editor <my_text_editor>` command, the new makefile will not be generated automatically every time you change the file. You will always need to run the 'update' command if you want to generate an updated makefile in these cases.

3.8.1 JSON Configuration File

Each benchmark has a local and exclusive compiling configuration file. This is the structure of a local JSON configuration:

```
{
   "CXX"           : "<compiler>",
   "CXX_FLAGS"     : "<compiler flags>",
   "PPI_CXX"       : "<custom PPI compiler>",
   "PPI_CXX_FLAGS": "<custom PPI compiler flags>",
   "PRE_SRC_CMD"   : "<specific commands added right before the benchmark source file>",
   "POST_SRC_CMD" : "<specific commands added right after the benchmark source file>",
   "MACROS"        : "<compiling macros>",
   "PKG-CONFIG":{
      "myPKG_1"    : "<package-config command>",
      "myPKG_2"    : "...",
      "myPKG_N"    : "..."
   },
   "INCLUDES":{
      "myINC_1"    : "-I <my includes>",
      "myINC_2"    : "...",
      "myINC_N"    : "..."
   },
   "LIBS":{
      "myLIB_1"    : "-L <my libs>",
      "myLIB_2"    : "...",
      "myLIB_N"    : "..."
   },
   "LDFLAGS"       : "-my_lib1 -my_lib2 ..."
}
```

The compiler defined in CXX is the compiler used to compile the SPBench source code and the backend applications. PPI_CXX is the compiler that will be used to compile the code that the user has access to. If no custom compiler is defined for PPI_CXX, the CXX compiler will be used by default.

Warning: SPBench recognizes all these keywords, but will not recognize others. So be careful not to change the keys. The exception are the subkeys from PKG-CONFIG, INCLUDES, and LIBS.

3.8.2 Example

The example below shows the configuration file of a benchmark implemented with the SPar PPI. Note that SPar has its custom compiler and its flags. For SPar it is also necessary to add a flag in front of the source code to be compiled. It is a case where PRE_SRC_CMD needs to be used. Also, note the syntax used to add pkg-config. You must not add the encapsulation of the command, only the parameters.

```
{
   "CXX"           : "g++",
   "CXX_FLAGS"     : "-std=c++1y -O3",
   "PPI_CXX"       : "$BENCH_DIR/ppis/SPar/bin/spar",
   "PPI_CXX_FLAGS": "-O3 -spar_ordered -spar_ondemand -spar_blocking",
   "PRE_SRC_CMD"   : "-spar_file",
   "POST_SRC_CMD" : "",
   "MACROS"        : "-DNO_DEFAULT_MAPPING",
   "PKG-CONFIG":{
      "opencv"     : "pkg-config --cflags --libs opencv"
   },
   "INCLUDES":{
      "opencv"     : "-I $BENCH_DIR/libs/opencv/opencv-2.4.13.6/include/",
      "UPL"        : "-I $BENCH_DIR/libs/upl/include/upl/"
   },
   "LIBS":{
      "UPL"        : "-L $BENCH_DIR/libs/upl/lib/x86 -lupl"
   },
   "LDFLAGS"       : "-lpthread"
}
```

3.8.3 Global Compiling Configuration

The global JSON configuration file is similar to the benchmark-specific one, but it adds some extra keys.

The basic rule is that the JSON keys in the global configuration file override the local ones. The exception are keys starting with "EXTRA", which are recognized only in the global JSON. These keys do not override local keys but add to them instead. Let us see what would happen if a user used the global configuration file below.

```
{
   "CXX"                : "clang",
   "CXX_FLAGS"          : "",
   "EXTRA_CXX_FLAGS"    : "-g",
   "PPI_CXX"            : "",
   "PPI_CXX_FLAGS"      : "-O3",
   "EXTRA_PPI_CXX_FLAGS": "",
   "MACROS"             : "",
   "EXTRA_MACROS"       : "-DNO_UPL",
   "PKG-CONFIG":{
      "myPKG"           : ""
```

(continues on next page)

(continued from previous page)

```
    },
    "INCLUDES":{
        "myINC"             : "-I /my/awesome/library/"
    },
    "LIBS":{
        "myLIB_N"           : ""
    },
    "LDFLAGS"               : "",
    "EXTRA_LDFLAGS"         : "-lpthread"
}
```

