CPRE 581 - X86 Gem5 Checkpoint Generation with SimPoint

Abir Mojumder

December 13, 2023

Abstract

In the realm of simulation, official benchmarks on a cycle-level accurate simulator can take an extremely long time in order to provide an accurate evaluation. Gem5 supports detailed microarchitecture simulation, and with the time constraints for the course, there is a need to run simulations quickly, and produce a feasible result. Simpoint checkpoints allow the execution of certain portions of a program that can save time, but still provide the expected behavior of said program. This report discusses the process of creating checkpoints in Gem5 using the SimPoint tool for sampling. An analysis is then performed on the BBV matrices, simulation speedup and CPI accuracy from a selection of SPEC2006 programs.

Introduction

The problem of long running programs is not only relevant to research studies, taking months to complete execution, but also an issue for university courses that expects its students to study simulators often having only a few days to complete running benchmarks. To shorten the simulation execution time frame, there has been prior research into analyzing program behavior. Programs generally have recurring behavior which consumes most of the time during simulation. This behavior is referred to as a program's phase [5]; the classification of these phases allows the creation of a few simulation regions that are representative of the repeated behavior. This allows a single execution of the region to greatly decrease the execution time while achieving the target behavior. To address this very issue, researchers at the University of California, San Diego created the SimPoint tool. This tool analyses several execution intervals from the entire execution based on basic block vectors and performs automated phase analysis to create checkpoints. These checkpoints can then be used with the Gem5 simulator to jump to the regions of interest and only simulate that portion of the program.

If checkpoints for industrial benchmarks exist, why do this project? Benchmarks have several billions of instructions taking months to complete when using a reference data set, so it makes sense to use checkpoints for research work. However, students learning about microarchitectural behavior in school are likely not going to spend this much time on one program, but instead may be provided with a much simpler program to study the behavior of cache eviction for example. This program could be a couple hundred million instructions long and a detailed simulation of this too could take weeks. The motivation for this project is to deal with such an issue by understanding checkpoint creation using SimPoint and learning how to use it with Gem5 to speed up simulation experiments. This report discusses the process of creating Basic Block Vectors of SPEC2006 benchmarks in Gem5, converting the BBVs into checkpoints for the simulator, and comparing a few of the benchmarks for accuracy. The results are then analyzed to determine if the process was successful and whether the checkpoints really represent the performance of a full simulation.

Related Works

The main paper guiding this project is “Automatically Characterizing Large Scale Program Behavior” by Sherwood et al.[4]. This paper discusses in detail, the various procedures in characterizing program behavior which determines the simulation points generated from classifying the BBVs. It discusses how 2 Basic Block Vectors can be used to calculate the Manhattan or the Euclidean distance between 2 execution intervals. If the 2 intervals are similar, it means that the code executed and therefore their behavior are similar. Similar intervals are referred to as phases and are the basis for generating the simulation points. Large programs usually have several intervals, therefore choosing a set of intervals to create a representative cluster further shortens execution time [3]. They discuss more about clustering and K-means phase finding algorithms that they implement to aid in automating Simpoint generation. The authors present two forms of Simpoint execution; a long single Simpoint or multiple Simpoints. The experiments in this report uses multiple simulation points as the paper showed a smaller error rate in IPC when compared to the real simulation. Originally, the choice of the single simulation (starting) point depended on the BBV with the smallest Manhattan distance as it represented the code closest to the entire execution. Their newer method chooses a BBV with the lowest Euclidean distance to the centroid of the 15D similarity matrix. The authors advise using multiple points so that sections can be broken down further and run in parallel while having a smaller error rate.

Sandberg et al.[2] call this technique sampled simulation, which is the whole genre of improving simulation speed for architectural performance evaluation. Their paper discusses the issue of cold-starting caches when virtualizing the sample start point and how it takes a significant amount of time to warm-up caches as a necessary step to evaluate realistic performance of the sampled area. They use some hardware virtualization to speed up the fast-forwarding, but they present a novel approach to solve the cache-warming problem. They evaluate the impact of insufficiently warmed cache and experiment with ways to exploit sample-level parallelism. They claim the speedup comes from separately warming up in parallel to fast-forwarding virtually to minimize waiting times: "the functional and detailed simulation of previous samples execute in parallel while the virtualized fast-forwarding runs ahead to generate the next sample".

