IS2202 - Laboration 1
Simulating Caches and RISC-V processor Cores in GEM5

Linghan Zhao Teammate: Han Tu

March 11, 2024

In the preparation part, we install Ubuntu 22.04 natively as a dual boot and VirtualBox.

Directed Part

Exercise 1: Run and analyze statistics

Booting the RISC-V system inside GEM5.
Run the benchmark "hotspot3D", both Natively and in VirtualMachine.
Question 1)

The time for the application execution inside the simulator system is **0.069945 s**.

The time for the application execution took outside the simulator system (in a real environment) is approximately **22 minutes = 1320 s**, which is from time 15:41 to 16:03.

The time for Real time elapsed on the host (Second) is **40.31 s**.

Let's calculate how much slower was a computer simulation, $\frac{1320}{0.069945} \approx 18872$, namely $10^4\times$ slower.

The significant slowdown can be attributed to several factors inherent in the nature of detailed system simulations:

1. Software Overhead: Running simulations on a general-purpose computing system introduces additional layers of software overhead. The host system's operating system, along with the GEM5 simulation environment itself, consumes resources that could otherwise be used for the simulation.

2. Detailed Emulation: GEM5 simulates the intricate details of processor architectures, including instruction execution, cache hierarchies, and memory access patterns. Each of these operations requires complex calculations that are inherently more resource-intensive than executing instructions on actual hardware.

3. Accuracy Over Speed: GEM5 prioritizes accuracy over execution speed. This means that it often opts for more detailed, slower simulation methods rather than faster, less accurate ones.

4. Lack of Hardware Optimization: Unlike real ICs, which can execute billions of operations per second thanks to hardware-level optimizations and parallelism, simulations run on much more general hardware and must emulate these optimizations at a software

level, which is inherently less efficient.

Question 2)

Under gem5 there is a library called m5out. Inspect the m5out/stats.txt.

a) There are **12490495092** instructions was simulated (in Gem5).

b) The time for Real time elapsed on the host natively (Second) is **40.31 s**.
The host seconds passed (on the VM Ubuntu running Gem5) is **0.74 s**.

c) It takes **0.069945 s** to pass in the simulated machine (UcanLinux).

d) The hostInstRate (natively) is **32014947 (inst/s)**.
The hostInstRate (on VM Ubuntu running Gem5) is **344521355 (inst/s)**

e) To see how much slower is it to run the simulation compared to run it Natively,

	simSeconds	hostSeconds	simInsts	hostInstRate
Natively	0.069945	40.31	1290495092	32014947
VM	0.001068	0.74	255807712	344521355

Table 1: simulation for HotSpot3D

We need to know that the comparison is based on the running of
Natively:

```
./3D.riscv 512 1 1 input/power_512x8 input/temp_512x8 output.out
```

VirtualMachine:

```
./3D.riscv 64 1 1 input/power_64x8 input/temp_64x8 output.out
```

Running on VM and natively, $\frac{40.31}{0.069945} \approx 576.31$.

Running on VM and natively, $\frac{0.74}{0.001068} \approx 692.88$.

Question 3)

Now, repeat above steps but also for the applications SRAD and LUD.
Note that

simSeconds is Number of seconds simulated (Second)

hostSeconds is Real time elapsed on the host (Second)

simInsts is Number of instructions simulated (Count)

hostInstRate is Simulator instruction rate (inst/s) ((Count/Second))

For SRAD,

	simSeconds	hostSeconds	simInsts	hostInstRate
Natively	1.762128	1001.55	1362724830	1360621
VM	1.763103	1205.20	1583608056	1313982

Table 2: simulation for SRAD

For LUD,

	simSeconds	hostSeconds	simInsts	hostInstRate
Natively	13.720721	7791.24	8420100207	1080714
VM	0.450737	305.95	516541277	1688310

Table 3: simulation for LUD

Exercise 2: Checkpoints

In GEM5, the easiest way performing a checkpoint is to make a checkpoint from inside the guest system. Start the simulator and the system as previously, make a checkpoint inside the OS, and then resume from the checkpoint we created.

Question 4)

Without the checkpoint, it takes **4 minutes and 32 seconds = 272 s** to boot the entire system.

