task1

May 23, 2024

[ ]:
```
from pathlib import Path
import numpy as np
from matplotlib import pyplot as plt
import json
```

[ ]:
```
from dataclasses import dataclass, field
from statistics import geometric_mean
from typing import DefaultDict
from collections import defaultdict


@dataclass
class TraceStat:
    insn: int
    cycles: int
    misp: int
    mpki: float = field(init=False)
    ipc: float = field(init=False)

    def __post_init__(self):
        self.mpki = self.misp * 1e3 / self.insn
        self.ipc = self.insn / self.cycles


@dataclass
class TracelistStats:
    stats: dict[str, TraceStat] = field(default_factory=dict)

    def add_stat(self, trace_name: str, stat: TraceStat):
        self.stats[trace_name] = stat

    @property
    def mpki(self) -> float:
        return geometric_mean(map(lambda x: x.mpki, self.stats.values()))

    @property
    def ipc(self) -> float:
```

```
        return geometric_mean(map(lambda x: x.ipc, self.stats.values()))
```

[ ]:
```
pred = ["bimodal", "freq", "prob"]
DIR = Path.cwd() / "results"
stat_dirs = [DIR / p for p in pred]
stats: DefaultDict[str, TracelistStats] = defaultdict(TracelistStats)
for stat_dir in stat_dirs:
    for stat_file in stat_dir.iterdir():
        assert stat_file.suffix == ".json"
        with open(stat_file) as f:
            data = json.load(f)[0]["sim"]["cores"][0]
        stat = TraceStat(
            insn=data["instructions"],
            cycles=data["cycles"],
            misp=sum(data["mispredict"].values()),
        )
        del data
        stats[stat_dir.stem].add_stat(stat_file.stem, stat)
mpkis = dict(zip(stats, map(lambda s: s.mpki, stats.values())))
ipcs = dict(zip(stats, map(lambda s: s.ipc, stats.values())))
```