First, the entire SPBench infrastructure would be compiled with the clang compiler since it would completely overwrite GCC. Then it would add '-g' to the existing compilation flags of each benchmark. Next, it would replace all the compile flags in the user code with '-O3'. It would add the 'NO_UPL' macro to the macros of each benchmark. It would do nothing concerning pkg-config since nothing was added. If the local configuration file of a benchmark had an include with the same key 'myINC', this include would be overwritten by the global JSON include. Otherwise, 'myINC' would be added to the local includes. Finally, all benchmarks would be compiled with '-lpthread', in addition to their original LDFLAGS.

Note: The contents of both global and local configuration files are converted directly into a Makefile. The global JSON does not modify the local JSON at any time. Nor is any resulting JSON created. So you need not be afraid of losing the local build configurations.

3.8.4 UPL

You can see in the example above that the configuration file includes the UPL library. UPL stands for Utility Performance Library. It is a static library that provides a set of functions useful for measuring performance in C/C++ programs (more info here).

In SPBench, we use UPL for measuring CPU and memory usage.

That is why it is included in the configuration file. However, this library is only available for **x86** and **arm** architectures. The benchmarks default include the x86 version, as seen in the configuration file above. However, you can change `upl/lib/x86` to `upl/lib/arm` if you want to run SPBench on an arm processor.

It may not be possible to use UPL on other architectures at the moment, so if this is the case, you can disable UPL by adding the macro `-DNO_UPL` in the configuration file. You will not be able to evaluate CPU and memory usage. However, the other metrics will be available.

Tip: `$BENCH_DIR` defines the root of the SPBench. It is basically an SPBench keyword. If you want, you can also use it to define the paths that you insert into configuration files.

Tip: The macro `-DNO_DEFAULT_MAPPING` is used to disable FastFlow/SPar thread pinning. This macro is active by default in all benchmarks. FastFlow developers recommend not using this pinning if running in a hyper-threading environment. This thread pinning serves to allocate the threads to the processor's physical cores first. But if you want to use it, to make FastFlow/SPar properly recognize and map the cores of the processor you must run at SPBench root `bash ./ppis/SPar/libraries/ff/mapping_string.sh` and `bash ./ppis/fastflow/ff/mapping_string.sh`. It is only required once. However, this FastFlow feature may not be able to automatically set this value in the `./ppis/../ff/config.sh` file. In this case, you must manually update this `config.hpp` file with the execution result of the

---

`mapping_string.sh` script.

3.9 SPBench workloads

Table of Contents

3.9.1 Workload Classes

SPBench provides different classes of workloads for each application. In this version all applications have the classes test, small, medium, large, and huge.

Note that the workloads provided by the SPBench need to be downloaded from a remote repository before first use. To do this, you must run the download-inputs command.

- `./spbench download-inputs`

This command will download all input classes for all applications. It may take several minutes for large workloads, depending on your internet speed.

You can choose to download the workload classes for a single application.

- `./spbench download-inputs -app <application_name>`

And you can also select a single workload class.

- `./spbench download-inputs -app <application_name> -class <workload_class>`

For more info, run:

- `./spbench download-inputs -h`

Lane Detection

The Lane Detection input video is a recording from a car driving on a highway. It is an mp4 video with 640×360 or 1280×720 resolution. This same video is used for all workload classes, but at different lengths between them.

- **Test class**: 3-second video
- **Small class**: 15-second video
- **Medium class**: 30-second video
- **Large class**: 60-second video
- **Huge class**: 120-second video

The default resolution is 360p. To access the 720p videos users can add a '-HQ' sufix (e.g. large-HQ).

Bzip2

The workloads for Bzip2 are dump files from the wikipedia database. The files are as follows:

- **Test class**: enwiki-20211120-pages-articles-multistream-index9.txt
- **Small class**: enwiki-20220601-pages-articles-multistream15.xml
- **Medium class**: enwiki-20211120-all-titles-in-ns0
- **Large class**: enwiki-20211120-pages-articles-multistream9
- **Huge class**: enwiki-20211120-pages-articles-multistream9-2x.xml

Users can use the class names with the '_d' sufix (e.g. large_d) to access the compressed versions for running the benchmarks in decompress mode.