With the checkpoint, it takes **15 second** to boot the entire system.

Let's calculate how much faster, $\frac{272}{15} = 18.13$ *times* faster to launch from the checkpoint than to boot the entire system.

Exercise 3: Simulating a cache memory hierarchy

Attach a simple cache hierarchy to our RISC-V system and explore the impact of the cache.

Firstly, we set up a system with no caches, rerun one of the three applications from exercise #1 and time it.

Question 5)

We choose to rerun **HotSpot3D** without cache, natively.

	simSeconds	hostSeconds	simInsts	hostInstRate
without cache	3.582664	125.26	1597573917	12753894
with cache	0.069945	40.31	1290495092	32014947

Table 4: simulation for HotSpot3D without/with cache

Calculate how much slower without cache, for simSeconds 5.12 *times* slower, and for hostSeconds 3.11 *times* slower.

Then We are going to setup a single PrivateL1PrivateL2 cache subsystem.

Now resume our previous checkpoint, and inspect the statistics (./m5out/stats.txt) to see cache performance.

Question 5)

We choose **HotSpot3D** and run with the cache (use the checkpoint).

The impact on the performance compared to the case NoCache as below.

	simSeconds	hostSeconds	simInsts	hostInstRate
NoCache	3.582664	125.26	1597573917	12753894
PrivateL1PrivateL2	0.071529	39.89	1280644608	32101957

Table 5: performance compared with NoCache

We can see that the Number of seconds simulated, Real time elapsed on the host and Number of instructions simulated all decrease. And the Simulator instruction rate (inst/s) increases.

It cost **3757000** clock cycles in Latency, which is number of demand (read+write) miss ticks (Tick).

l1dcaches.overallHits	19079414	# number of overall hits (Count)
l1icaches.overallHits	49635474	# number of overall hits (Count)
l1dcaches.overallMisses	86743	# number of overall misses (Count)
l1icaches.overallMisses	29634	# number of overall misses (Count)
l2caches.overallHits	38445	# number of overall hits (Count)
l2caches.overallMisses	16595	# number of overall misses (Count)

Table 6: Important numbers

Discuss the reasons of the performance improvement:

1. Reduced Simulation and Real Time: Caching reduces the need for slow memory accesses by storing frequently accessed data close to the processor. This efficiency speeds up the simulation, decreasing both the simulated seconds and the real time elapsed on the host.

2. Fewer Instructions Simulated: With data more readily available in the cache, fewer instructions, especially those related to memory access, need to be simulated. This reduction contributes to the overall decrease in simulation time.

3. Increased Instruction Rate: The simulation can process instructions faster because it spends less time waiting on memory accesses. As a result, the simulator instruction rate (inst/s) increases, indicating a more efficient simulation process.

These improvements are consistent with the known benefits of caching in computer architectures, demonstrating the value of accurately simulating these components in architectural studies.

Exercise 4: Finding out the application cache working set

Determine the working set size of each of the benchmarks by varying the size of the simple PrivateL1PrivateL2 cache.

Question 6)

By incrementally increasing the L1D data-cache sizes and record the measurements. (remain l1i_size= 2KiB , l2_size= 64KiB)

For **HotSpot3D**

L1D data-cache sizes	simSeconds	l1dcaches.overallMissLatency
2 KiB	0.071529	2056889000
4 KiB	0.071643	2158197000
8 KiB	0.071462	1976408000
16 KiB	0.071140	1604948000
32 KiB	0.070792	1238724000
64 KiB	0.070648	1153704000

Table 7: simulation for HotSpot3D increments L1D data-cache sizes

For **SRAD**

L1D data-cache sizes	simSeconds	l1dcaches.overallMissLatency
2 KiB	2.019181	399704631000
4 KiB	1.871111	243093063000
8 KiB	1.847962	219165084000
16 KiB	1.843146	212791475000
32 KiB	1.844730	213323883000