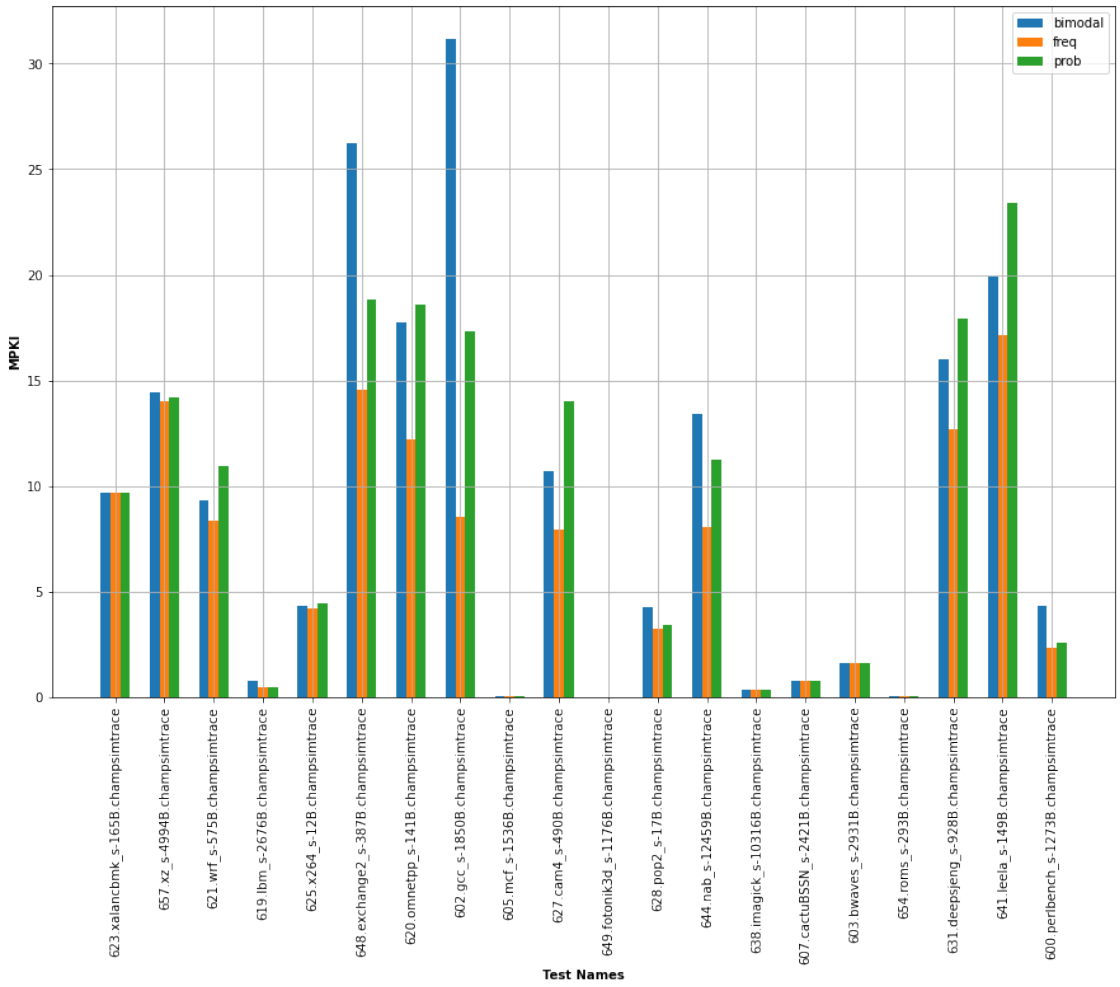
1 1

markov predictor : 1. (freq) 2. , (prob)

[ ]:
```
mpki_stats = dict(
    zip(
        stats,
        map(
            lambda v: dict(
                zip(v.stats, map(lambda x: x.mpki, v.stats.values()))
            ),
            stats.values(),
        ),
    )
)
ipc_stats = dict(
    zip(
        stats,
        map(
            lambda v: dict(
                zip(v.stats, map(lambda x: x.ipc, v.stats.values()))
            ),
            stats.values(),
        ),
```

```
    )
)
bar_w = 0.2
plt.figure(figsize=[15, 10])
for i, (pred, data) in enumerate(mpki_stats.items()):
    plt.bar(
        [r + bar_w * i for r in range(20)],
        data.values(),
        label=pred,
        width=bar_w,
    )
plt.xlabel("Test Names", fontweight="bold")
plt.ylabel("MPKI", fontweight="bold")
plt.xticks(
    [r + bar_w for r in range(20)],
    list(mpki_stats.values())[0].keys(),
    rotation=90,
)
plt.grid()
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7fa1d7a79180>
```