Another paper discussing the start-up procedure for sampling by Van Biesbrouck et al.[6] compares how the state of the starting image affects the quality of the simulation. They discuss 2 starting states, *fast-forwarding* and a *full checkpoint set* by comparing it to their approach of *touched memory image* which is claimed to consume a smaller disc space than regular checkpoints, while having the simplicity of fast-forwarding. They touch on the subject warming up the starting state; some strategies such as *Hit-on-cold* and *fixed-length-warm up*. They conclude that 2 warming strategies work best for accuracy and efficiency: "Memory reference reuse latency" and another based on the "Memory hierarchy state"".

With regards to sampling performance in general, there has been prior work in this field to study the tradeoff between accuracy and speed by J.Yi et al. and Wunderlich et al.[7],[8]. The *SMARTS* paper by Wunderlich et al. proposes a sampling methodology that utilizes inferential statistics to calculate the confidence values of samples. They compared their mean benchmark runtime to SimPoint and found that their SMARTS benchmarks took twice as long, but had an extremely small CPI error rate of around 0.64%. Their 8-way out-of-order Pentium 4 showed a 30x speedup over simulating without sampling. The paper by J.Yi et al. also compared the speed and accuracy of SMARTS and SimPoint (and various other methods). They also analyzed the configuration dependence of the different sampling methods. It was found that SimPoint, SMARTS, and random sampling had the highest accuracy and least configuration dependence when compared to the reduced input set and truncated execution techniques. They also concluded that Simpoint had a better speed vs accuracy *SvAT* tradeoff even though SMARTS had noticeably higher accuracy.

A recent study by Sabu et al.[1] introduces another checkpoint-driven sampling technique called *LoopPoint*. The idea of this paper is to develop a method to sample multi-threaded applications where earlier research in this area did not provide significant speedup. Their method claims to have a 11,587x speedup on average using the SPEC2017 reference set benchmarks. The technique for multi-threaded sampling cannot use IPC as a reference for BBV, but instead requires measuring the "work-done" by each thread. They discuss how BBV capturing required a change to support parallelism, by using a global BBV to concatenate intervals from multiple-threads. Various other techniques were proposed such as loop searching via Dynamic-Control Flow Graphs, etc proving a massive speedup.

Main Idea

The project consists of 6 stages:
1. Selecting benchmarks and their data sets from SPEC2006 suite.
2. Generate BBV from each of the benchmark programs via Gem5.
3. Feed the BBV data into the SimPoint tool to get simulation points and their respective weights.
4. Re-run the benchmarks in Gem5 with the simulation points to capture sampling information (checkpoints).
5. Run benchmarks with the created checkpoints (Multiple checkpoint runs per program).
6. Compare the accuracy in terms of CPI, and the speedup in terms of real execution time.
These stages are further described in the next section, with details on what commands were used, what simulation points were made, etc.

Experiment Methodology

For this project, there were 3 main tools used: `Gem5 - 23.0.1.0`, `SimPoint 3.2`, and a python program to visualize the basic block similarity matrices (referred to as `BBMatrix`). The initial plan was to cover the entire benchmark suite, however several of the programs crashed due to memory issues and kernel panics a few days into their execution. The benchmarks that reached completion were: `astar`, `bzip2`, `gobmk`, `h264ref`, `hmmer`, `namd` and `sjeng`.

Generating BBV of the programs is a built-in feature of the Gem5 version used for this project. It requires just 2 flags: `--simpoint-profile` and `--simpoint-interval N`. This tells Gem5 that while the simulation is running, it needs to sample every N instructions for the BBV intervals. N was chosen to be 10 million in this case. A smaller interval results in more accurate profiling of similar regions, but also produces many more simulation points. The SimPoint paper [4] had a limit of 6 simpoints per program, but this bound was not set for this experiment (ie. the number of simpoints depended on the number of instructions executed). Once the full simulation completes, it generates a gzipped "simpoint.bb.gz" file.