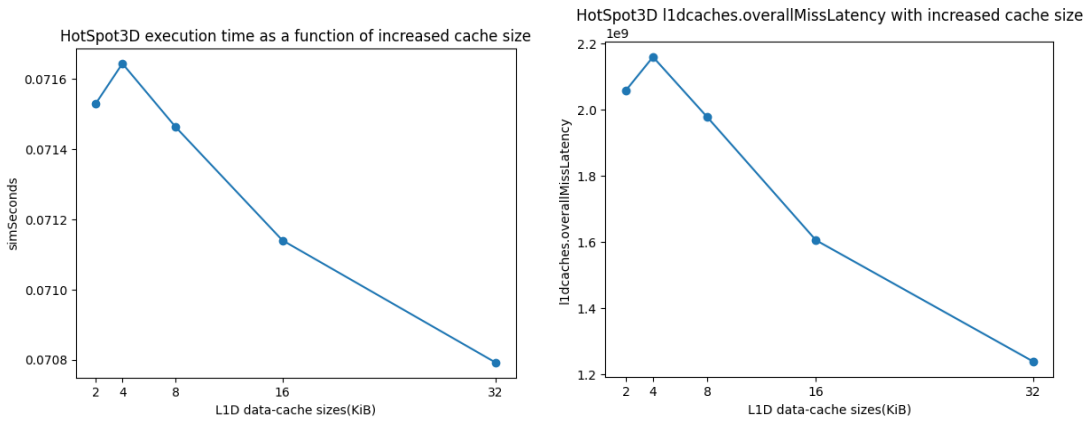
Table 8: simulation for SRAD increments L1D data-cache sizes

For **LUD**

L1D data-cache sizes	simSeconds	l1dcaches.overallMissLatency
2 KiB	0.457566	7106297000
4 KiB	0.456994	6422495000
8 KiB	0.456547	5986519000
16 KiB	0.454785	3949158000
32 KiB	0.453591	2596260000
64 KiB	0.451986	1394246000
128 KiB	0.451654	1153323000

Table 9: simulation for LUD increments L1D data-cache sizes

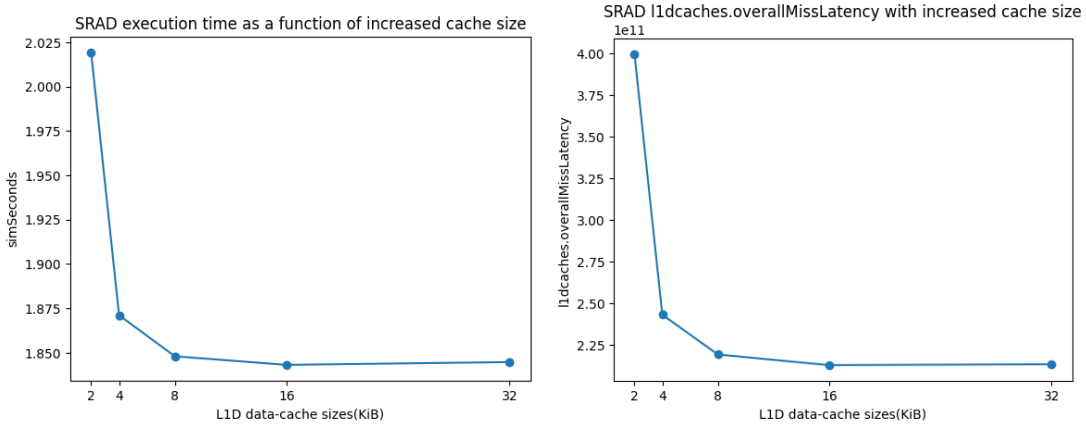
Plot a graph showing how the execution time is reduced as a function of increased cache size.