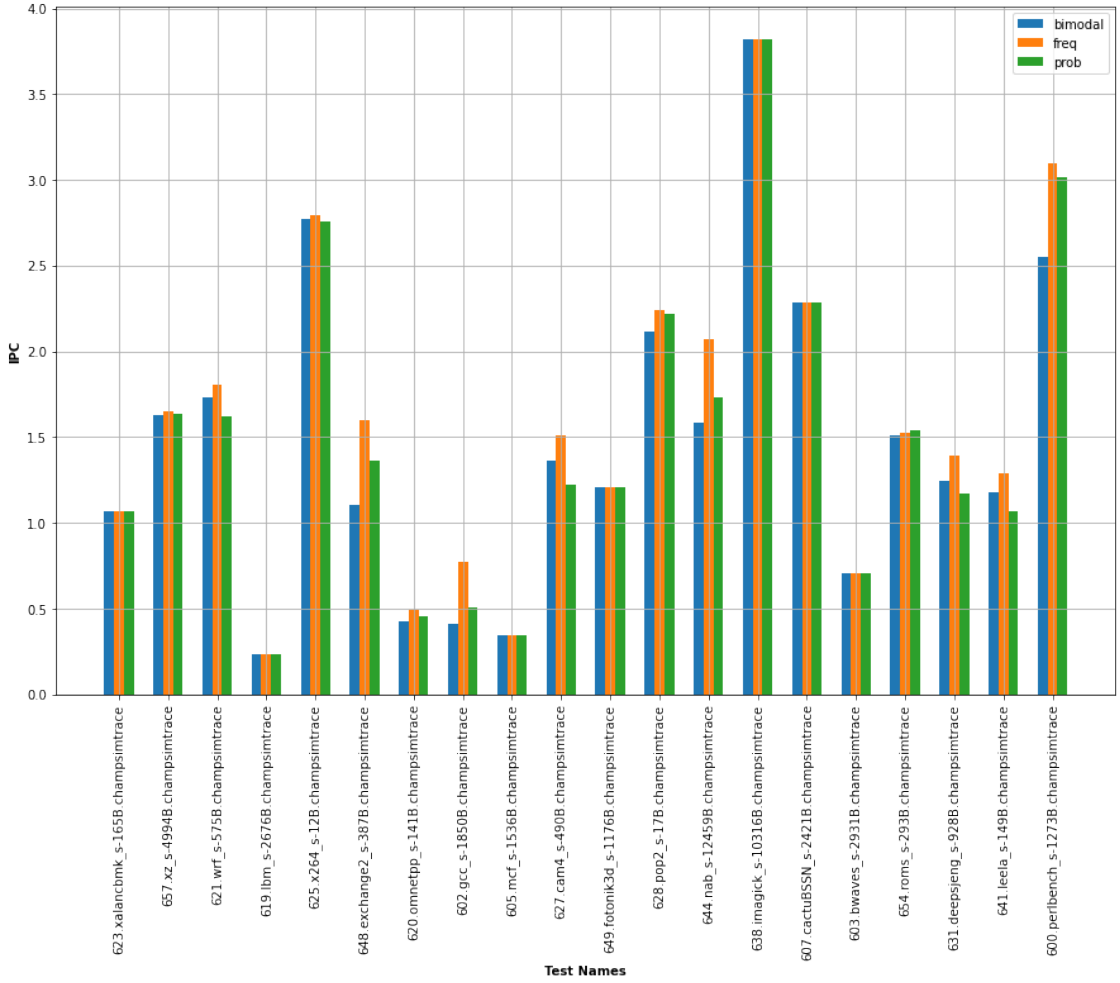


, ( ),
- - , .
(,
).

[ ]:
```
bar_w = 0.2
plt.figure(figsize=[15, 10])
for i, (pred, data) in enumerate(ipc_stats.items()):
    plt.bar(
        [r + bar_w * i for r in range(20)],
        data.values(),
        label=pred,
        width=bar_w,
    )
plt.xlabel("Test Names", fontweight="bold")
plt.ylabel("IPC", fontweight="bold")
plt.xticks(
```

```
    [r + bar_w for r in range(20)],
    list(ipc_stats.values())[0].keys(),
    rotation=90,
)
plt.grid()
plt.legend()
```

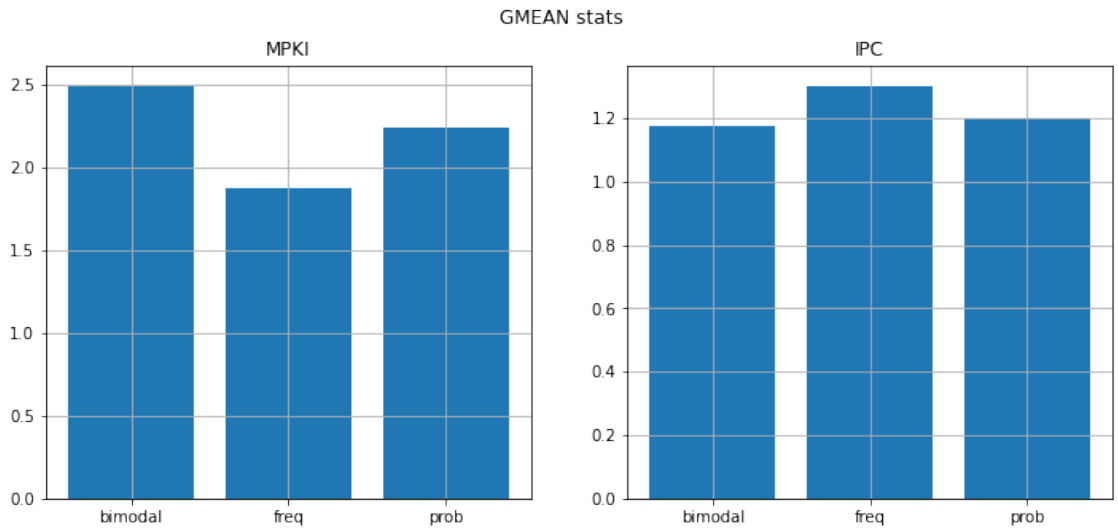
[ ]: <matplotlib.legend.Legend at 0x7fa1d747dea0>