Person Recognition

The Person Recognition input files consist of a set of photos of former US president Barack Obama's face plus a video recorded during a talk that contains images of his face and also the faces of other people in the audience at some points. The set of images is used to train the application. The video is used to try to recognize the face of the former president.

All workload classes use the same video, but at different lengths, as described below:

- **Test class**: 0.3-second video
- **Small class**: 1-second video
- **Medium class**: 3-second video
- **Large class**: 15-second video
- **Huge class**: 30-second video

Ferret

For Ferret the original workloads available on the PARSEC website are used.

In SPBench the 'native' workload is available under the 'huge' alias.

3.9.2 Using external workload

To run an application with an input that is not provided by the SPBench, you must first register that input. For this you can use the new-input command (*new-input*).

E.g. `./spbench -new-input -id <my_input_name> -app <spbench_app> -input-string <"my_input_string">`

Adding result checking

The SPBench applications allow the result to be checked at the end of the execution. This is done by comparing the md5 hash of the benchmark output with a pre-computed md5 hash stored in the input register. Therefore, to add this result check also for a new external input you have registered, we recommend that you first run a benchmark in the sequential version with this input, to have a more reliable result file.

To add the md5 hash you can use the `-md5` argument.

E.g. `./spbench -new-input ... -md5 <expected_md5_hash_to_check>`

> **Warning:** It is this **resulting output file** that you should use to compute the md5 hash you will add to the input log, not the original file.

3.9.3 Workload Characterization

Forthcoming…

3.10 Performance Metrics

Table of Contents

SPBench provides the following metrics for users to evaluate the benchmarks:

- Latency
- Throughput
- CPU usage (%)

- Memory usage (KB)

These metrics can be collected at different levels, such as per time unit, per operator, per source, or global average. Some of them can be combined as well.

Also, the metrics can be collected statically (selected and configured before starting the run), or dynamically (collected at any point during the benchmark execution through function calls inside the source code).

The table below shows more details.

Metric	Granularity	Usage
Latency	Global average (end-to-end) Global average (per-operator) Per time window Per item	Dynamic and static
Throughput	Global averag Average per time window	Dynamic and static
CPU and Mem. Usage	Global average Per time interval	Static

3.10.1 Statically Selecting Performance Metrics

Statically selected metrics are the metrics that are selected as an argument through the exec command in the CLI. See *Command-Line Interface* or run `./SPBench exec -h` for more information.

Here are five optional arguments for metrics you can use when running the 'exec' command:

- `-monitor <time_interval>`

  It monitors latency, throughput, CPU and memory usage, and other parameters. Here users can indicate a monitoring time interval in milliseconds (e.g. 250). It will generate a log file inside a log folder (spbench/log/). This file will contain the results got for each metric at each time interval.

- `-latency`

  It prints the average latency results for each application operator on the screen after the execution.

- `-throughput`

  It prints the average throughput results after the execution.

- `-latency-monitor`

  It monitors the latency per stage and per item and generates a log file inside the log folder.

- `-resource-usage`

  It prints the global memory and CPU usage for the selected application.

Note: You must run the benchmark as root or adjust paranoid value to use the resource usage metric.

3.10.2 Dinamically Selecting Performance Metrics

SPBench also allows some metrics to be collected at runtime, such as throughput and latency.

These metrics are available in two modes. The first one returns the global average results from the beginning of the execution to the current moment. The second mode is the instantenous one, where the metrics are computed over a short period (time window).

To get the instantaneous throughput: `spb::Metrics::getInstantThroughput(<time_window_in_sec>)`

To get the average throughput: `spb::Metrics::getAverageThroughput()`

To get the instantaneous latency: `spb::Metrics::getInstantLatency(<time_window_in_sec>)`

To get the average latency: `spb::Metrics::getAverageLatency()`

These metrics are also used by the performance monitoring. There, the time window used for the instantaneous metrics is the same value used for the monitoring time interval.

3.10.3 Performance Metrics for Multi-source Benchmarks

All metrics available for single source benchmarks are also available for multi-source.

Their usage does not change for static metrics, but for the dynamic ones it still quite similar.