(a) execution time

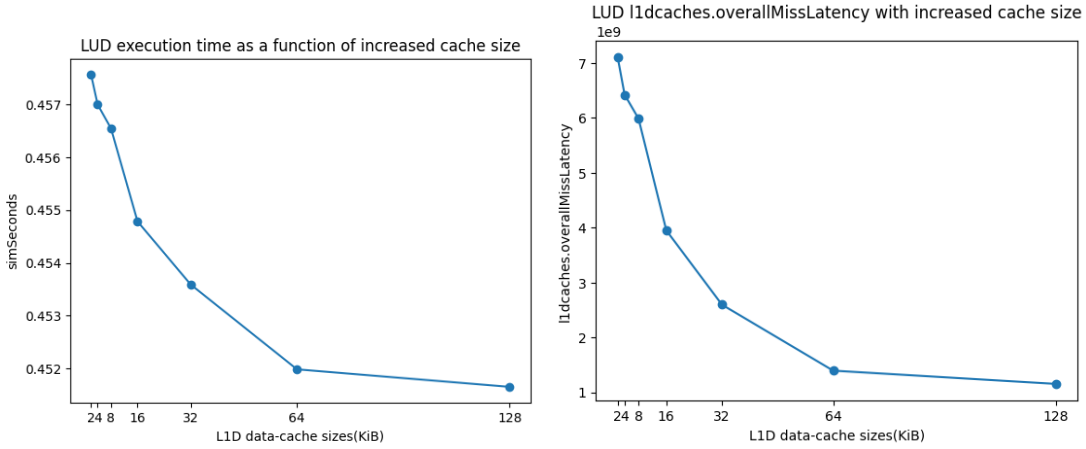
(b) l1dcaches MissLatency

Figure 1: simulation for HotSpot3D increments L1D data-cache sizes



(a) execution time (b) l1dcaches MissLatency

Figure 2: simulation for SRAD increments L1D data-cache sizes



(a) execution time (b) l1dcaches MissLatency

Figure 3: simulation for LUD increments L1D data-cache sizes

Let's identify the cache working set size for each benchmark according to the variation of the curves, at which point the curve's decreasement not much strong. So the cache working set size for Hotspot3D should be around 16 to 32 KiB, for SRAD should be around 8 to 16 KiB, for LUD should be around 64 to 128 KiB.

Question 7)

Choose the best configuration above as Working set size(L1Data Cache).

Hotspot3D	SRAD	LUD
32 KiB	32 KiB	128 KiB

Table 10: Best config. working set size(L1Data Cache)

For **HotSpot3D**, the best configuration above is l1d_size=32 KiB, l1i_size=2 KiB, l2_size=64 KiB, then doubling the L2 cache size to l2_size=128 KiB. Similar for SRAD and LUT. See table below.

Note that board.cache_hierarchy.l2caches.overallMissLatency::total is the total miss latency the processor experience due to misses to the cache, board.cache_hierarchy.l2caches.

overallMissLatency::cache_hierarchy.l1dcaches.prefetcher is the L2 cache absorb misses from the L1.

	L2 cache size	simSeconds	L2 absorb miss from L1	l2.totalMissLatency
HotSpot3D	64 KiB	0.070792	1673871954	3031049954
	128 KiB	0.07070	1681103951	2952482951
SRAD	64 KiB	1.844730	278898935207	442156445751
	128 KiB	1.798264	206041225781	315637002260
LUD	64 KiB	0.451654	516949991	2038935991
	128 KiB	0.451348	470285997	1619803997

Table 11: simulation for doubling L2 cache size

The time for simulation decreases, the L2 cache absorb misses from the L1 increases when choose double size L2 cache. And the total miss latency the processor experience due to misses to the L2 cache decreases.

These performance changes means improvements of time consuming in cache miss latency, with bigger L2 cache size.

Question 8)

The benchmark **LUD** has the largest working set, based on the former analysis.

Question 9)

In the above measurements, compute the L1 and L2 cache hit rates. Note that
Hotspot3D with l1d_size=32KiB, l1i_size=2KiB, l2_size=128KiB
SRAD with l1d_size=32KiB, l1i_size=2KiB, l2_size=128KiB
LUD with l1d_size=128KiB, l1i_size=2KiB, l2_size=128KiB

		caches total Hits	caches total Misses	cache Miss rates
HotSpot3D	L1D	19139548	26728	0.001395
	L2	47146	35664	0.430673
SRAD	L1D	225531929	5971793	0.025785
	L2	9525907	3552109	0.271609
LUD	L1D	93703583	18645	0.000199
	L2	41685	21457	0.339821

Table 12: computation of the L1 and L2 cache miss rates

We calculate the Miss rates of L1D and L2 caches, with formulas below.

$$MissRate = \frac{CacheMisses}{CacheMisses + CacheHits} \tag{1}$$

$$HitRate = 1 - MissRate \tag{2}$$

we can see that which has lowest miss rate means highest hit rate. So LUD's has the highest L1Data cache hit rate, SRAD has the lowest. SRAD has the highest L2 cache hit rate, HotSpot3D has the lowest.

