benchkit:

A Declarative Framework for Composable Performance Evaluation of System Software

Antonio Paolillo **Mats Van Molle** Ken Hasselmann

RMA Brussels
The only Belgian military university

Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors

David Tam Reza Azimi Michael Stumm

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada M5S 3G4
{tamda, azimi, stumm}@eecg.toronto.edu

ABSTRACT

The major chip manufacturers have all introduced chip multiprocessing (CMP) and simultaneous multithreading (SMT) technology into their processing units. As a result, even low-end computing systems and game consoles have become shared memory multiprocessors with L1 and L2 cache sharing within a chip. Mid- and large-scale systems will have multiple processing chips and hence consist of an SMP-CMP-SMT configuration with non-uniform data sharing overheads. Current operating system schedulers are not aware of these new cache organizations, and as a result, distribute threads across processors in a way that causes many unnecessary, long-latency cross-chip cache accesses.

In this paper we describe the design and implementation of a scheme to schedule threads based on sharing patterns detected online using features of standard performance monitoring units (PMUs) available in today's processing units. The primary advantage of using the PMU infrastructure is that it is fine-grained (down to the cache line) and has relatively low overhead. We have implemented our scheme in Linux running on an *8-way* Power5 SMP-CMP-SMT multiprocessor. For commercial multithreaded server workloads (VolanoMark, SPECjbb, and RUBiS), we are able to demonstrate reductions in cross-chip cache accesses of up to 70%. These reductions lead to application-reported performance improvements of up to 7%.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*concurrency, scheduling, threads*; D.4.8 [**Operating Systems**]: Performance—*measurements, monitors*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.5.3 [**Pattern Recognition**]: Clustering—*algorithms, similarity measures*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*multiple-instruction-stream, multiple-data-stream processors*; C.5.5 [**Computer System Implementation**]: Servers; C.5.1 [**Computer System Implementation**]: Large and Medium ("Mainframe") Computers; B.3.2 [**Memory Structures**]: Design Styles—*cache memories, shared memory, virtual memory*; H.2.4 [**Database Management**]: Systems—*concurrency, parallel databases*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*

General Terms

Algorithms, Management, Measurement, Performance, Design, Experimentation

Keywords

Affinity scheduling, cache behavior, cache locality, CMP, detecting sharing, hardware performance counters, hardware performance monitors, multithreading, performance monitoring unit, resource allocation, shared caches, sharing, simultaneous multithreading, single-chip multiprocessors, SMP, SMT, thread migration, thread placement, thread scheduling

EuroSys'07, March 21–23, 2007, Lisboa, Portugal.

1. INTRODUCTION

With diminishing potential improvements in clock speeds, processor chip manufacturers have turned towards increasing parallelism to obtain further performance gains. The major chip manufacturers have all introduced chip multiprocessing (CMP) and simultaneous multithreading (SMT) technology over the last several years for their laptop, desktop, and server processing units. As a result, even low cost computing systems and game consoles have become shared memory multiprocessors. Small- to medium-sized systems will contain a small number of processing chips (e.g., 1 to 4), while the number of cores and hardware threads in each core will likely increase over the next few years. For example, the Sun Niagara chip currently has 32 hardware contexts.

A key difference between the more traditional small-scale shared memory multiprocessors (SMPs) and these newer systems is that the latter have non-uniform data sharing overheads; i.e., the overhead of data sharing between two processing components differs depending on their physical location. For the processing units that reside on the same CPU core (i.e., hardware virtual contexts), communication

Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors

David Tam Reza Azimi Michael Stumm

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada M5S 3G4
{tamda, azimi, stumm}@eecg.toronto.edu

ABSTRACT

The major chip manufacturers have all introduced chip multiprocessing (CMP) and simultaneous multithreading (SMT) technology into their processing units. As a result, even low-end computing systems and game consoles have become shared memory multiprocessors with L1 and L2 cache sharing within a chip. Mid- and large-scale systems will have multiple processing chips and hence consist of an SMP-CMP-SMT configuration with non-uniform data sharing overheads. Current operating system schedulers are not aware of these new cache organizations, and as a result, distribute threads across processors in a way that causes many unnecessary, long-latency cross-chip cache accesses.

In this paper we describe the design and implementation of a scheme to schedule threads based on sharing patterns detected online using features of standard performance monitoring units (PMUs) available in today's processing units. The primary advantage of using the PMU infrastructure is that it is fine-grained (down to the cache line) and has relatively low overhead. We have implemented our scheme in Linux running on an *8-way* Power5 SMP-CMP-SMT multiprocessor. For commercial multithreaded server workloads (VolanoMark, SPECjbb, and RUBiS), we are able to demonstrate reductions in cross-chip cache accesses of up to 70%. These reductions lead to application-reported performance improvements of up to 7%.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*concurrency, scheduling, threads*; D.4.8 [**Operating Systems**]: Performance—*measurements, monitors*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.5.3 [**Pattern Recognition**]: Clustering—*algorithms, similarity measures*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*multiple-instruction-stream, multiple-data-stream processors*; C.5.5 [**Computer System Implementation**]: Servers; C.5.1 [**Computer System Implementation**]: Large and Medium ("Mainframe") Computers; B.3.2 [**Memory Structures**]: Design Styles—*cache memories, shared memory, virtual memory*; H.2.4 [**Database Management**]: Systems—*concurrency, parallel databases*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*

General Terms

Algorithms, Management, Measurement, Performance, Design, Experimentation

Keywords

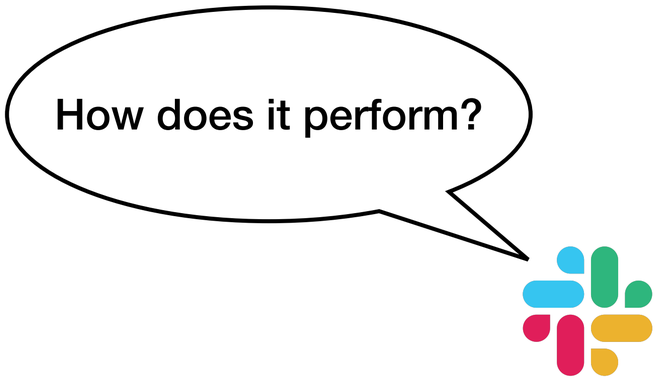
Affinity scheduling, cache behavior, cache locality, CMP, detecting sharing, hardware performance counters, hardware performance monitors, multithreading, performance monitoring unit, resource allocation, shared caches, sharing, simultaneous multithreading, single-chip multiprocessors, SMP, SMT, thread migration, thread placement, thread scheduling

EuroSys'07, March 21–23, 2007, Lisboa, Portugal.

1. INTRODUCTION

With diminishing potential improvements in clock speeds, processor chip manufacturers have turned towards increasing parallelism to obtain further performance gains. The major chip manufacturers have all introduced chip multiprocessing (CMP) and simultaneous multithreading (SMT) technology over the last several years for their laptop, desktop, and server processing units. As a result, even low cost computing systems and game consoles have become shared memory multiprocessors. Small- to medium-sized systems will contain a small number of processing chips (e.g., 1 to 4), while the number of cores and hardware threads in each core will likely increase over the next few years. For example, the Sun Niagara chip currently has 32 hardware contexts.

A key difference between the more traditional small-scale shared memory multiprocessors (SMPs) and these newer systems is that the latter have non-uniform data sharing overheads; i.e., the overhead of data sharing between two processing components differs depending on their physical location. For the processing units that reside on the same CPU core (i.e., hardware virtual contexts), communication



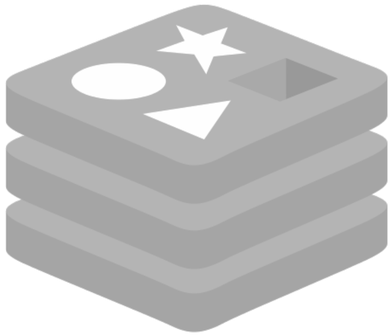

MySQL


京都
キャビネット
8 EiB

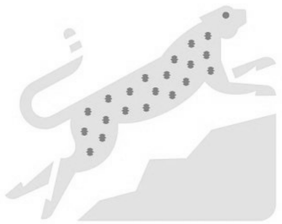

levelDB


redis


RocksDB



redis

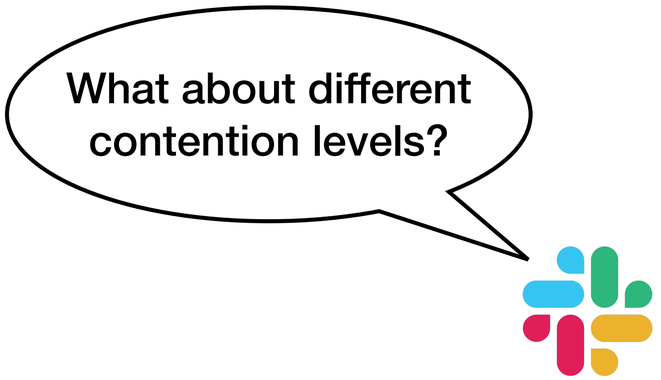


RocksDB

```
$ git clone --recurse-submodules https://github.com/google/leveldb.git
$ cd leveldb/

$ mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j
```

```
$ ./db_bench
LevelDB: version 1.23
Date: Tue Jan 17 11:28:29 2023
CPU: 12 * Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz
CPUCache: 8448 KB
Keys: 16 bytes each
Values: 100 bytes each (50 bytes after compression)
Entries: 1000000
RawSize: 110.6 MB (estimated)
FileSize: 62.9 MB (estimated)
WARNING: Snappy compression is not enabled
------------------------------------------------
fillseq   : 2.239 micros/op; 49.4 MB/s
fillsync  : 341.212 micros/op; 0.3 MB/s (1000 ops)
fillrandom: 2.810 micros/op; 39.4 MB/s
overwrite : 3.620 micros/op; 30.6 MB/s
readrandom: 2.776 micros/op; (864322 of 1000000 found)
readrandom: 2.262 micros/op; (864083 of 1000000 found)
readseq   : 0.126 micros/op; 878.2 MB/s
readreverse: 0.238 micros/op; 464.9 MB/s
compact   : 628564.000 micros/op;
readrandom: 1.521 micros/op; (864105 of 1000000 found)
readseq   : 0.109 micros/op; 1012.7 MB/s
readreverse: 0.214 micros/op; 518.0 MB/s
fill100K  : 813.094 micros/op; 117.3 MB/s (1000 ops)
crc32c    : 1.273 micros/op; 3069.1 MB/s (4K per op)
snappycomp: 5801.000 micros/op; (snappy failure)
snappyuncomp: 1873.000 micros/op; (snappy failure)
```

```
$ ./db_bench
LevelDB: version 1.23
Date: Tue Jan 17 11:28:29 2023
CPU: 12 * Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz
CPUCache: 8448 KB
Keys: 16 bytes each
Values: 100 bytes each (50 bytes after compression)
Entries: 1000000
RawSize: 110.6 MB (estimated)
FileSize: 62.9 MB (estimated)
WARNING: Snappy compression is not enabled
------------------------------------------------
fillseq   : 2.239 micros/op; 49.4 MB/s
fillsync  : 341.212 micros/op; 0.3 MB/s (1000 ops)
fillrandom: 2.810 micros/op; 39.4 MB/s
overwrite : 3.620 micros/op; 30.6 MB/s
readrandom: 2.776 micros/op; (864322 of 1000000 found)
readrandom: 2.262 micros/op; (864083 of 1000000 found)
readseq   : 0.126 micros/op; 878.2 MB/s
readreverse: 0.238 micros/op; 464.9 MB/s
compact   : 628564.000 micros/op;
readrandom: 1.521 micros/op; (864105 of 1000000 found)
readseq   : 0.109 micros/op; 1012.7 MB/s
readreverse: 0.214 micros/op; 518.0 MB/s
fill100K  : 813.094 micros/op; 117.3 MB/s (1000 ops)
crc32c    : 1.273 micros/op; 3069.1 MB/s (4K per op)
snappycomp: 5801.000 micros/op; (snappy failure)
snappyuncomp: 1873.000 micros/op; (snappy failure)
```



```
#!/bin/sh
set -e

nb_threads="1 2 4 8"

for nb_thread in ${nb_threads}
do
    ./db_bench --benchmarks=readrandom --threads=${nb_thread}
done
```