IPC - , ,

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[12, 5])
fig.suptitle("GMEAN stats")
ax1.set_title("MPKI")
ax1.bar(mpkis.keys(), mpkis.values())
ax1.grid()
```

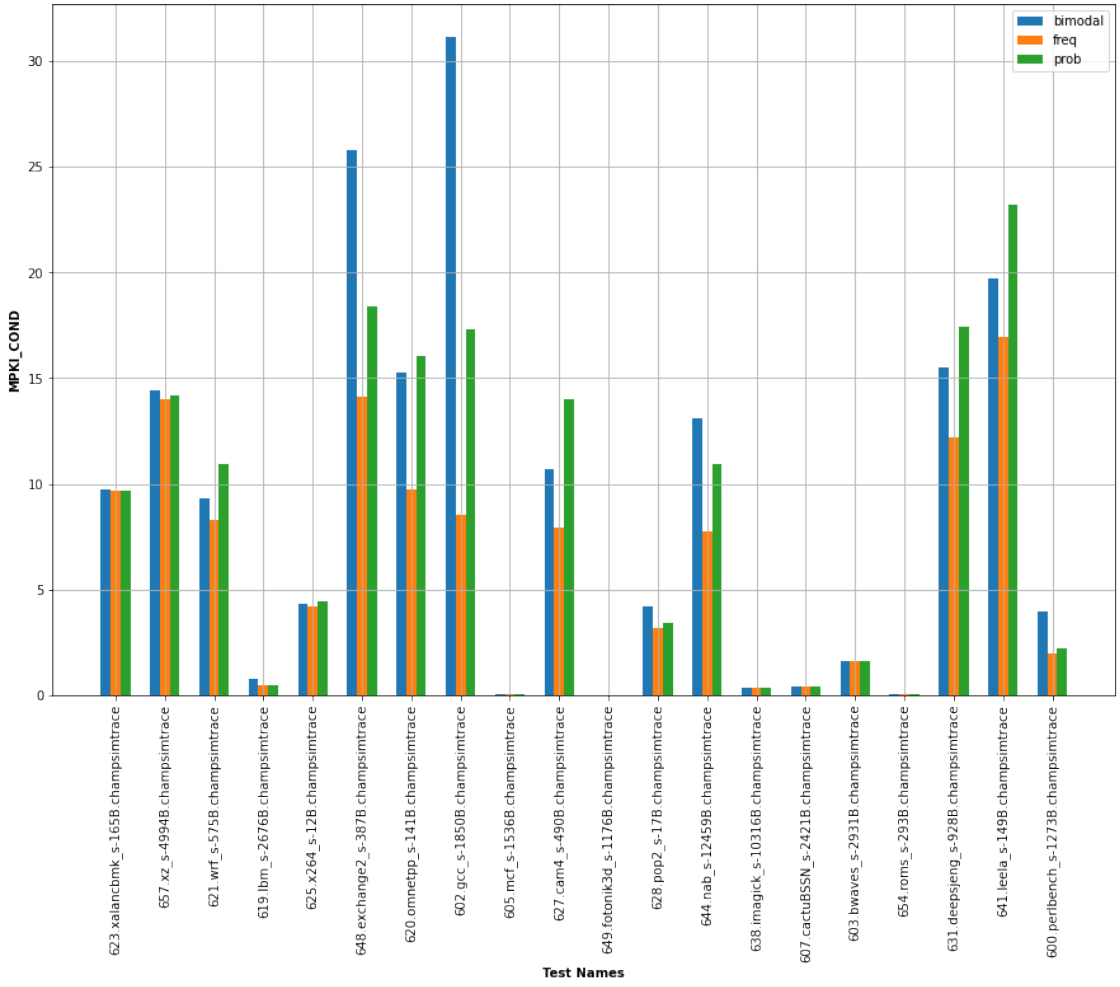
```
ax2.set_title("IPC")
ax2.bar(ipcs.keys(), ipcs.values())
ax2.grid()
```



Geomean mpki IPC. , ,
, . IPC . ,
. ,
( ) “ ” , CPU -
ROB

[ ]:
```
pred = ["bimodal", "freq", "prob"]
DIR = Path.cwd() / "results"
stat_dirs = [DIR / p for p in pred]
stats = defaultdict(dict)
for stat_dir in stat_dirs:
    for stat_file in stat_dir.iterdir():
        assert stat_file.suffix == ".json"
        with open(stat_file) as f:
            data = json.load(f)[0]["sim"]["cores"][0]
        mpki_cond = data["mispredict"]["BRANCH_CONDITIONAL"]
        stats[stat_dir.stem][stat_file.stem] = (
            mpki_cond * 1e3 / data["instructions"]
        )
mpki_stats = stats
bar_w = 0.2
plt.figure(figsize=[15, 10])
for i, (pred, data) in enumerate(mpki_stats.items()):
    plt.bar(
        [r + bar_w * i for r in range(20)],
```