The SimPoint tool is required next to convert the BBVs into discrete simulation points via its automated phase classification algorithm. This produces 2 files, one containing the list of instruction intervals, and another with the weight of each simulation point. The interval is a small number, for example *24*. It tells Gem5 that the actual instruction pointer is at (24 - 1) * *interval size*, so in this case the simulation point is located at instruction 230 million from the start of execution. The weight value for that simpoint is the fraction of the number of instructions in that cluster from the total number of instructions.

The generated simulation points can now be fed back into Gem5 with the corresponding benchmark program to start creating checkpoints. Gem5 reads the weight and simpoint files with the command `--take-simpoint-checkpoint` to start the simulation and set up the state of execution once that region is reached. This means that a warmup length needs to be input among the parameters. Smaller intervals require a longer warmup phase, which adds significant overhead to the time required to generate the checkpoints. Taking `bzip2` as an example, BBV generation with the test data set required 3 hours, and generating the checkpoints took another 2 hours and 40 minutes. The warmup length is chosen to be 1 million, as the SimPoint authors often used this value with their 10 million interval simulation points to be fairly optimal. Originally Gem5 required the use of the `AtomicSimpleCPU` to correctly generate checkpoints, however as of writing this report it requires the use of the `NonCachingSimpleCPU` model.

Once the checkpoints were generated for the aforementioned SPEC2006 programs, Gem5 can now execute these simulation points theoretically with any CPU configuration as seen in the homework. One constraint is that the configuration should have a maximum memory of 512MB since this was used during the creation of the checkpoints and the warm-up phase may not switchover otherwise. Due to time constraints, only the `NonCachingSimpleCPU` was used to re-run the benchmarks using the checkpoints to compare with the full run during BBV creation. As mentioned in the Related Works section, for accuracy comparison, multiple checkpoints are simulated instead of a single long checkpoint. The weighted CPI for each checkpoint run is then calculated before converting it to IPC. Without weights, since a smaller weight checkpoint is going to run faster (fewer instructions in that cluster)it would skew the results. The same weighted process is required to compare other forms of error rates.

Results

As there were no bounds to the number of simulation points, different programs produced a varying number of simpoints. Table 1 shows the number of simulation points determined by the SimPoint tool when `kMax` was set to 30, which is the maximum number of phases to detect. There is no direct correlation between the initial number of BBVs and the number of simpoints. It seems that depending on the Bayesian Information Criterion(BIC) analysis of the 30 possible phases, it selects phases that score the highest (exact algorithm and analysis details in [4]). From these scores, it selects the best phases and their corresponding simulation point.

Benchmark ID	**# of Simpoints**	**BBV Size**
astar	15	3.5MB
bzip2	19	1.3MB
hmmer	13	270MB
gobmk	23	897MB
h264	17	77MB
namd	24	10.6MB
sjeng	7	21MB

Table 1: Number of simulation points generated from the programs

An attempt was made to see if the basic block similarity matrices had any direct relation to the number of simulation points. Figure 1 and Figure 2 shows the similarity matrices of `bzip2` and `astar` respectively. This were generated using the BBMatrix tool, by taking the Manhattan distance. The program reduced the input basic block vectors to 15 dimensions, similar to what was done by the SimPoint authors. This normalizes the values, with the axes representing N millions of instructions. The dark regions show intervals that are similar (smaller Manhattan distance), while lighter regions show intervals that are different. Bzip2 shows more recurring patterns than Astar, however, Astar has larger concentrated segments that have similar behavior. Ideally, the checkpoints should cover as much of the darker regions as possible to illustrate the behavior of the entire program's execution, although the point is that a single chunk of the recurring behavior should suffice.

Once the programs were simulated using their set of checkpoints, each checkpoint execution's CPI and runtime were logged. As mentioned earlier, to get the representative CPI, a weighted CPI must be calculated. For speedup values, the full simulation's runtime is divided by the total run time of all the checkpoints. The checkpoints were run sequentially, therefore executing them in parallel would show an even larger speedup; this is especially useful when running the training or larger reference data sets. Figure 3 shows the CPI values of 5 benchmark programs; CPI values from the checkpoints