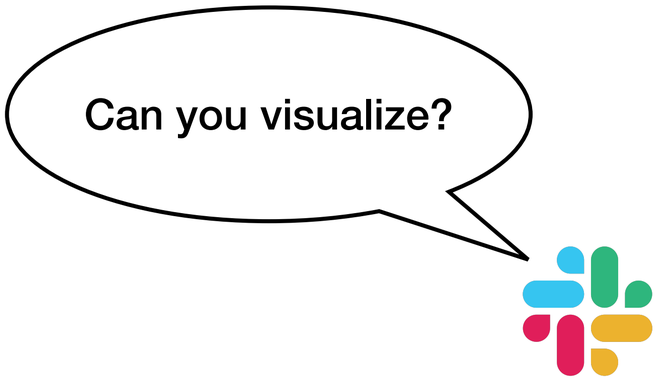
```
# how do you collect results in a meaningful way?

nb_threads="1 2 4 8"
bench_name=readrandom

echo "bench_name;nb_thread;microsop" | tee /tmp/results.csv

for nb_thread in ${nb_threads}
do
    ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
        > /tmp/leveldb_results.txt
    results=$(cat /tmp/leveldb_results.txt \
        | tail -n 1 \
        | grep -o "[0-9.]\+ micros/op" \
        | cut -d ' ' -f 1)
    echo "${bench_name};${nb_thread};${results}" | tee -a /tmp/results.csv
done
```

```
# how do you collect results in a meaningful way?

nb_threads="1 2 4 8"
bench_name=readrandom

echo "bench_name;nb_thread;microsop" | tee /tmp/results.csv

for nb_thread in ${nb_threads}
do
    ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
        > /tmp/leveldb_results.txt
    results=$(cat /tmp/leveldb_results.txt \
        | tail -n 1 \
        | grep -o "[0-9.]\+ micros/op" \
        | cut -d ' ' -f 1)
    echo "${bench_name};${nb_thread};${results}" | tee -a /tmp/results.csv
done
```

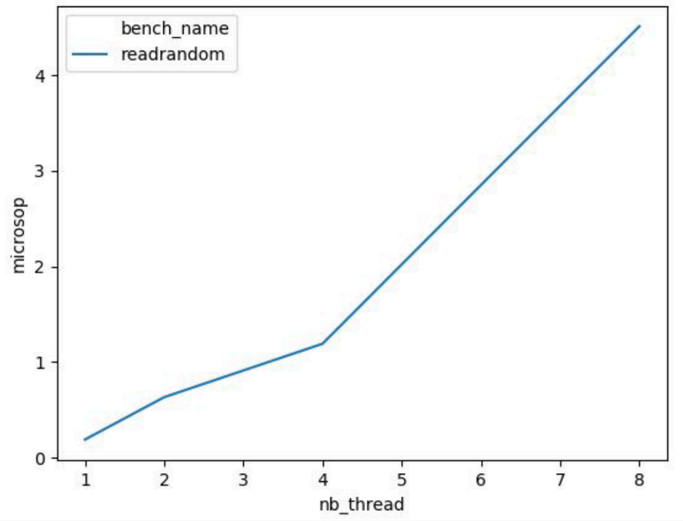
```
$ cat /tmp/results.csv
bench_name;nb_thread;microsop
readrandom;1;0.189
readrandom;2;0.632
readrandom;4;1.191
readrandom;8;4.510
```

```
# how do you collect results in a meaningful way?

nb_threads="1 2 4 8"
bench_name=readrandom

echo "bench_name;nb_thread;microsop" | tee /tmp/results.csv

for nb_thread in ${nb_threads}
do
    ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
        > /tmp/leveldb_results.txt
    results=$(cat /tmp/leveldb_results.txt \
        | tail -n 1 \
        | grep -o "[0-9.]\+ micros/op" \
        | cut -d ' ' -f 1)
    echo "${bench_name};${nb_thread};${results}" | tee -a /tmp/results.csv
done
```

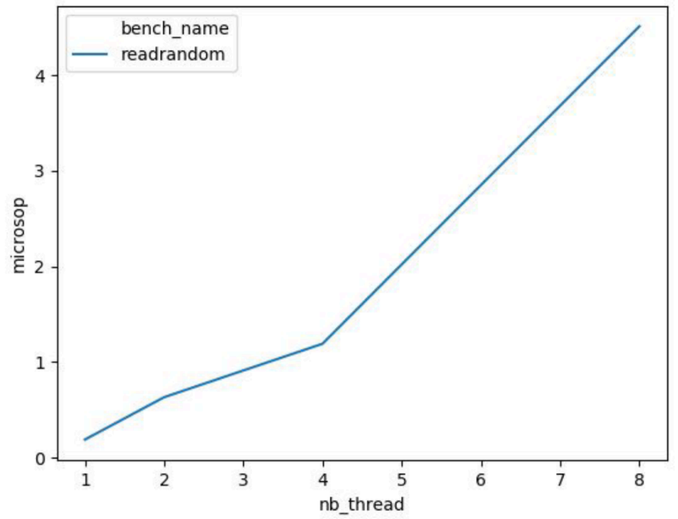
```
$ cat /tmp/results.csv
bench_name;nb_thread;microsop
readrandom;1;0.189
readrandom;2;0.632
readrandom;4;1.191
readrandom;8;4.510
```



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plot

df = pd.read_csv('/tmp/results.csv',
sep=';')
ax = sns.lineplot(
    data=df,
    x='nb_thread',
    y='microsop',
    hue='bench_name'
)
```



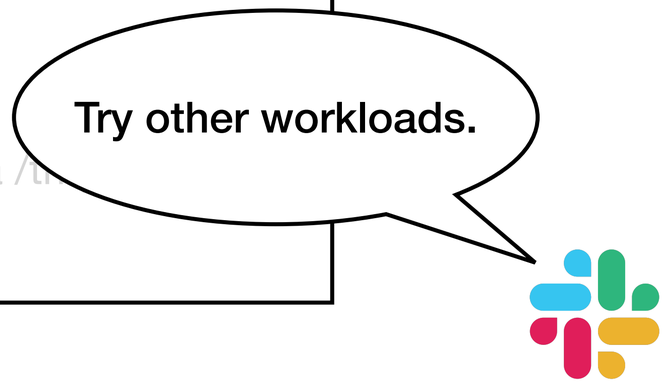
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plot

df = pd.read_csv('/tmp/results.csv',
sep=';')
ax = sns.lineplot(
    data=df,
    x='nb_thread',
    y='microsop',
    hue='bench_name'
)
```



Compare with different schedulers.

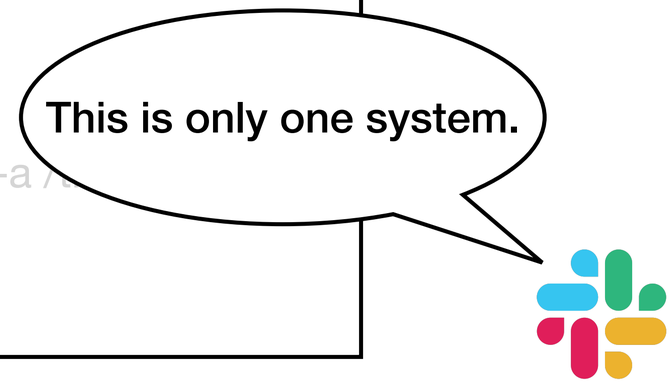
```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_name=readrandom

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for scheduler in ${schedulers}
do
    for nb_thread in ${nb_threads}
    do
        ./schedkit --policy=${scheduler}
        ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
            > /tmp/leveldb_results.txt
        ./schedkit --cleanup

        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
    done
done
```

```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_name=readrandom

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for scheduler in ${schedulers}
do
    for nb_thread in ${nb_threads}
    do
        ./schedkit --policy=${scheduler}
        ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
            > /tmp/leveldb_results.txt
        ./schedkit --cleanup

        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /t
    done
done
```



```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
   for scheduler in ${schedulers}
   do
      for nb_thread in ${nb_threads}
      do
         ./schedkit --policy=${scheduler}
         ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
            > /tmp/leveldb_results.txt
         ./schedkit --cleanup

         results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

         echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
      done
   done
done
```

```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
    for scheduler in ${schedulers}
    do
        for nb_thread in ${nb_threads}
        do
            ./schedkit --policy=${scheduler}
            ./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
                > /tmp/leveldb_results.txt
            ./schedkit --cleanup

            results=$(cat /tmp/leveldb_results.txt \
                | tail -n 1 \
                | grep -o "[0-9.]\+ micros/op" \
                | cut -d ' ' -f 1)

            echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /t
        done
    done
done
```




```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
   for scheduler in ${schedulers}
   do
      for nb_thread in ${nb_threads}
      do
         ./schedkit --policy=${scheduler}
         ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
         ./schedkit --cleanup

         results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

         echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
      done
   done
done
```


```
#RocksDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/facebook/rocksdb.git
cd rocksdb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
   for scheduler in ${schedulers}
   do
      for nb_thread in ${nb_threads}
      do
         ./schedkit --policy=${scheduler}
         ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
         ./schedkit --cleanup

         results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

         echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
      done
   done
done
```


```
# Kyoto Cabinet

#!/bin/sh
set -e

curl -L -o /tmp/kyotocabinet-1.2.76.tar.gz https://dbmx.net/kyotocabinet/pkg/kyotocabinet-1.2.76.tar.gz
cd /tmp

tar xf kyotocabinet-1.2.76.tar.gz
cd kyotocabinet-1.2.76

./configure
make -j"$(nproc)"
make benchmark

duration_s=10
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"

echo "bench_name;nb_thread;duration;global_count;operations_per_second" | tee /tmp/results.csv

for scheduler in ${schedulers}
do
   for nb_thread in ${nb_threads}
   do
      ./schedkit --policy=${scheduler}
      ./benchmark -t "${nb_thread}" -d "${duration_s}" > /tmp/kyotocabinet_results.txt
      ./schedkit --cleanup

      total_ops=$(grep "Summary: total_ops=" /tmp/kyotocabinet_results.txt \
         | sed 's/Summary: total_ops=//')

      ops_per_second=$(awk -v ops="$total_ops" -v duration="$duration_s" \
         'BEGIN { print ops / duration }')

      echo "${bench_name};${nb_thread};${duration_s};${total_ops};${ops_per_second}" \
         | tee -a /tmp/results.csv
   done
done
```


```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
   for scheduler in ${schedulers}
   do
      for nb_thread in ${nb_threads}
      do
         ./schedkit —policy=${scheduler}
         ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
         ./schedkit --cleanup

         results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

         echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
      done
   done
done
```


```
# Kyoto Cabinet

#!/bin/sh
set -e

curl -L -o /tmp/kyotocabinet-1.2.76.tar.gz https://dbmx.net/kyotocabinet/pkg/
kyotocabinet-1.2.76.tar.gz
cd /tmp

tar xf kyotocabinet-1.2.76.tar.gz
cd kyotocabinet-1.2.76

./configure
make -j"$(nproc)"
make benchmark

duration_s=10
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"

echo "bench_name;nb_thread;duration;global_count;operations_per_second" | tee /tmp/results.csv

for scheduler in ${schedulers}
do
   for nb_thread in ${nb_threads}
   do
      ./schedkit --policy=${scheduler}
      ./benchmark -t "${nb_thread}" -d "${duration_s}" > /tmp/kyotocabinet_results.txt
      ./schedkit --cleanup

      total_ops=$(grep "Summary: total_ops=" /tmp/kyotocabinet_results.txt \
         | sed 's/Summary: total_ops=//')

      ops_per_second=$(awk -v ops="$total_ops" -v duration="$duration_s" \
         'BEGIN { print ops / duration }')

      echo "${bench_name};${nb_thread};${duration_s};${total_ops};${ops_per_second}" \
         | tee -a /tmp/results.csv
   done
done
```

RQ: Can we make benchmark definitions reusable?