```
        data.values(),
        label=pred,
        width=bar_w,
    )
plt.xlabel("Test Names", fontweight="bold")
plt.ylabel("MPKI_COND", fontweight="bold")
plt.xticks(
    [r + bar_w for r in range(20)],
    list(mpki_stats.values())[0].keys(),
    rotation=90,
)
plt.legend()
plt.grid()
```



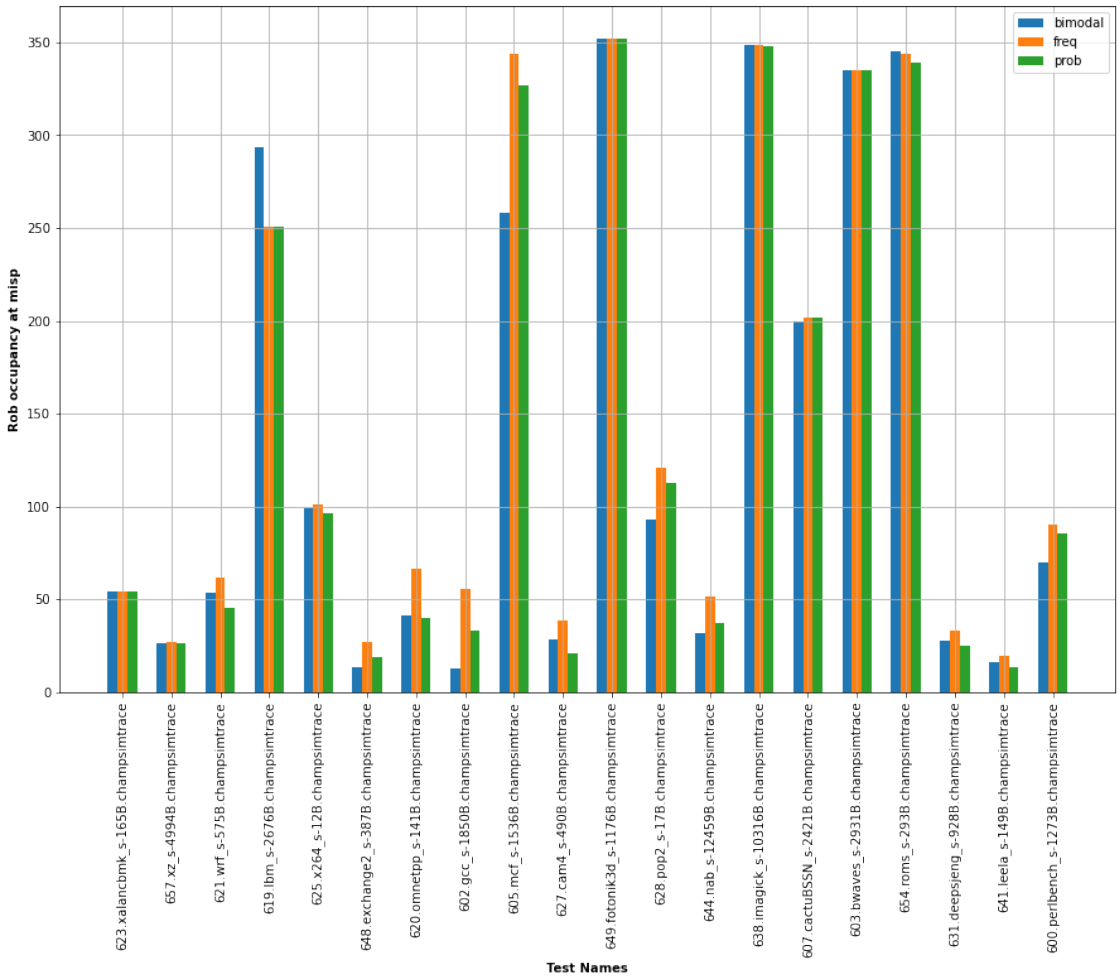
[ ]:

conditional branches

[ ]:
```
pred = ["bimodal", "freq", "prob"]
DIR = Path.cwd() / "results"
stat_dirs = [DIR / p for p in pred]
stats = defaultdict(dict)
for stat_dir in stat_dirs:
    for stat_file in stat_dir.iterdir():
        assert stat_file.suffix == ".json"
        with open(stat_file) as f:
            data = json.load(f)[0]["sim"]["cores"][0]
        rob = data["Avg ROB occupancy at mispredict"]
        stats[stat_dir.stem][stat_file.stem] = rob