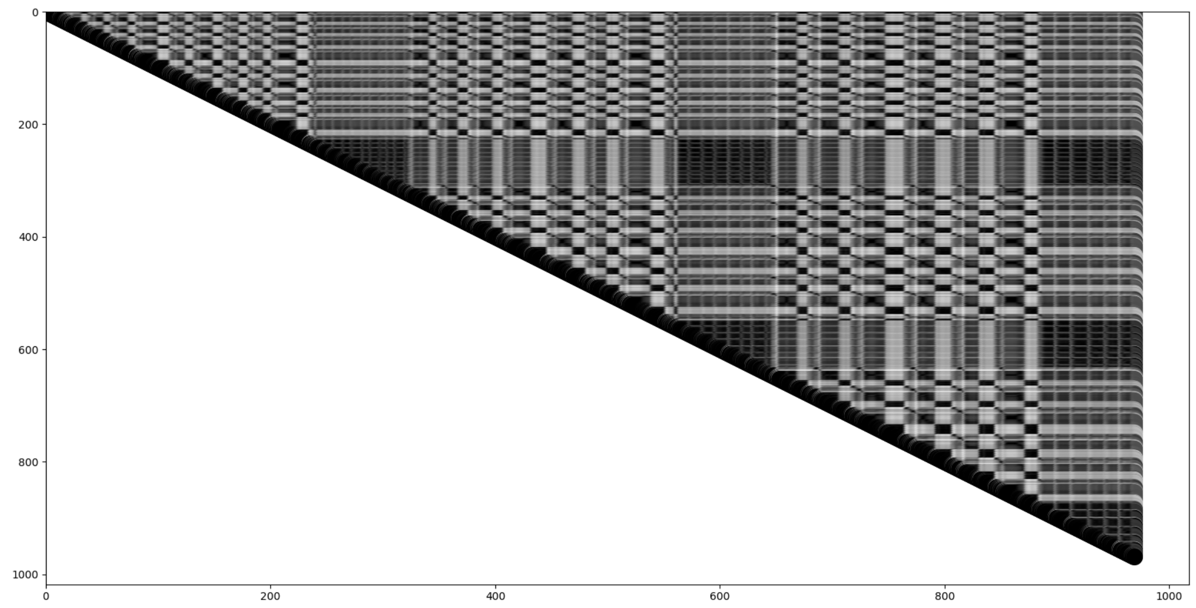

Figure 1: BB Similarity Matrix of bzip2

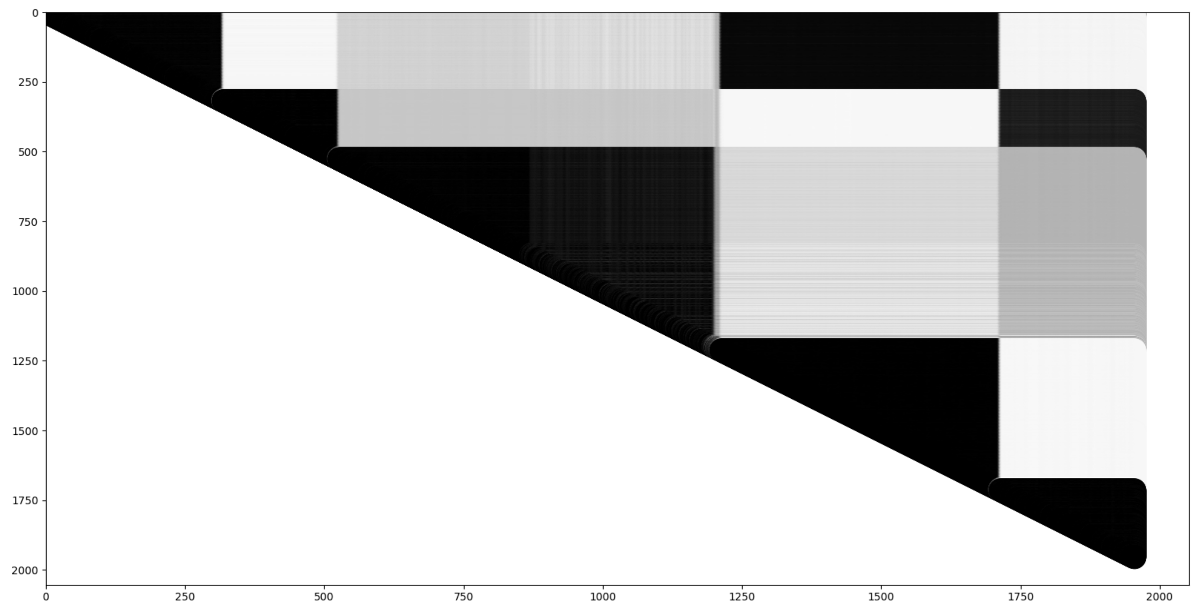

Figure 2: BB Similarity Matrix of astar (with lake.cfg)