Benchmark

How to explore


```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for bench_name in ${bench_names}
do
   for scheduler in ${schedulers}
   do
      for nb_thread in ${nb_threads}
      do
         ./schedkit --policy=${scheduler}
         ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
         ./schedkit --cleanup

         results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

         echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
      done
   done
done)
```

Benchmark

How to explore

(1) Fetch

```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

echo "bench_name;nb_thread;scheduler;microsop" | tee /tmp/results.csv

for scheduler in ${schedulers}
do
    for nb_thread in ${nb_threads}
    do
        ./schedkit --policy=${scheduler}
        ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
        ./schedkit --cleanup

        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
    done
  done
done
```

git clone --recurse-submodules https://github.com/google/leveldb.git

Benchmark

How to explore

(2) Build

```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"
```


```
mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j
```


```
    for nb_thread in ${nb_threads}
    do
        ./schedkit --policy=${scheduler}
        ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt
        ./schedkit --cleanup

        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
    done
  done
done
```

Benchmark

How to explore


```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/

mkdir -p build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build -- -j

nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"
```

```
./db_bench --benchmarks=${bench_name} --threads=${nb_thread}
    > /tmp/leveldb_results.txt
```

(3) Run

```
    for nb_thread in ${nb_threads}
    do
        ./db_bench --benchmarks=${bench_name} --threads=${nb_thread} > /tmp/leveldb_results.txt

        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
    done
  done
done
```

Benchmark

How to explore


```
#LevelDB

#!/bin/sh
set -e

git clone --recurse-submodules https://github.com/google/leveldb.git
cd leveldb/
```

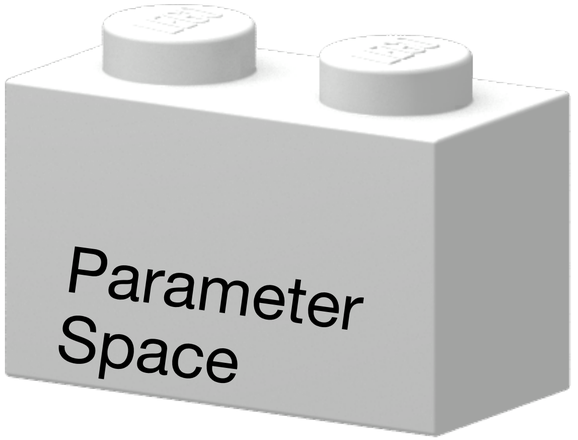
```
        results=$(cat /tmp/leveldb_results.txt \
            | tail -n 1 \
            | grep -o "[0-9.]\+ micros/op" \
            | cut -d ' ' -f 1)

        echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
```

(4) Collect

```
            ./schedkit --cleanup

            results=$(cat /tmp/leveldb_results.txt \
                | tail -n 1 \
                | grep -o "[0-9.]\+ micros/op" \
                | cut -d ' ' -f 1)

            echo "${bench_name};${nb_thread};${scheduler};${results}" | tee -a /tmp/results.csv
        done
    done
done
```

Parameter Space

What to explore: the cartesian product



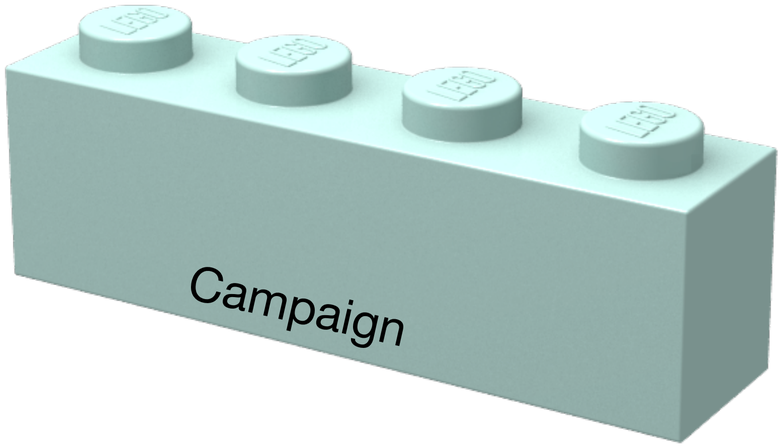

```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

for bench_name in ${bench_names}
do
    for scheduler in ${schedulers}
    do
        for nb_thread in ${nb_threads}
        do
            ./bench
        done
    done
done
```

nb_threads × scheduler × bench_name

readrandom;Normal;1
readrandom;Normal;2
readrandom;Normal;4
readrandom;Normal;8
readrandom;SAS;1
readrandom;SAS;2
readrandom;SAS;4
readrandom;SAS;8
readrandom;SAM;1
readrandom;SAM;2
readrandom;SAM;4
readrandom;SAM;8
seekrandom;Normal;1
seekrandom;Normal;2

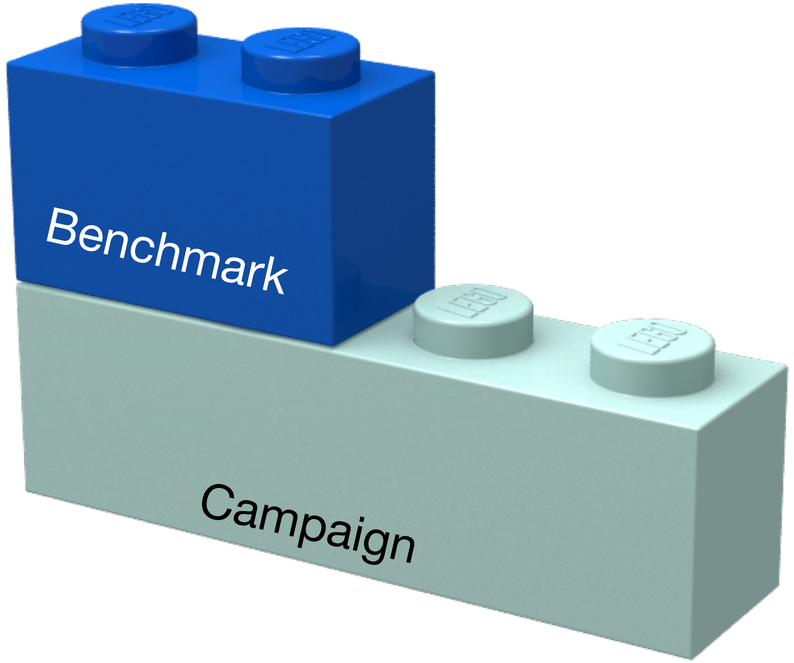
Campaign

Orchestrating the exploration



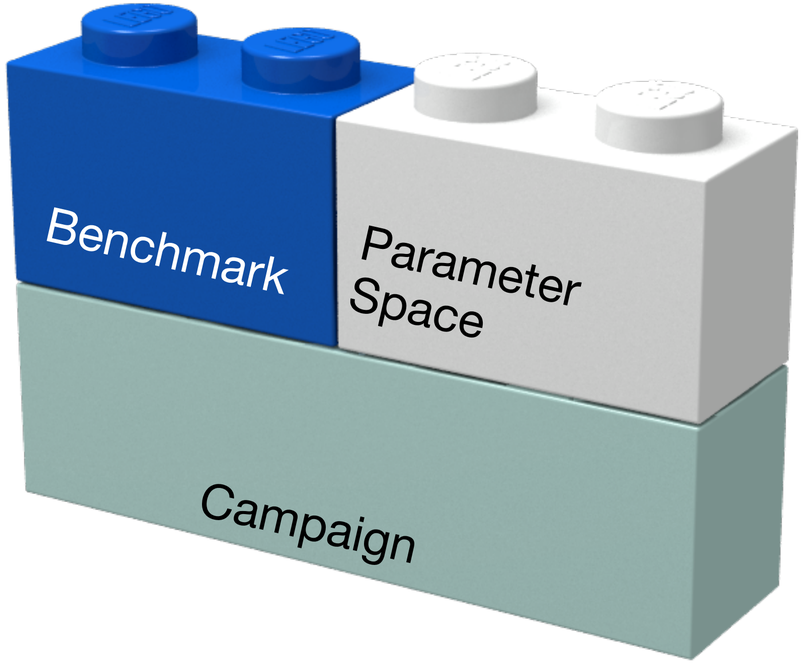
Campaign

Orchestrating the exploration



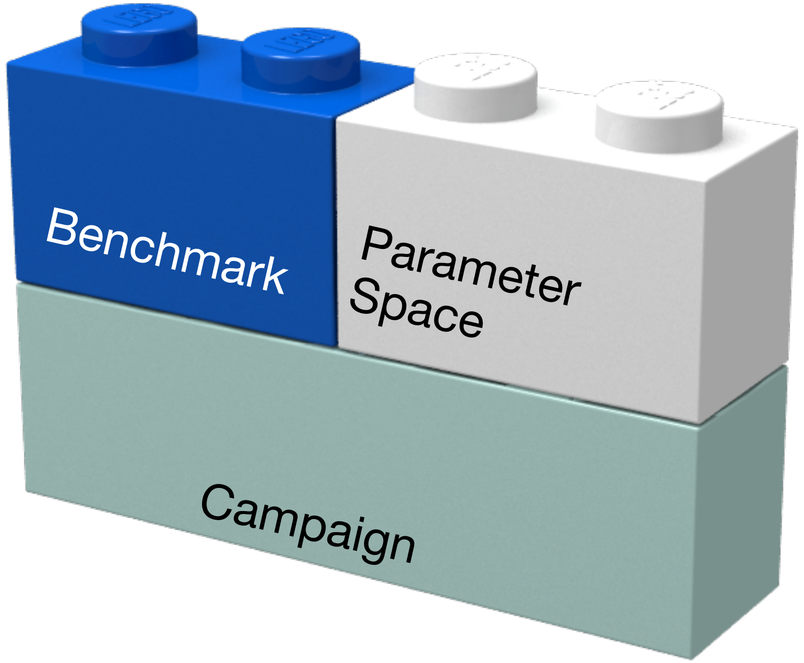
Campaign

Orchestrating the exploration

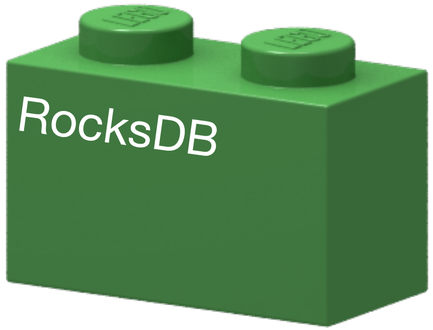


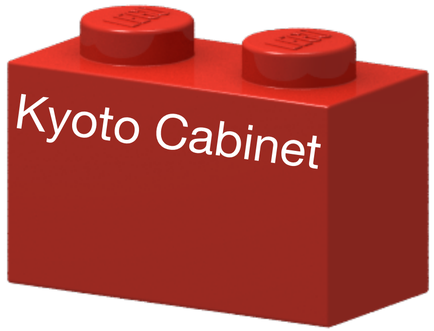
Campaign

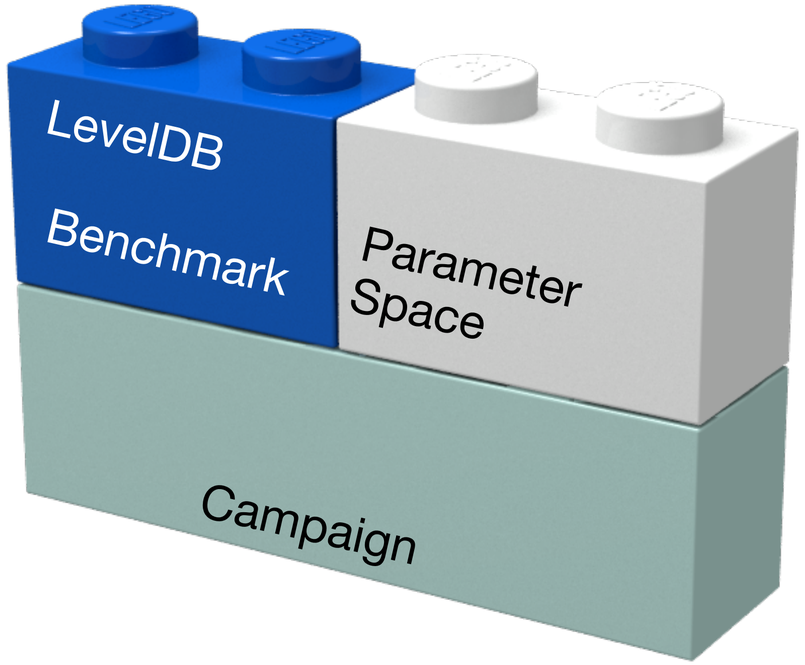
Orchestrating the exploration

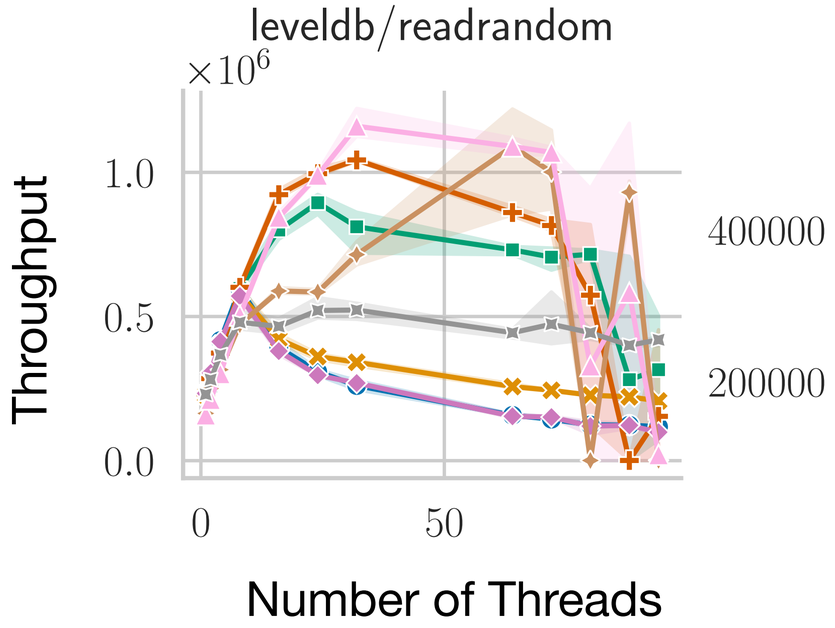


This is only one system.