Exercise 5: Multithreading and Speed-up

Now we are going to study multi-core system. Open the system configuration file, change the processor editing the number of cores to two. Next, launch the simulator and boot

the image.

Question 10

	num of core	simSeconds
HotSpot3D	1	0.069945
	2	0.069858
SRAD	1	1.762128
	2	0.883697
LUD	1	13.720721
	2	6.872400

Table 13: simulation for doubling cache count

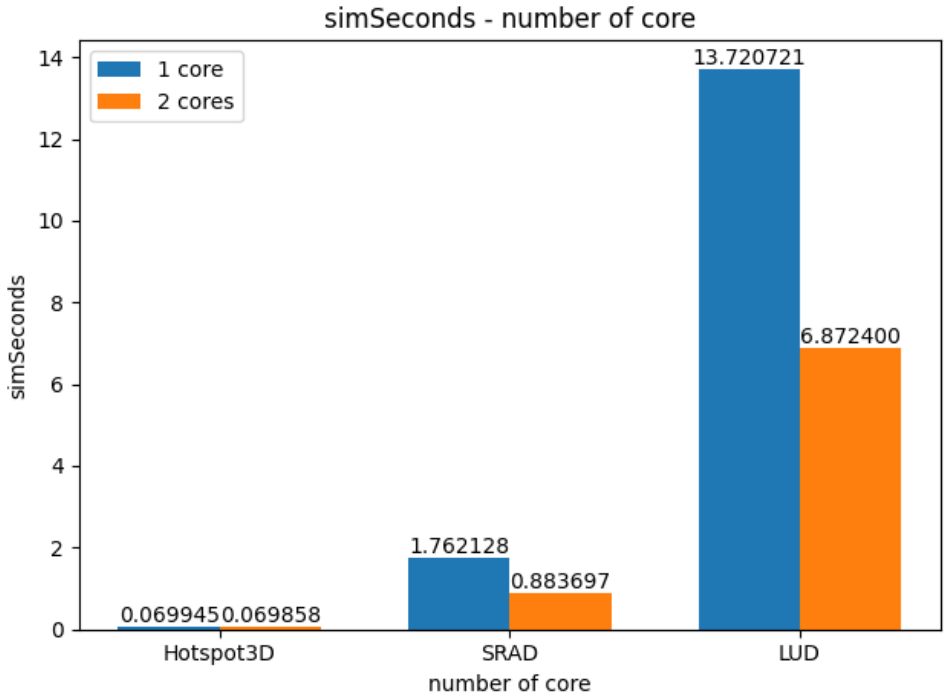


Figure 4: performance improves with the number of cores

When set the number of core from 1 to 2, the time for SRAD and LUD simulation decreases, speedup the execution of these applications. But for Hotspot3D there's few changes in the simSeconds.

Question 11

The **Hotspot3D** application has not been parallelized, and thus could not benefit from more cores, according to the former analysis of the changes of simulation time.

Open Ended Portion: Running an application of your choice

We cross-compiled the applications from OpenMP suite for this part and choose one of them to exploit:

- Benchmark suite: Bots OpenMP Suite
- Application: N-queens
- Version: omp-task
- Input size: 10
- Statistic collection: m5 utilities
- Time complexity: O(n^3)

Program analysis

N-queens is a classical problems which can be solved by backtracking. Its computation likes a tree expanding in its solution space from a single node. The subtrees with nodes in the same layer have no data dependence to each other, so finding new solutions on a specific root can be executed in parallel.