The difference here is that these metrics are implemented as methods from the spb::Source class, instead of spb::Metrics class.

For example:

To get the instantaneous throughput: `source.getInstantThroughput(<time_window_in_sec>)`

To get the average throughput: `source.getAverageThroughput()`

To get the instantaneous latency: `source.getInstantLatency(<time_window_in_sec>)`

To get the average latency: `source.getAverageLatency()`

3.11 Management Options

Table of Contents

SPBench allows users to manage the input stream, workload behavior, and other options in some ways. Some characteristics, such as data input frequency and batching, can be tuned statically or dynamically during the execution of a benchmark.

3.11.1 General Management

Here we present some general management options. These options can be statically set using specific arguments in the command-line through the `exec` command. See *Command-Line Interface* or run `./SPBench exec -h` for more information.

All of these are also OPTIONAL arguments that you can use within the 'exec' command.

- `-in-memory`

  It runs the application in-memory, which means that all the input is first loaded into the memory before start processing it. The opposite is true for the writing phase, where the result stays in memory and is only writen to the disk after all the input is proccessed.

- `-nthreads <number_of_threads>`

  It defines a number of threads to run the parallel benchmark available within SPBench. To be more precise, the degree of parallelism of the parallel stages (Obs.: some PPIs can create more threads than parallel stages). It shows up as a global variable that can be used inside the parallel benchmarks, so users can also use this variable to set the parallelism in their custom benchmarks.

- `-user-arg`

  Allow users to use custom arguments. Arguments that users pass on the command line via this option can be retrieved inside the benchmark code via the `spb::SPBench::getArg(<argument_index>)` method.

- `-d`

  This a specific argument for bzip2 benchmarks. You can use it to run this application in decompress mode. Observe that for de decompression mode it is required a compressed file as input (e.g. my_compressed_file.tar.bz2). The compressed input classes are registered with a '_d' at the end (e.g. small_d, large_d, etc.).

3.11.2 Batching Management

There are some ways of changing the size of batches in SPBench. The first way is to set fixed-size batches at the beginning of the execution by adding some arguments to the exec command.

- `-batch <batch_size>`

  Change the size of the batches based on a fixed number of items. By default, the size of each batch is 1. If used with `-batch-interval`, the batch size will be set according to whichever occurs first.

- `-batch-interval <time_interval_sec>`

  Change the size of the batches based on a given time period (seconds). By default, the time period is 0, which means it is disabled. If used with `-batch`, the batch size will be set according to whichever occurs first.

There are some methods of the SPbench class that users can use to dynamically access or modify the batch sizes It can be done either inside or outside the stream region of the benchmarks. Users can use the following SPBench methods for that:

- `spb::SPBench::setBatchSize(<batch_size>)`.

or

- `spb::SPBench::setBatchInterval(<time_interval_sec>)`

Usage example:

```
// Example of possible use for custom user argument
my_PPI.setQueueSize(spb::SPBench::getArg(0));

/* Stream region */
while(1){
    spb::Item item;
    if(!spb::Source::op(item)) break;
    if(/*my_condition*/){
        /* Increment the current batch size by 1 */
        spb::SPBench::setBatchSize(spb::SPBench::getBatchSize()+1);
    }
    spb::Some::op(item);
    spb::Some_other::op(item);
    spb::Sink::op(item);
} // End of stream region
```

3.11.3 Frequency Management

Frequency in SPBench means the number of items avaliable for the sources per second. Of course, there is a maximum frequency for each application, which is limited by the speed at which the items are read from the input stream. In normal executions this speed is limited by the time it takes to read the data from the disk. Therefore, to achieve high frequencies it is recommended to run the benchmark using the in-memory option, since reading data from memory is usually faster than reading from disk.

The way to manage the data input frequency in SPBench is similar to the way we manage the batching. Users can change the frequency once at the beginning of the execution or they can also change it dynamically during the execution of the benchmark.

To change it statically, users can add the following arguments in the **exec** command:

- `-frequency <items_per_second>`

  Set a frequency for items to be available for the source operator. By default, there is no limit, which means that the default frequency is the one given by the speed of reading items from the disk (or from the memory, in a in-memory execution). The frequency control in SPBench is done by adding a varying time-delay for each item generation. So higher time-delays imply in lower frequencies.