RocksDB

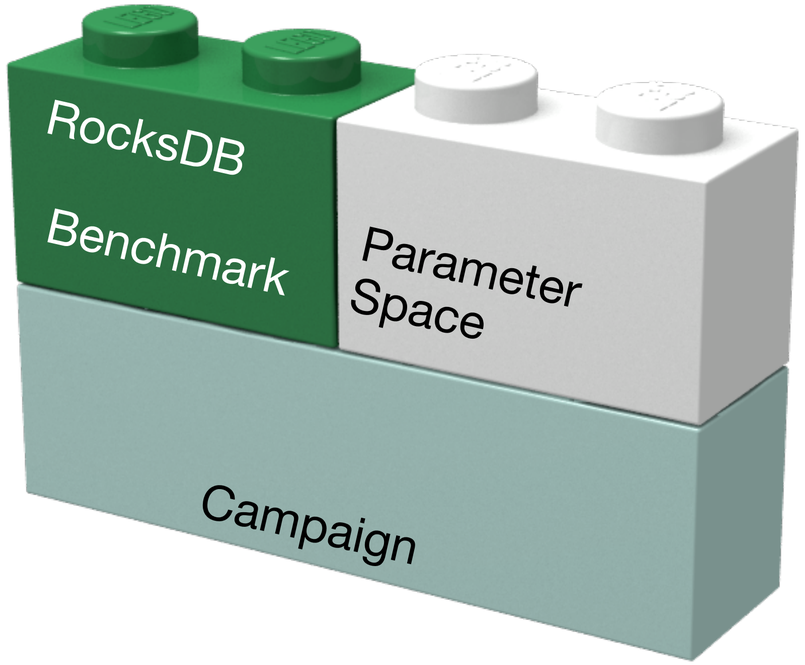

Kyoto Cabinet

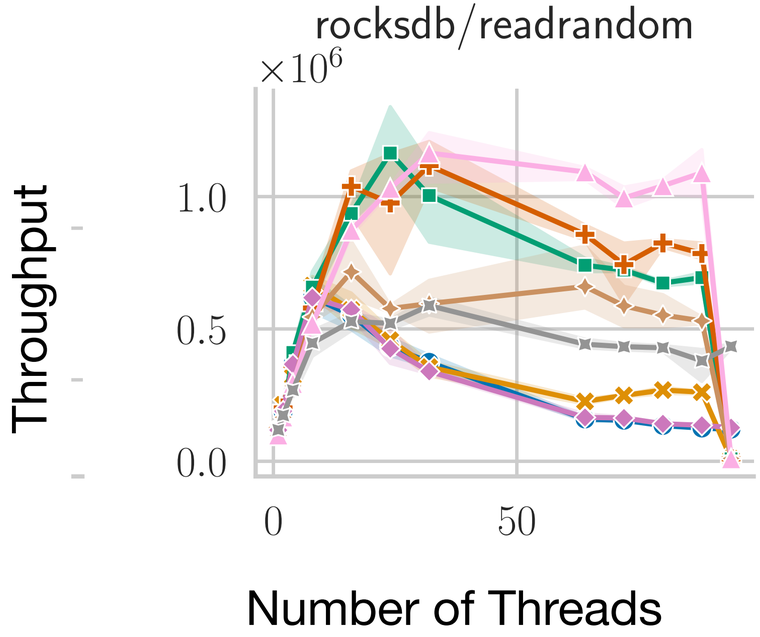

LevelDB
Benchmark
Parameter
Space
Campaign


leveldb/readrandom
×10^6
1.0
0.5
0.0
Throughput
0
50
Number of Threads
400000
200000

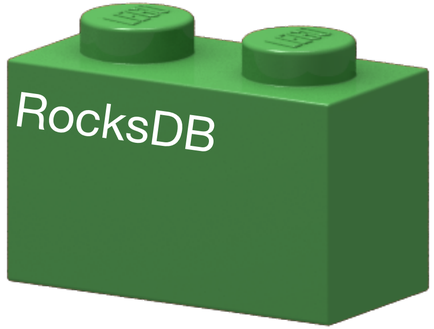
Kunpeng

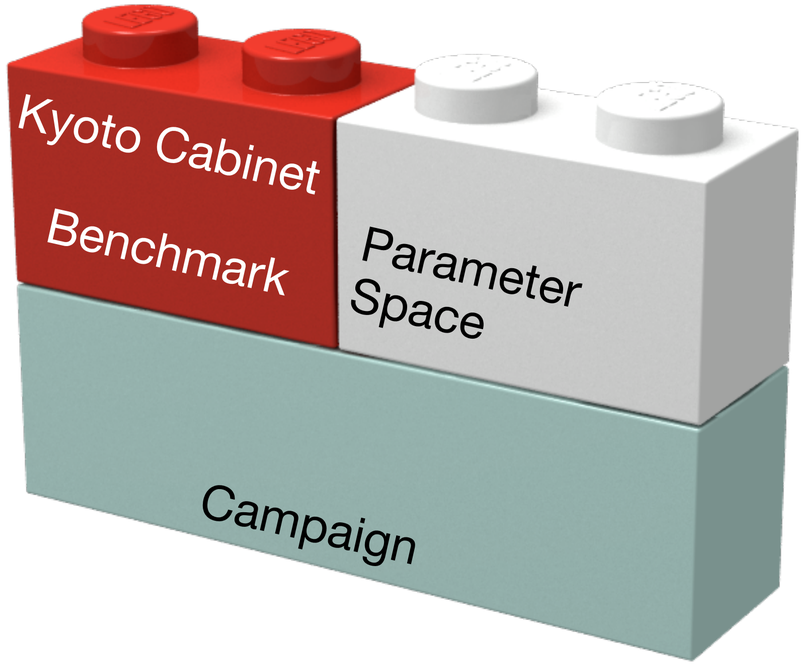
RocksDB

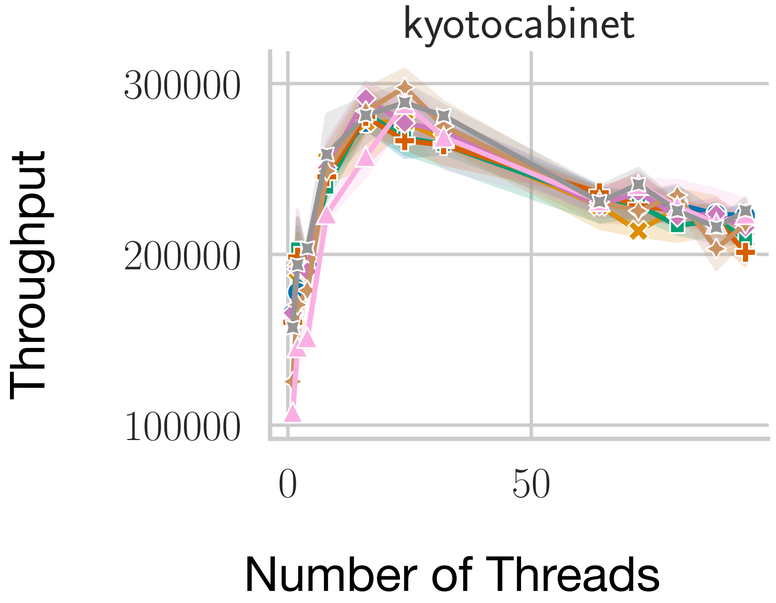





LevelDB


RocksDB

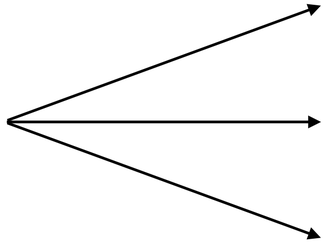

Kyoto Cabinet
Benchmark
Parameter
Space
Campaign


kyotocabinet
Throughput
300000
200000
100000
0
50
Number of Threads


Kunpeng
京都
キャビネット
8 EiB

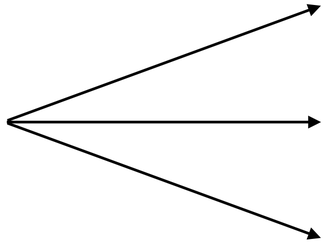
Performance evaluation requires further analysis...

Adjust the system behavior



- Set a scheduling policy
- Adjust the CPU frequency
- …

Collect extra information



- Get #cache-misses
- Get #cpu-migrations
- …

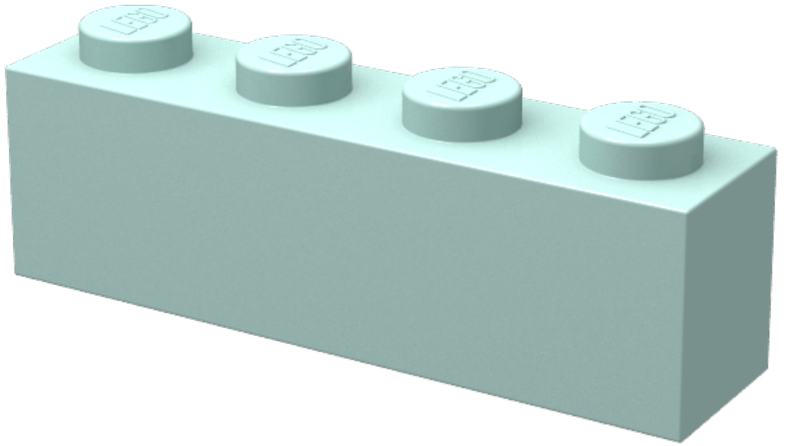
```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

for bench_name in ${bench_names}
do
    for scheduler in ${schedulers}
    do
        for nb_thread in ${nb_threads}
        do
            ./bench
        done
    done
done
```

```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

for bench_name in ${bench_names}
do
    for scheduler in ${schedulers}
    do
        for nb_thread in ${nb_threads}
        do
            ./bench
        done
    done
done
```

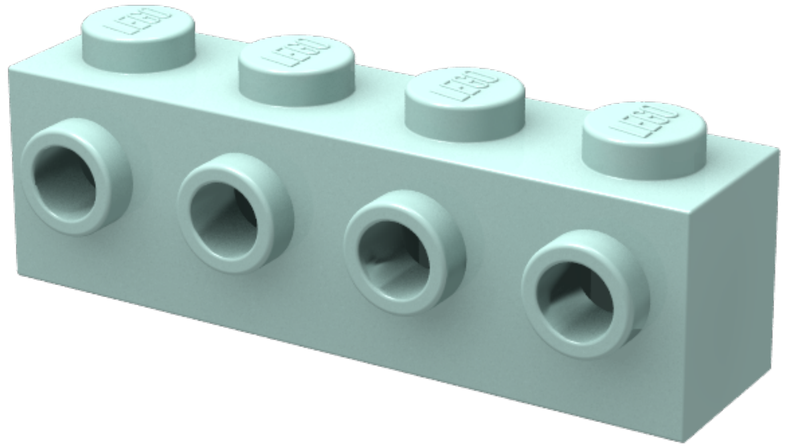
```
nb_threads="1 2 4 8"
schedulers="Normal SAS SAM"
bench_names="readrandom seekrandom"

for bench_name in ${bench_names}
do
    for scheduler in ${schedulers}
    do
        for nb_thread in ${nb_threads}
        do
            ./schedkit --policy=${scheduler}
            ./bench
            ./schedkit --cleanup
        done
    done
done
```

RQ: Can we compose experiments from reusable parts?