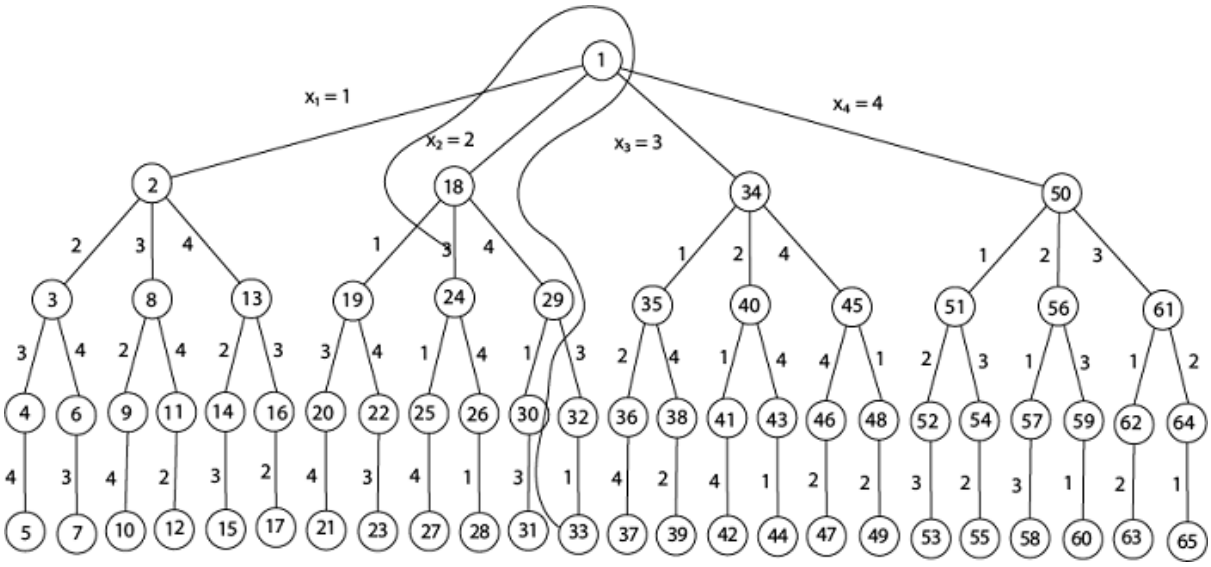


Figure 5: 4 - Queens solution space with nodes numbered in DFS

The computation of the program focuses on 2 parts:

1. Verifying if the subsolution is valid:

```
int ok(int n, char *a)
{
     int i, j;
     char p, q;

     for (i = 0; i < n; i++) {
          p = a[i];

          for (j = i + 1; j < n; j++) {
               q = a[j];
               if (q == p || q == p - (j - i) || q == p + (j - i)
                  )
```

```
                    return 0;
            }
        }
        return 1;
}
```

2. Test every possible positions under the current subsolution(involves memory allocation and memory block copy(read and write)):

```
for (i = 0; i < n; i++) {
    #pragma omp task untied if(depth < bots_cutoff_value)Â§Â§
    {
        /* allocate a temporary array and copy <a> into it */
        char * b = (char *)alloca(n * sizeof(char));
        memcpy(b, a, j * sizeof(char));
        b[j] = (char) i;
        if (ok(j + 1, b))
#ifndef FORCE_TIED_TASKS
            nqueens(n, j + 1, b,&csols[i],depth+1);
#else
            nqueens(n, j + 1, b,depth+1);
#endif
    }
}
```

Parameter selection

1. No Cache vs With Cache

Firstly, we set up the system with and without cache separately to make the comparison, running natively. Case of with cache has the cache parameter " l1d_size=32KiB, l1i_size=32KiB, l2_size=128KiB ".

	simSeconds	hostSeconds	simInsts	hostInstRate
without cache	19.212773	1354.19	484927752	358094
with cache	0.575586	333.62	364875995	1093688

Table 14: simulation for Bots OpenMP Suite without/with cache

Calculate how much slower without cache, for simSeconds $33,12$ *times* slower, and for hostSeconds 4.06 *times* slower.

2. Cache Size

Then we are going to setup a PrivateL1PrivateL2 cache subsystem.