- `-freq-pattern <frequency_pattern>` (pattern,period,max,min).

  This argument overwrites the -frequency command and sets a varying frequency for the input stream. Users can select and set a frequency variation pattern. For this argument users need to supply a tuple string: <pattern,period,max,min>, where pattern can be "wave", "spike", "binary", "increasing" or "decreasing". Period is the duration in seconds of one cycle of the pattern. And min is the minimum frequency and max the maximum frequency, defined in items per second. For the spike pattern users can also set the duration of the spikes as percentage of the period (0-100) (e.g. <pattern,period,max,min,spike>). The default spike duration is 10 percent of the period. Usage example: `-freq-pattern wave,10,5,80`

These two frequency management options can also be changed at anytime during the execution by using the methods below:

- `spb::SPBench::setFrequency(<items_per_second>)`

Set a frequency for items to be available for the source operator (same as the `-frequency` argumment).

- `spb::SPBench::setFrequencyPattern(<frequency_pattern>)`

This SPBench method allows users to set and change a frequency pattern at anytime during a benchmark's execution.

Usage example:

```
// Example of frequency pattern usage
spb::SPBench::setFrequencyPattern("wave", 12, 5.3, 70);

/* Stream region */
while(1){
    spb::Item item;
    if(!spb::Source::op(item)) break;
    if(/*my_condition*/){
        /* Decrement the current frequency by 5 */
        spb::SPBench::setFrequency(spb::SPBench::getFrequency()-5); // 100 items per␣
↪second
    }
    spb::Some::op(item);
    spb::Some_other::op(item);
    spb::Sink::op(item);
} // End of stream region
```

3.11.4 Management Options for Multi-source Benchmarks

In multi-source benchmarks, each source offers to users some methods to control its behavior individually.

- `source.setBatchSize(items_per_batch)`

  This method can be used to change the batch size of the items on the stream.

- `source.setBatchInterval(batch_interval_in_sec)`

  This method can be used to change the batch time interval of the stream.

- `source.setFrequency(items_per_second)`

  If the in-memory option was not enable in the execution command, the maximum frequency will be limited by the speed of the disk. Ideally, this option may be used in an in-memory execution to achieve high speed data.

- `source.setQueueMaxSize(queue_size);`

  This option changes the queue size for te respective source. It has no impact on the inter-stage queues of the parallel implementations.

- `source.init()`

  It is necessary to run this method once before the stream region to start the source execution.

Alternativelly, these attributes can be specified through the constructor of the source object:

- `spb::Source source(<batch_size>, <batch_interval>, <queue_size>, <frequency>);`

When using this second method, `source.init()` is not required.

Code example:

```
...
    // Compact source creation method. This source will run immediately.
    // Parameters: <batch_size>, <batch_interval>, <queue_size>, <frequency>
    spb::Source source1(2, 0, 1, 0);

// Alternative source creation method
    spb::Source source2;

    // These parameters can be changed anytime during execution
    source2.setBatchSize(1); // 1 item per batch
    source2.setBatchInterval(0.5); // 500 ms batch window
    source2.setQueueMaxSize(3); // 3 slots in this source's queue
    source2.setFrequency(30); // 30 items per second

    // You must use the init() method to run the source2
// because it dos not run immediately on creation
    source2.init();

    /* Stream region */
    while(!(source1.depleted() && source2.depleted())){
    ...
    if(/*my condition*/){
        // all parameters can be changed anytime
        source1.setQueueMaxSize(4);
        source2.setFrequency(50);
    }
    ...
    }
...
```