Composable Benchmarking

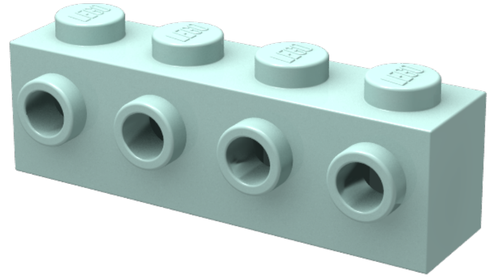


Composable Benchmarking



Composable Benchmarking

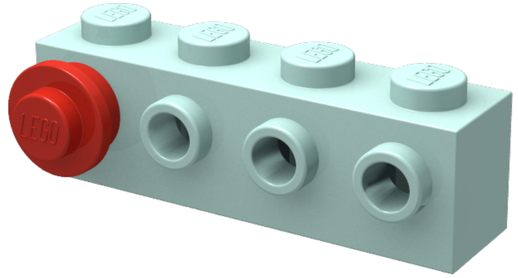
Command Wrappers



```
./bench
```

Composable Benchmarking

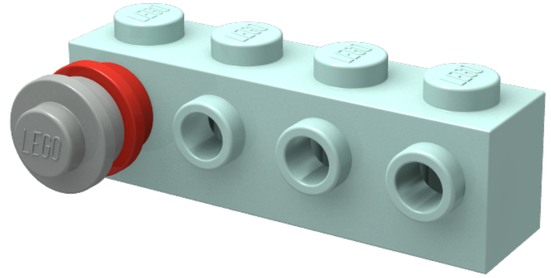
Command Wrappers



```
perf stat -e cache-misses ./bench
```

Composable Benchmarking

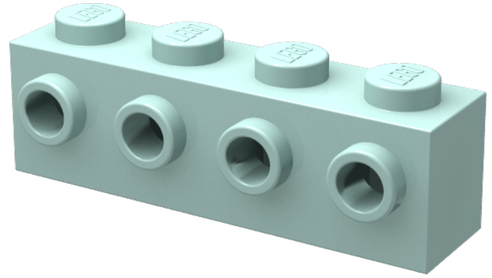
Command Wrappers



```
taskset -c 1 \
    perf stat -e cache-misses \
        ./bench
```

Composable Benchmarking

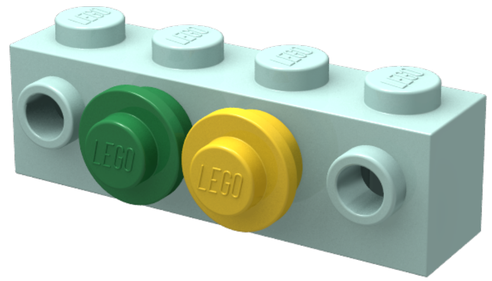
Pre and Post RunHooks



```
./bench
```

Composable Benchmarking

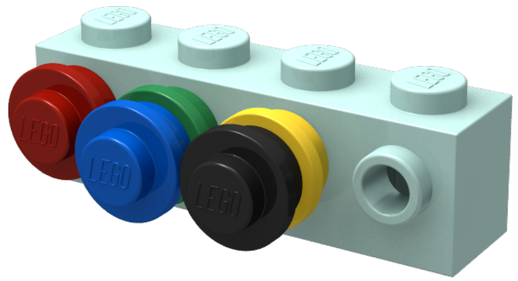
Pre and Post RunHooks



```
./schedkit --policy=FAR
./bench
./schedkit --cleanup
```

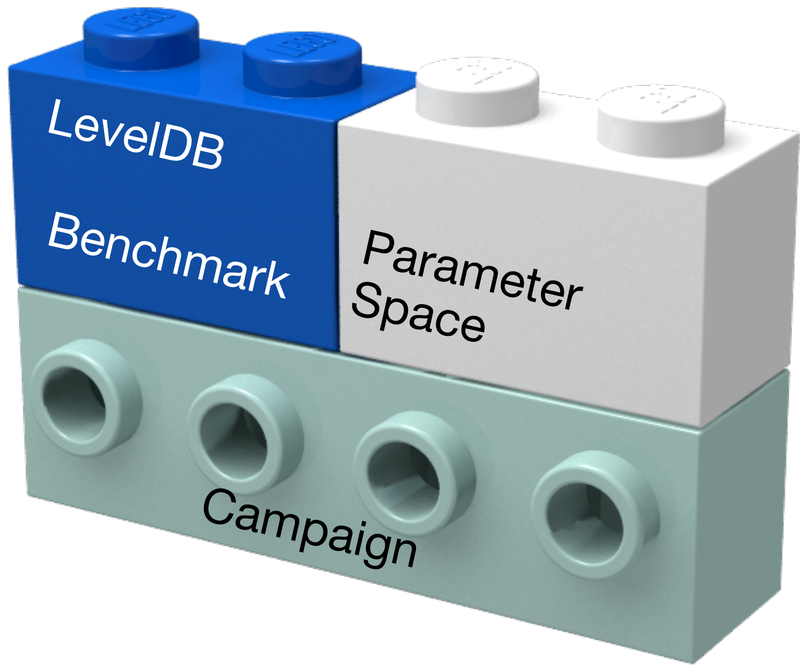
Composable Benchmarking

Combining Wrappers and Hooks



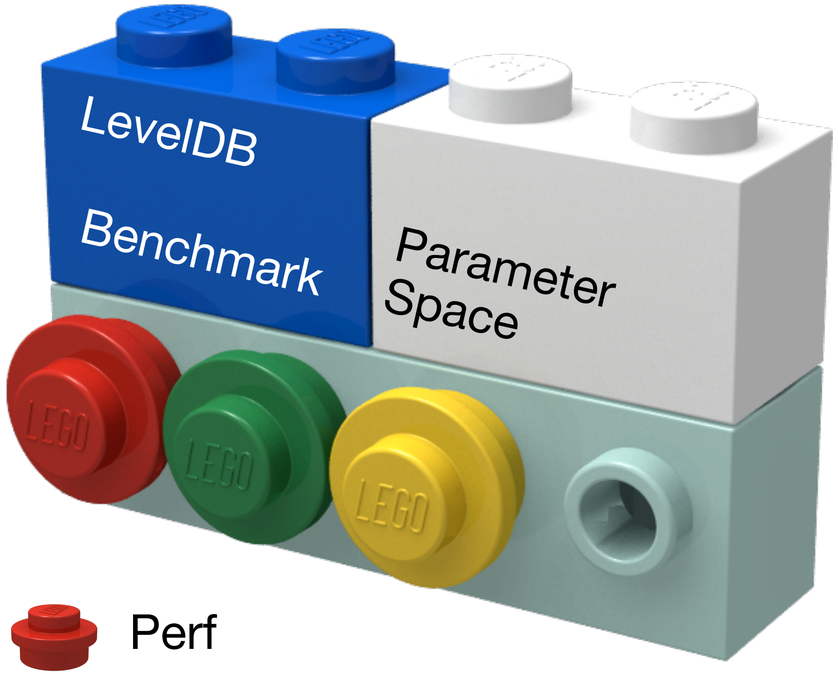
```
cpupower frequency-set -g performance
./schedkit --policy=FAR
perf stat -e cache-misses ./bench
./schedkit --cleanup
cpupower frequency-set -g normal
```

Building Different Experiments



Building Different Experiments

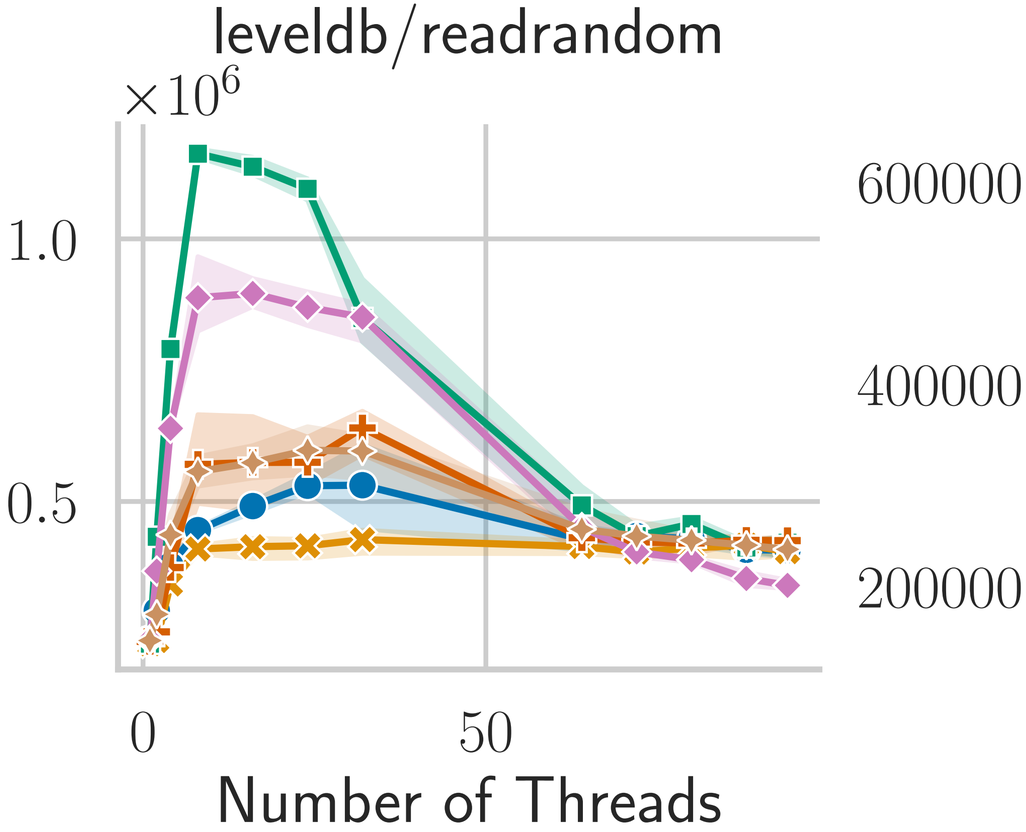
Replication Study of Numa-Aware Schedulers



Perf

schedkit Pre RunHook

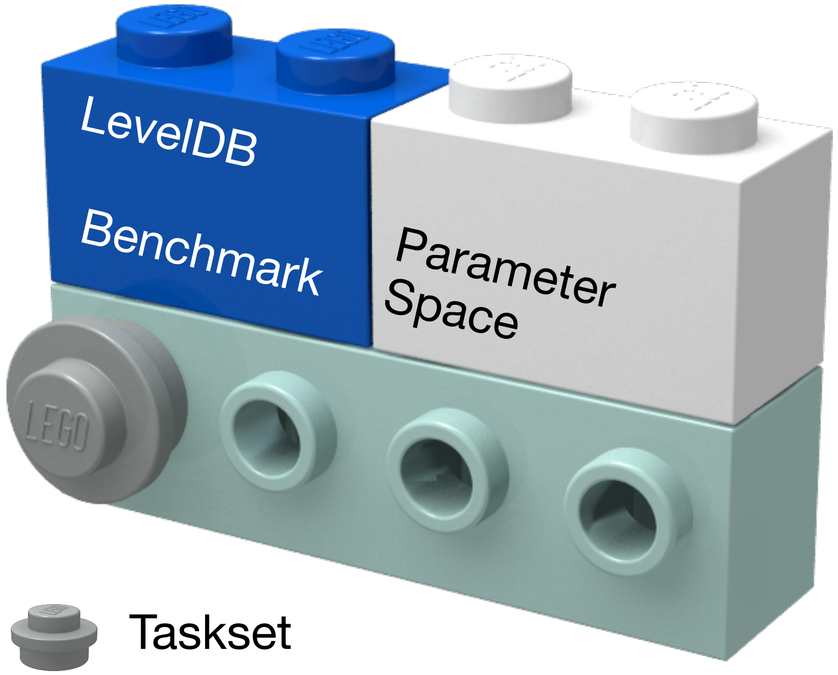
schedkit Post Run Hook



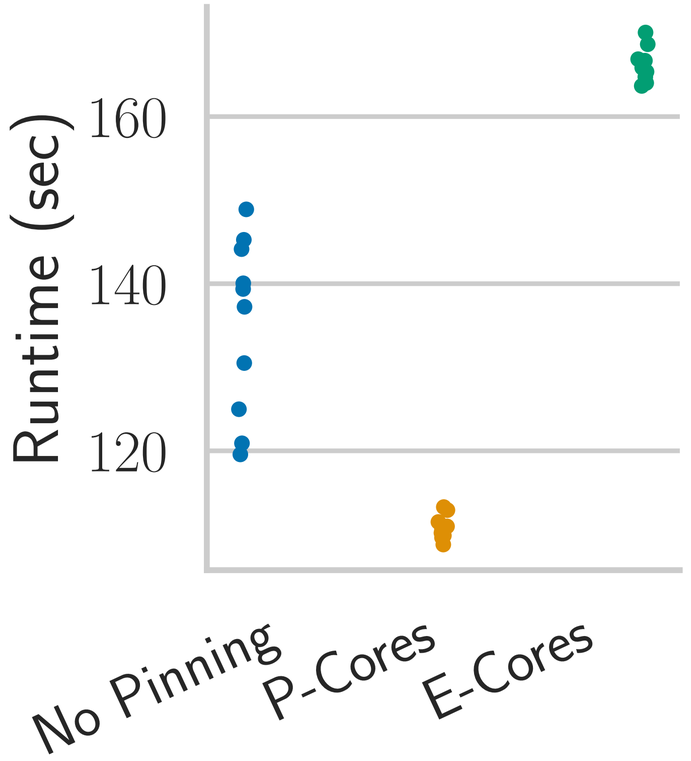
https://github.com/softwarelanguageslab/icpe26-benchkit-ae/blob/main/experiments/fig07_locks.py