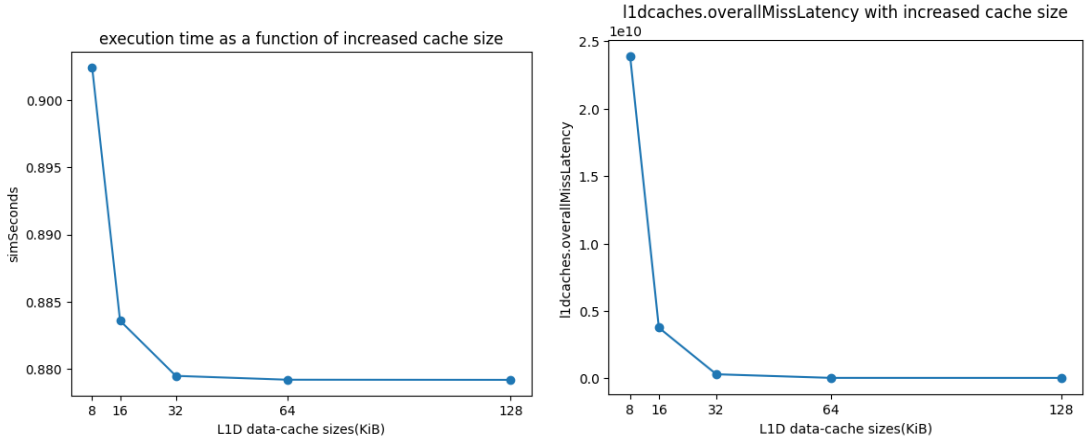
For L1D data-cache size

By incrementally increasing the L1D data-cache sizes and record the measurements. (remain l1i_size= 2KiB , l2_size= 64KiB)

L1D data-cache sizes	simSeconds	l1dcaches.overallMissLatency
8 KiB	0.902413	23892982000
16 KiB	0.883602	3762988000
32 KiB	0.879502	311833000
64 KiB	0.879216	43070000
128 KiB	0.879206	40569000

Table 15: simulation with incrementing L1D data-cache sizes

Plot a graph showing how the execution time is reduced as a function of increased cache size.



(a) execution time

(b) l1dcaches MissLatency

Figure 6: simulation for Bots OpenMP increments L1D data-cache sizes

These improvements are consistent with the known benefits of caching in computer architectures, demonstrating the value of accurately simulating these components in architectural studies.

Let's identify the cache working set size for this benchmark according to the variation of the curves, at which point the curve's decreasement not much strong. So the cache working set size for Bots OpenMP suite applying to N-queen should be around 128 KiB.

Doubling the L2 cache size

Choose the configuration a l1d_size=128 KiB, l1i_size=2 KiB, l2_size=64 KiB, then doubling the L2 cache size to l2_size=128 KiB. See table below.

L2 cache size	simSeconds	L2 absorb miss from L1	l2.totalMissLatency
64 KiB	0.879206	12627997	125802997
128 KiB	0.879166	11444997	96143997

Table 16: simulation for doubling L2 cache size

The time for simulation decrease a little, the L2 cache absorb misses from the L1 also decreases because the total miss decrease, when choose double size L2 cache.

Follow the above measurements, compute the L1 and L2 cache hit rates. Choose the configuration a l1d_size=128 KiB, l1i_size=2 KiB, l2_size=128 KiB, see the table below.

	caches total Hits	caches total Misses	cache Miss rates
L1i	424720587	21133635	0.047400
L1D	127550804	509	0.000004
L2	21133033	1333	0.000063

Table 17: computation of the L1 and L2 cache miss rates

We calculate the Miss rates of L1i, L1D and L2 caches, with formulas we mentioned before EQUATION 1. The L1i cache hit rate is 0.9526, L1Data cache hit rate is 0.999996, L2 cache hit rate is 0.999937. These are very high Hit Rates, which express good performances.

3. Core Count(single core vs dual core)

Now we are going to study multi-core system. Case of this has the cache parameter " l1d_size=32KiB, l1i_size=32KiB, l2_size=128KiB ". Change the processor editing the number of cores to two.

num of core	simSeconds
1	0.575586
2	0.406161

Table 18: simulation for doubling cache count

When set the number of core from 1 to 2, the time for simulation decreases, speed up the execution of these applications.