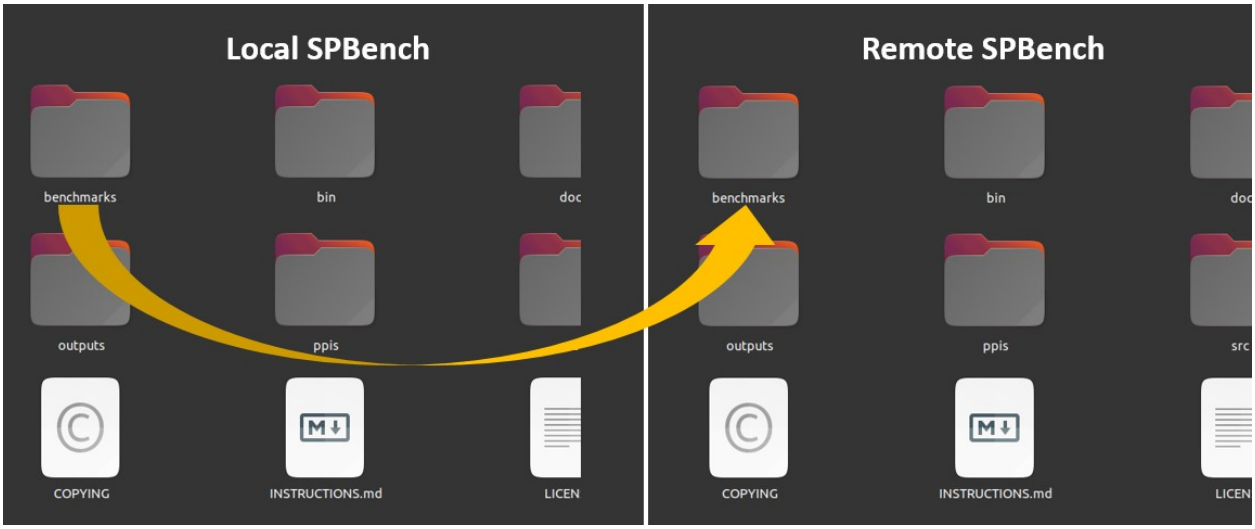
3.12 Moving SPBench

Here we discuss how to move SPBench somewhere else, which could be another directory or another computer.

There is no easy way to move the entire SPBench since all libraries are installed locally. So if its path is changed in any way, you have to reinstall all the libraries again, using the `./spbench install` command.

However, there is also a very common situation: a user has created new benchmarks in SPBench, or modified existing benchmarks and wants to move just these benchmarks to run on a remote machine.

For these scenarios, there is a simple solution. Just copy the `/benchmarks` directory and replace it in the root of SPBench on the other machine. Of course, SPBench needs to be installed on this other machine as well.



3.13 Publications

Table of Contents

This page lists the main published papers that used SPBench.

3.13.1 How To Cite

Use the following paper to cite SPBench:

- GARCIA, A. M.; GRIEBLER, D. J.; SCHEPKE, C.; FERNANDES, L. G. "SPBench: A Framework for Creating Benchmarks of Stream Processing Applications". COMPUTING, v. 1, p. 1, 2021, doi: 10.1007/s00607-021-01025-6. [link]

```
@article{GARCIA:Computing:22,
    title = {SPBench: a framework for creating benchmarks of stream processing␣
↪applications},
    author = {Adriano Marques Garcia and Dalvan Griebler and Claudio Schepke and Luiz␣
↪Gustavo Fernandes},
    url = {https://doi.org/10.1007/s00607-021-01025-6},
    doi = {10.1007/s00607-021-01025-6},
    year = {2022},
    date = {2022-01-01},
    journal = {Computing},
    volume = {In press},
    number = {In press},
    pages = {1-23},
    publisher = {Springer},
```

(continues on next page)

(continued from previous page)

```
    abstract = {In a fast-changing data-driven world, real-time data processing systems␣
↪are becoming ubiquitous in everyday applications. The increasing data we produce, such␣
↪as audio, video, image, and, text are demanding quickly and efficiently computation.␣
↪Stream Parallelism allows accelerating this computation for real-time processing. But␣
↪it is still a challenging task and most reserved for experts. In this paper, we␣
↪present SPBench, a framework for benchmarking stream processing applications. It aims␣
↪to support users with a set of real-world stream processing applications, which are␣
↪made accessible through an Application Programming Interface (API) and executable via␣
↪Command Line Interface (CLI) to create custom benchmarks. We tested SPBench by␣
↪implementing parallel benchmarks with Intel Threading Building Blocks (TBB), FastFlow,␣
↪and SPar. This evaluation provided useful insights and revealed the feasibility of the␣
↪proposed framework in terms of usage, customization, and performance analysis. SPBench␣
↪demonstrated to be a high-level, reusable, extensible, and easy of use abstraction to␣
↪build parallel stream processing benchmarks on multi-core architectures.},
    keywords = {},
    pubstate = {published},
    tppubtype = {article}
}
```