Building Different Experiments

Drill Down of Single Core Performance



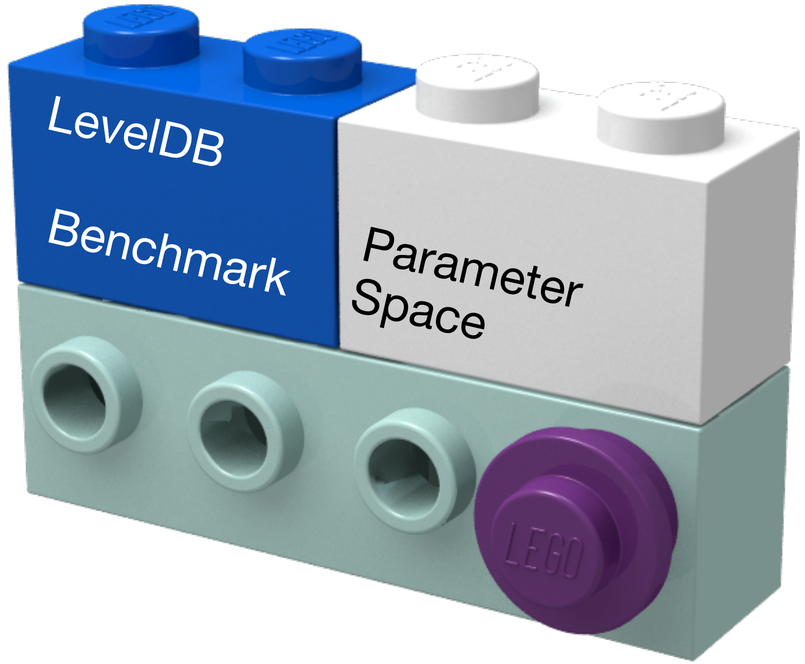
Taskset

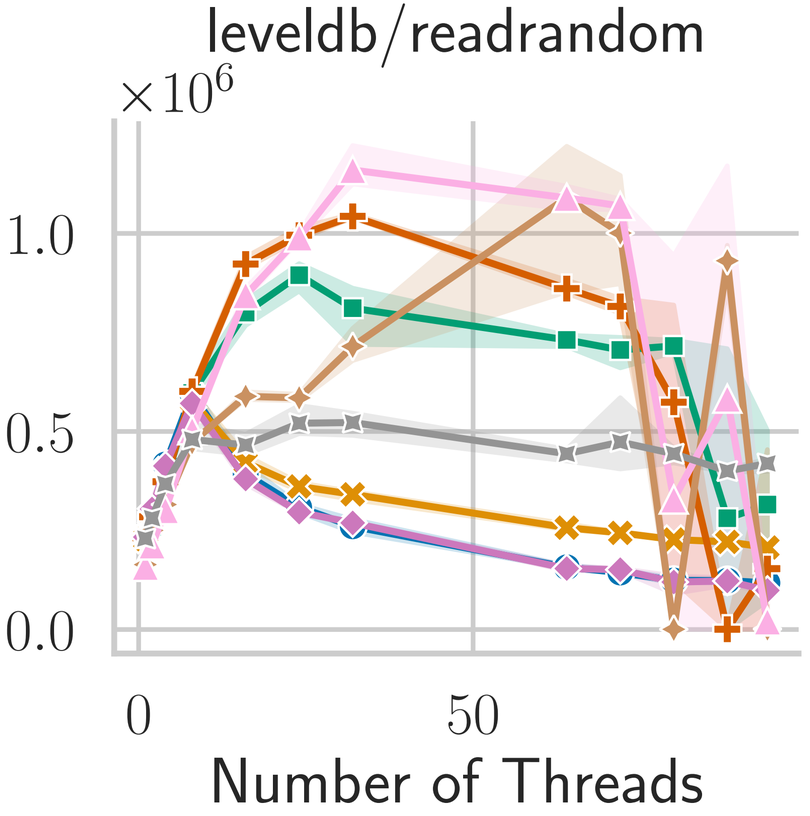


https://github.com/softwarelanguageslab/icpe26-benchkit-ae/blob/main/experiments/fig05_placement_leveldb.py

Building Different Experiments

Replication Study of Numa-Aware Locks



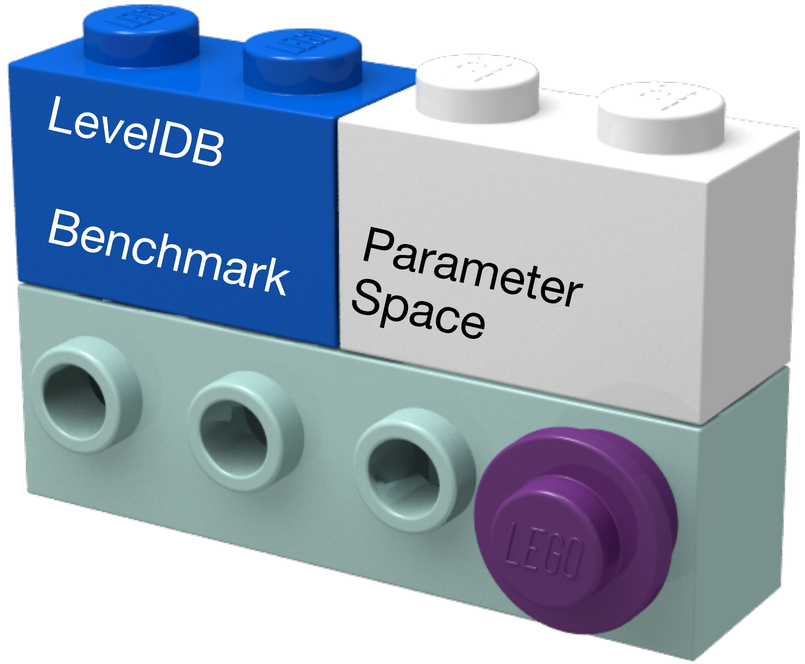


LD_PRELOAD: drop in replacement for pthread lock

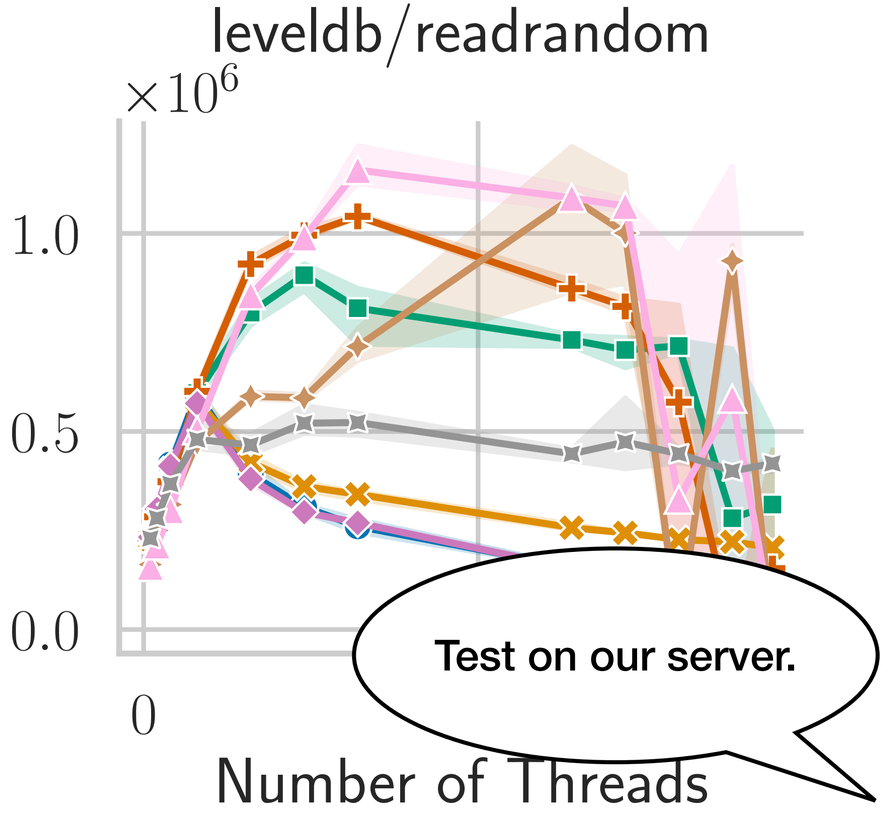
https://github.com/softwarelanguageslab/icpe26-benchkit-ae/blob/main/experiments/fig07_locks.py

Building Different Experiments

Replication Study of Numa-Aware Locks



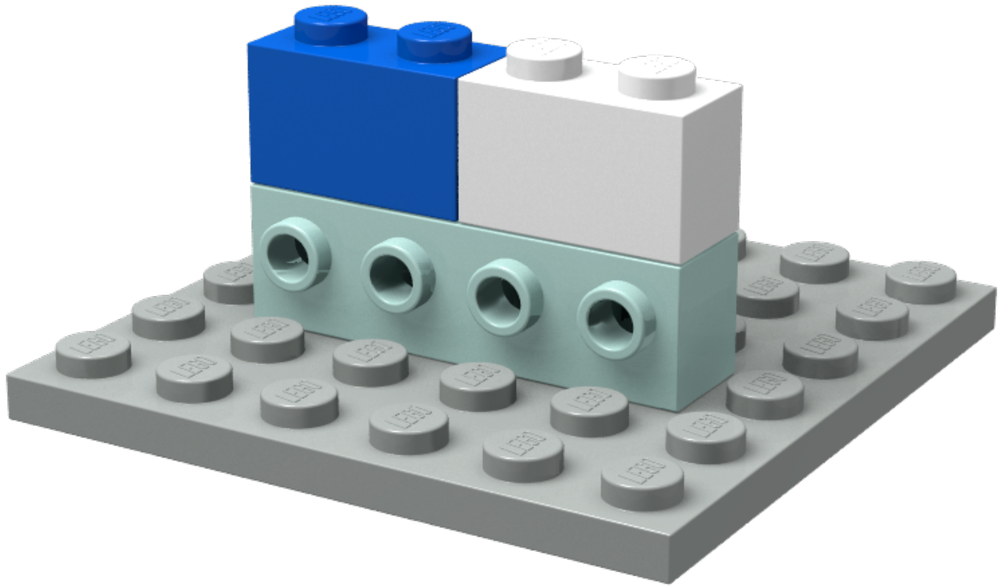
LD_PRELOAD: drop in replacement for pthread lock



https://github.com/softwarelanguageslab/icpe26-benchkit-ae/blob/main/experiments/fig07_locks.py