bar_w = 0.2
plt.figure(figsize=[15, 10])
for i, (pred, data) in enumerate(stats.items()):
    plt.bar(
        [r + bar_w * i for r in range(20)],
        data.values(),
        label=pred,
        width=bar_w,
    )
plt.xlabel("Test Names", fontweight="bold")
plt.ylabel("Rob occupancy at misp", fontweight="bold")
plt.xticks(
    [r + bar_w for r in range(20)],
    list(stats.values())[0].keys(),
    rotation=90,
)
plt.legend()
plt.grid()
avgs = {k: np.mean(list(v.values())) for k, v in stats.items()}
print(f"Avg ROB occupancy at mispredict: {avgs}")
```

```
Avg ROB occupancy at mispredict: {'bimodal': 134.89204793529754, 'freq':
146.07604776821924, 'prob': 138.07368316869616}
```


bimodal
freq
prob
Rob occupancy at misp
0
50
100
150
200
250
300
350
623.xalancbmk_s-165B.champsimtrace
657.xz_s-4994B.champsimtrace
621.wrf_s-575B.champsimtrace
619.lbm_s-2676B.champsimtrace
625.x264_s-12B.champsimtrace
648.exchange2_s-387B.champsimtrace
620.omnetpp_s-141B.champsimtrace
602.gcc_s-1850B.champsimtrace
605.mcf_s-1536B.champsimtrace
627.cam4_s-490B.champsimtrace
649.fotonik3d_s-1176B.champsimtrace
628.pop2_s-17B.champsimtrace
644.nab_s-12459B.champsimtrace
638.imagick_s-10316B.champsimtrace
607.cactuBSSN_s-2421B.champsimtrace
603.bwaves_s-2931B.champsimtrace
654.roms_s-293B.champsimtrace
631.deepsjeng_s-928B.champsimtrace
641.leela_s-149B.champsimtrace
600.perlbench_s-1273B.champsimtrace
Test Names

ROB , , ,
,