3.13.2 Other papers using SPBench

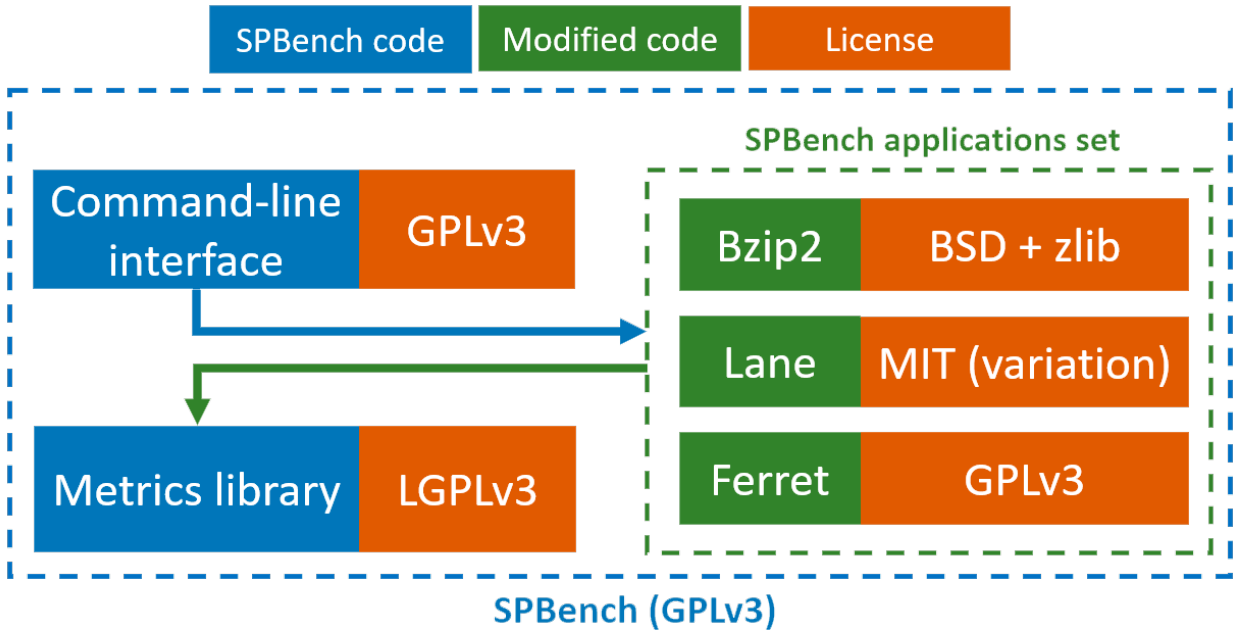
- GARCIA, A. M.; GRIEBLER, D. J.; SCHEPKE, C.; FERNANDES, L. G. "Evaluating Micro-batch and Data Frequency for Stream Processing Applications on Multi-cores". In: 2022 30th Euromicro International Conference on Parallel, Distributed and NetworkBased Processing (PDP), 2022, Valladolid. 2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2022. p. 10-17, doi: 10.1109/PDP55904.2022.00011. [link]
- GARCIA, A. M.; GRIEBLER, D. J.; SCHEPKE, C.; FERNANDES, L. G. "Introducing a Stream Processing Framework for Assessing Parallel Programming Interfaces", 2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2021, pp. 84-88, doi: 10.1109/PDP52278.2021.00021. [link]

3.14 Copying

SPBench is under GNU General Public License version 3 (see COPYING files in the source for more details). Each application included by SPBench is also under their original license. The diagram below illustrates how the licenses are assigned.

3.15 About

SPBench is a framework depeloped by Adriano Marques Garcia as part of its Ph.D. in Computer Science at PUCRS. This project have been depeloped inside the Parallel Applications Modelling Group (GMAP). It is supervised by the professors Dalvan Griebler, Claudio Schepke, and Luiz Gustavo Fernandes.



3.15.1 Collaborators

Fig. 9: Dalvan Griebler

Fig. 11: Luiz Gustavo Fernandes