Platform

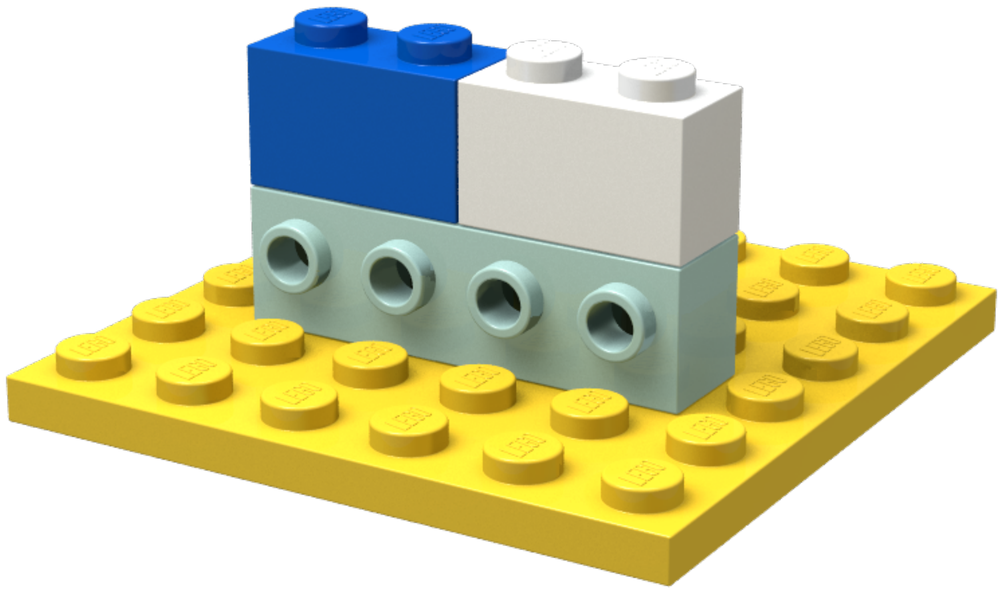
./bench



Local Machine

Platform

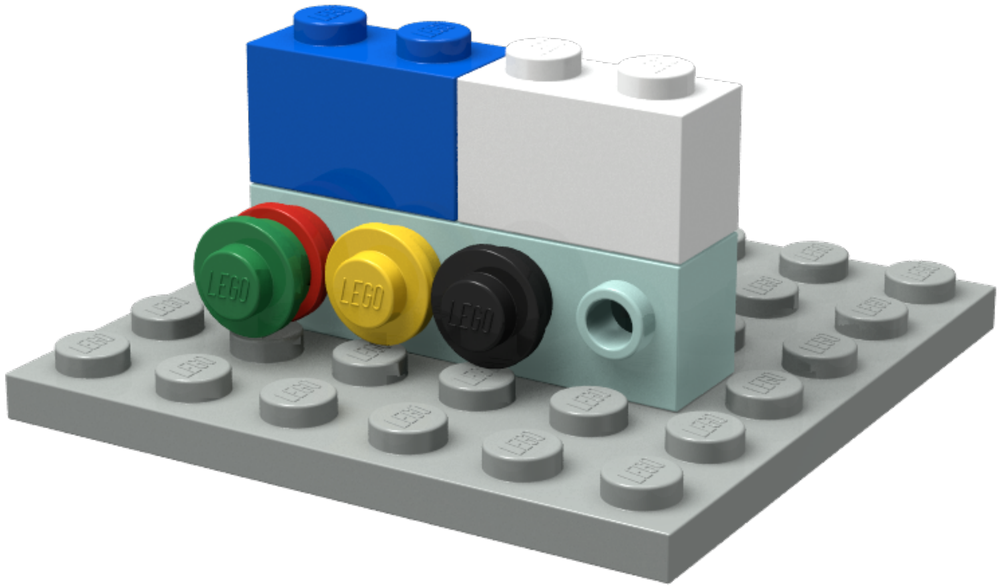
```
ssh remotehost ./bench
```

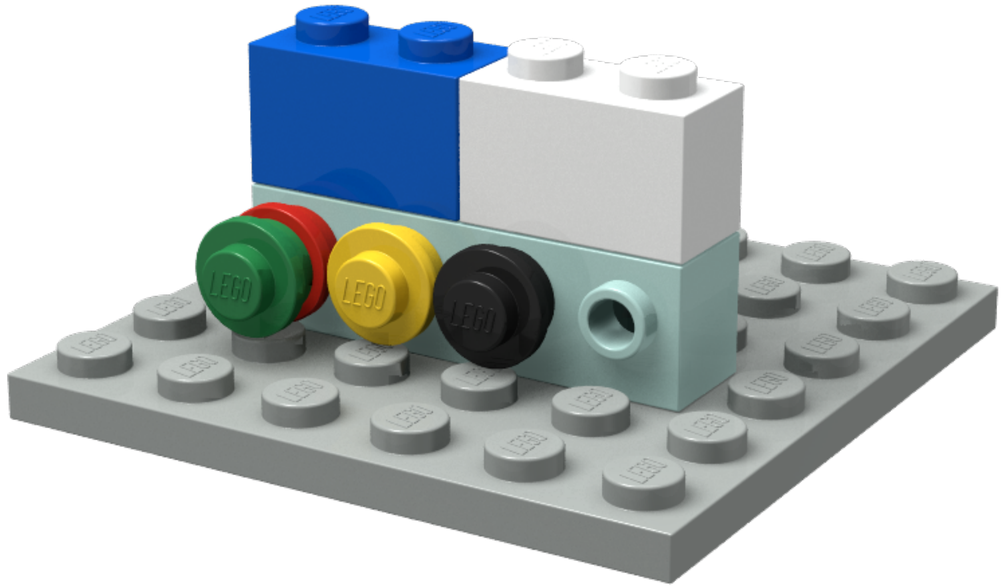


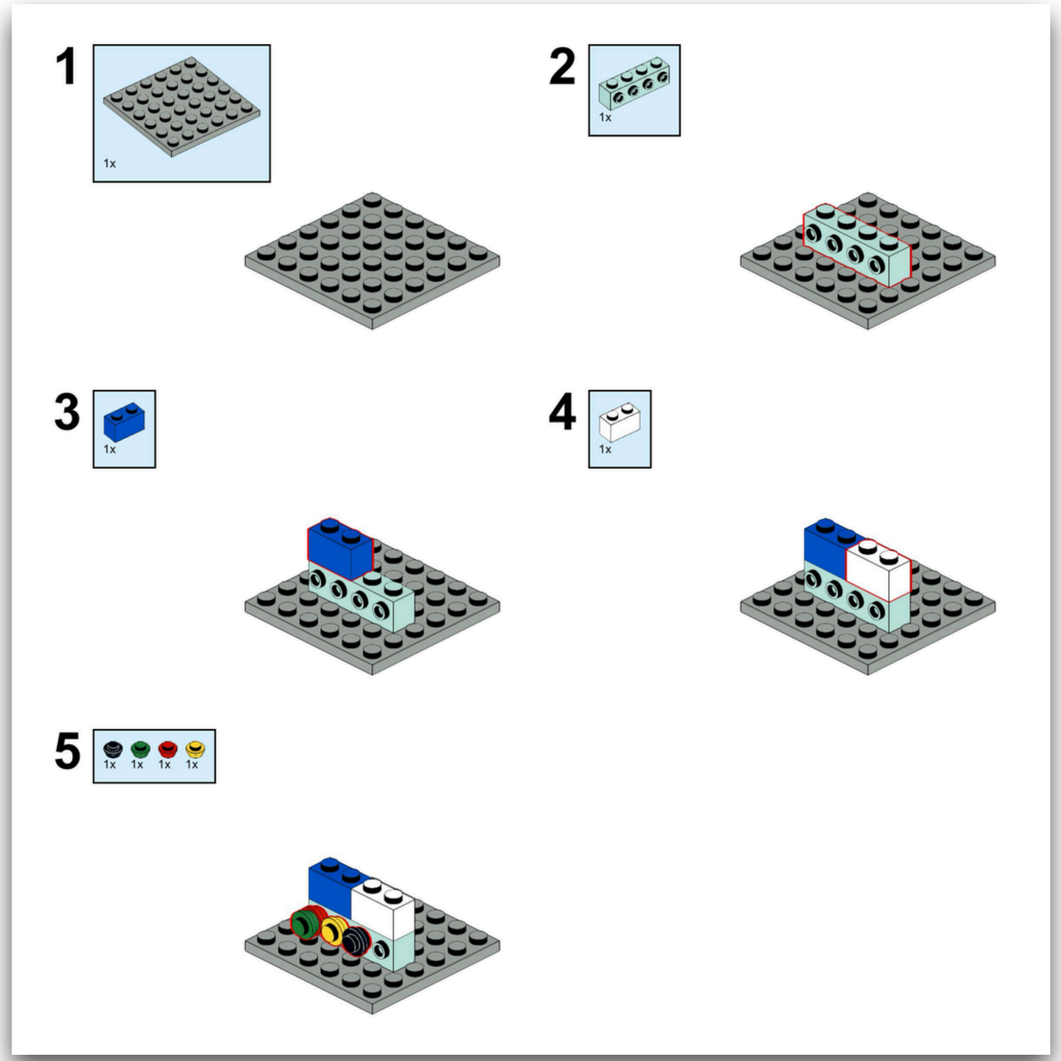
Local Machine

Remote Server (SSH)

RQ: Can we replicate experiments across platforms?

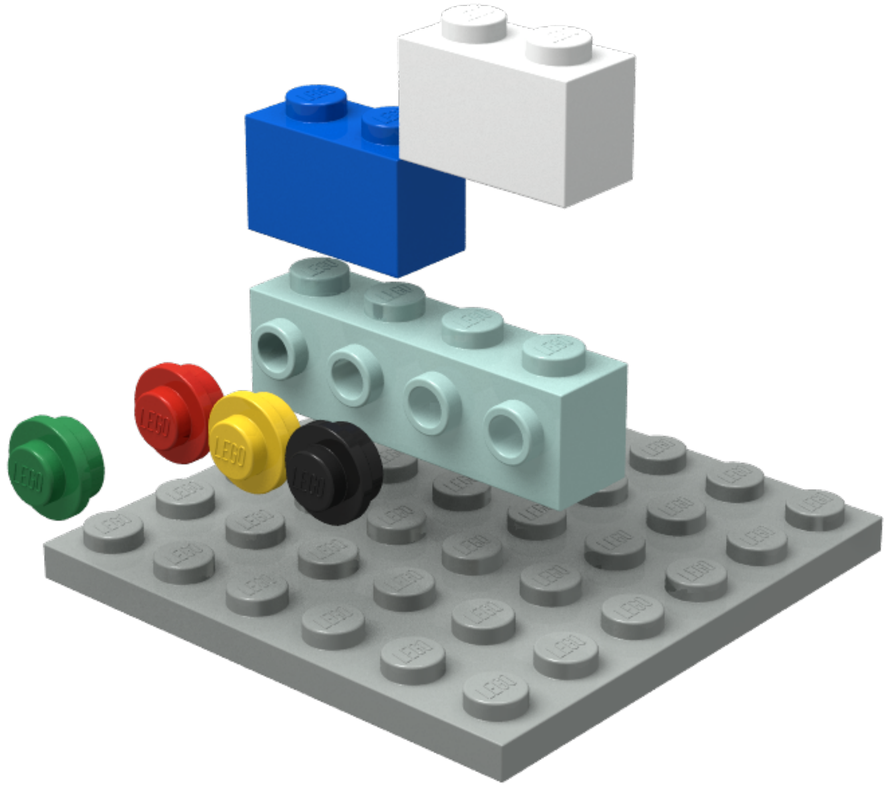





1
1x
2
1x
3
1x
4
1x
5
1x 1x 1x 1x

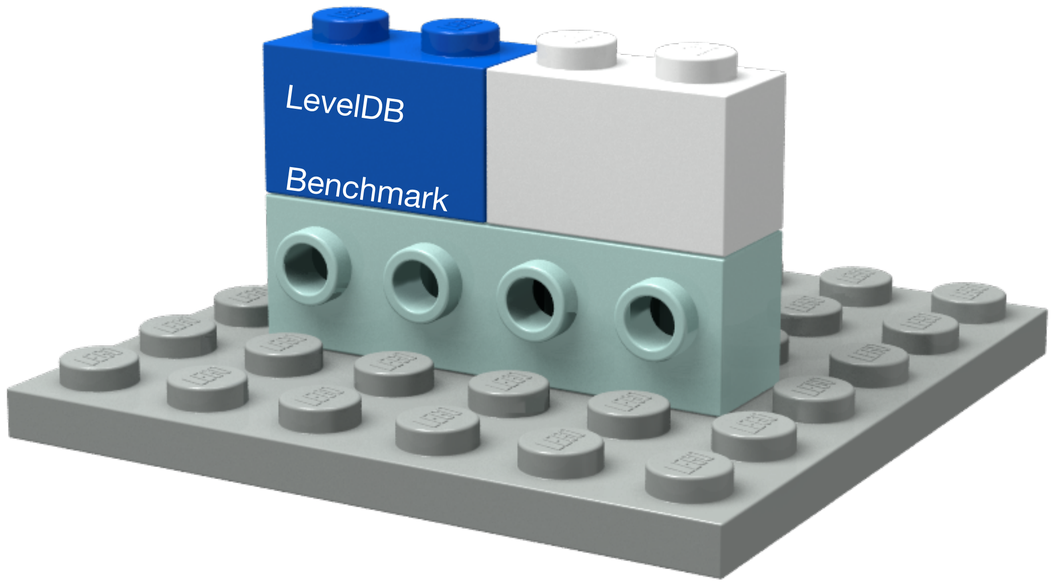
benchkit

- Python library
- Declarative definition of performance evaluation campaigns
- MIT License
- Composable & Extensible



benchkit

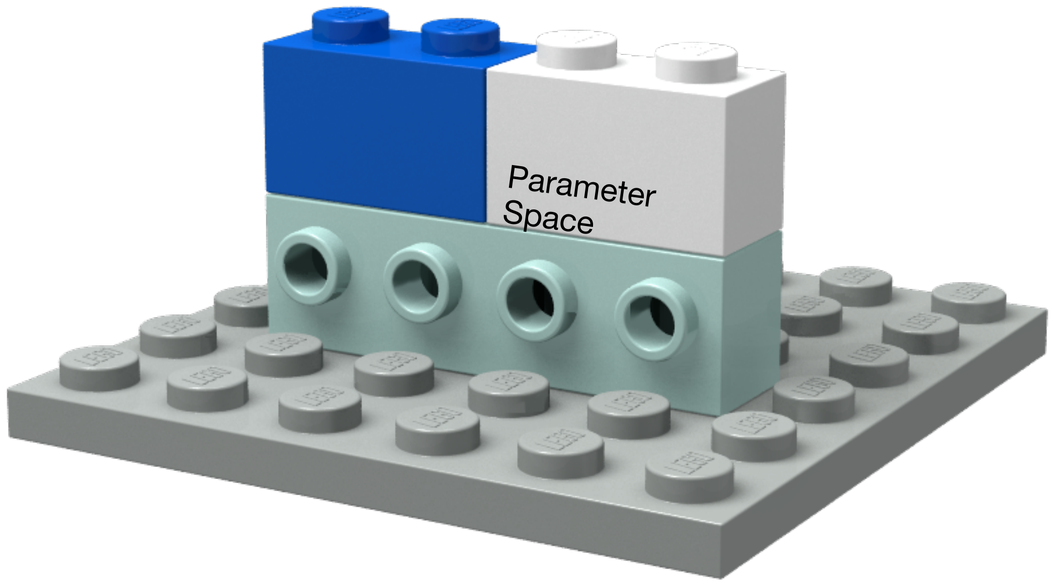
Benchmark class definition




```python
class LevelDBBench:

    def fetch(self, platform):
        url = "https://github.com/google/leveldb.git"
        platform.shell(f"git clone {url} src")

    def build(self, platform):
        b = "src/build"
        platform.makedir(b)
        platform.shell("cmake ..", dir=b)
        platform.shell("make", dir=b)

    def run(self, platform, bench_name, nb_threads):
        out = platform.run_shell(
            f"./db_bench"
            f" --benchmarks={bench_name} --threads={nb_threads}"
        )
        return out

    def collect(self, output):
        count = int(parse_count(output))
        dur   = float(parse_duration(output))
        return {"throughput": count / dur}
```

benchkit

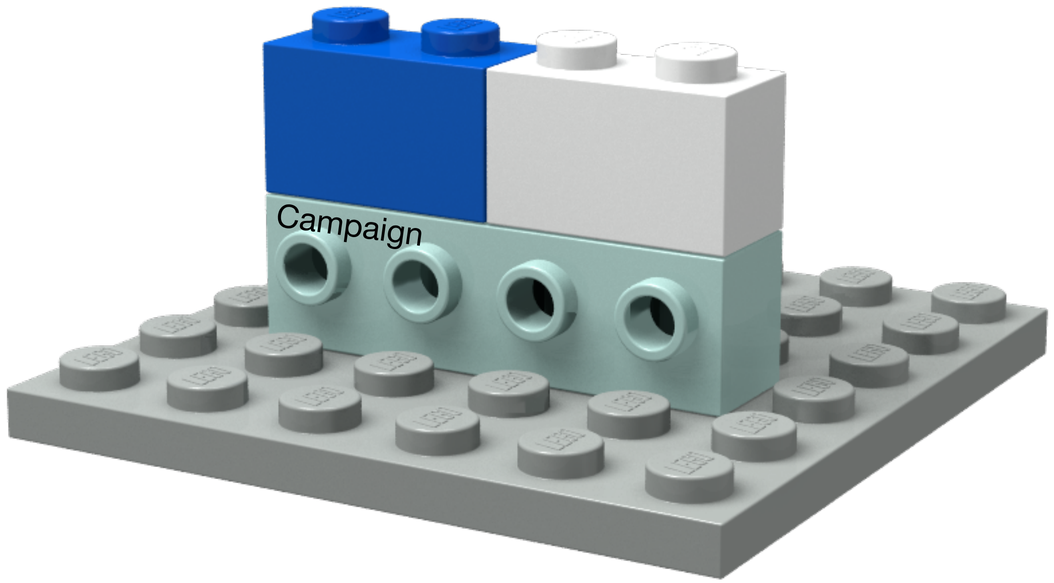
Parameter space definition




```
parameter_space = {
    "bench_name": ["readrandom", "seekrandom"],
    "nb_threads": [2, 4, 8],
    "scheduler":  ["Normal", "SAS", "SAM"]
}
```

benchkit

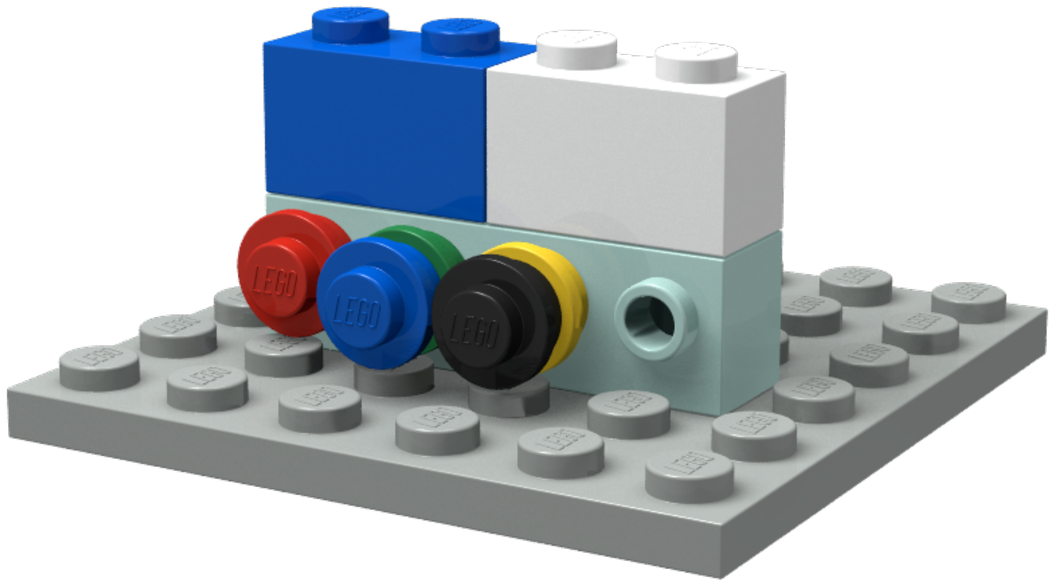
Campaign of experiments




```
campaign = CampaignCartesianProduct(
    benchmark=LevelDBBench(),
    variables=parameter_space,
)
campaign.run()

campaign.generate_graph(
    plot_name="lineplot",
    x="nb_threads",
    y="throughput",
    hue="bench_name",
)
```

benchkit

Campaign with wrappers & hooks

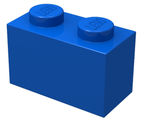


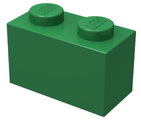
```
campaign = CampaignCartesianProduct(
    benchmark=LevelDBBench(),
    variables=parameter_space,
    pre_run_hooks=[cpupower.pre_run_hook,
                   schedkit.start_sched_hook],
    post_run_hooks=[cpupower.post_run_hook,
                    schedkit.end_sched_hook],
    wrappers=[perfstatWrapper(event=[“cache-misses”])]
)
campaign.run()

campaign.generate_graph(
    plot_name="lineplot",
    x="nb_threads",
    y="throughput",
    hue="bench_name",
)
```

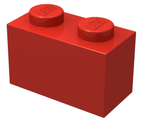
benchkit

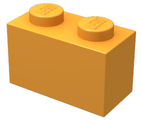
Ecosystem

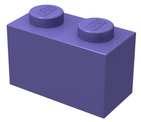
Benchmarks

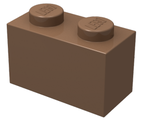

LevelDB

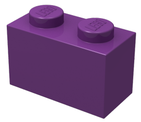

RocksDB


Kyoto Cabinet

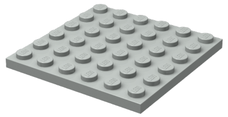

SPEC CPU 2017

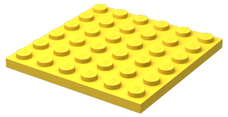

NPB

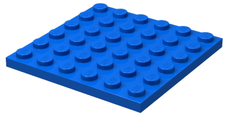

Volano

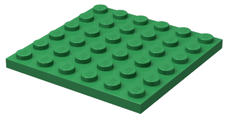

Will-it-scale

Platforms


Local (Linux)


Remote (SSH)


Docker

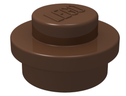

Mobile (ADB)

Command Wrappers

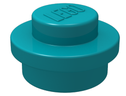

Perf


Taskset


Strace


Env

Pre and Post RunHooks

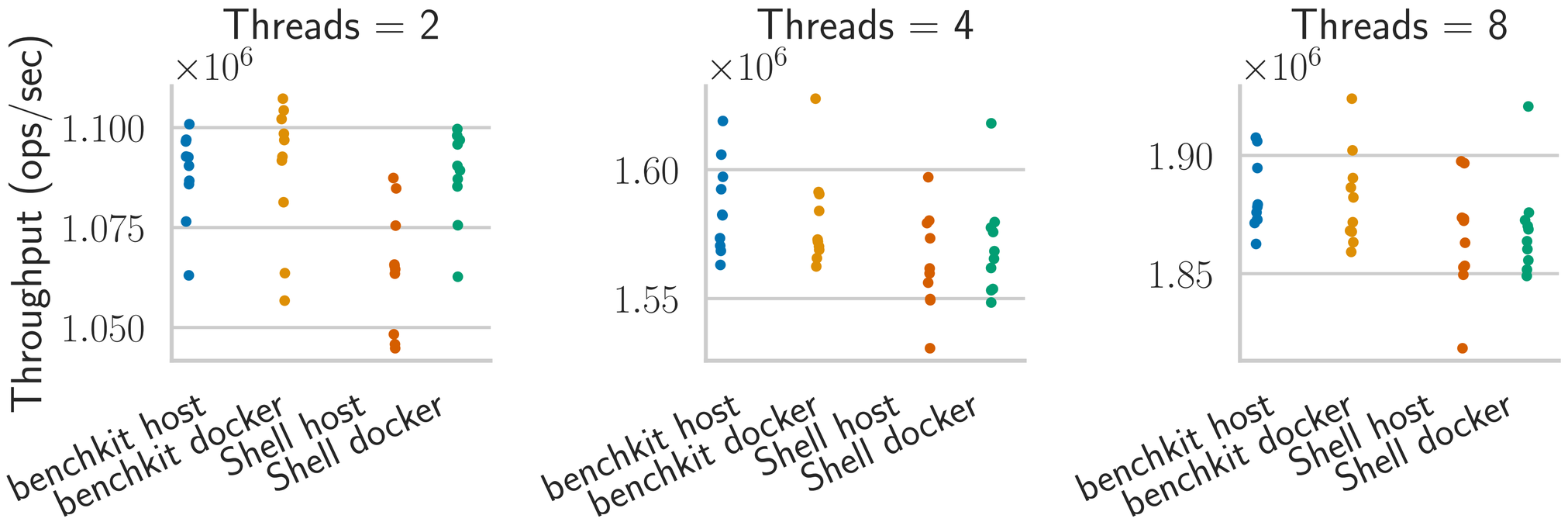

StressNG


Flame Graphs

RQ: What does the abstraction cost?

benchkit

Overhead Study



≤ 2,22% overhead vs hand scripted shell

https://github.com/softwarelanguageslab/icpe26-benchkit-ae/tree/main/experiments/fig15_overhead

benchkit turns performance evaluation from scripts into reusable experimental objects

Try benchkit Today!

https://github.com/open-s4c/benchkit

- Under MIT License
- 25+ contributors
- 14k+ lines of code
- 7 scientific papers
- Used to teach performance evaluation