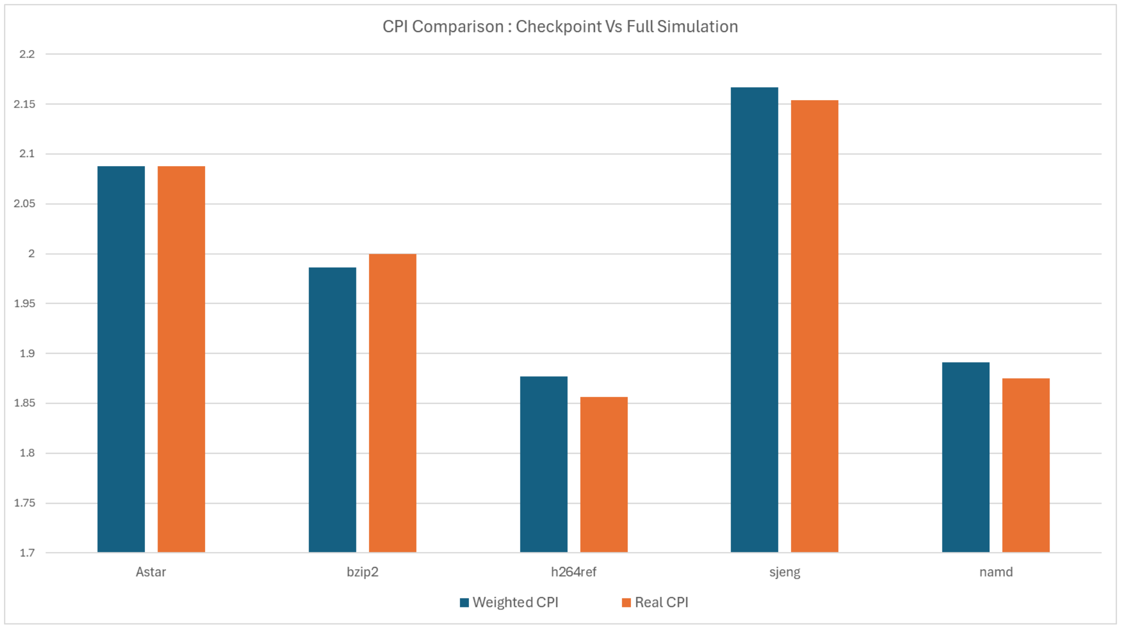

Figure 3: CPI comparison of the full simulation and simulating with checkpoints

were almost always lower (faster) than the real CPI with `h264ref` being the outlier. For the test data set used, these values are fairly close, with the worst CPI error rate being 1.11% and an average of 0.64%. This error rate should be higher and closer to 3% when using the reference data set as mentioned in [8]. Error rates are shown in Figure 4.

The error rate for Astar is extremely low, and likely not representative of actual benchmarks. This may have been caused due to the very small size of the "lake.cfg" test data, which was used to fit within the time constraints. It is still unclear what this means for checkpoint accuracy; further testing is required with a different configuration CPU model to verify if the simulation points work as expected. As for speedup, there is of course a significant reduction in simulation time due to only a tiny portion of the program running combined with the smaller test data. Table 2 shows that even with small data sets, detailed execution takes an extremely long time. The results don't include the data for `hmmer` since checkpoint creation took too long. `hmmer's` execution for BBV generation itself took 16 days. It would take another 14-16 days to generate its checkpoints, so it is omitted. The programs `h264ref` and `namd` had the largest speedup of 575x and 229x respectively. The other programs don't see as much of a speedup since this is directly related to the number of checkpoints and the weight of the specific simulation point. `bzip2` and `astar` had much shorter full simulation times, so the smaller speedup is expected.

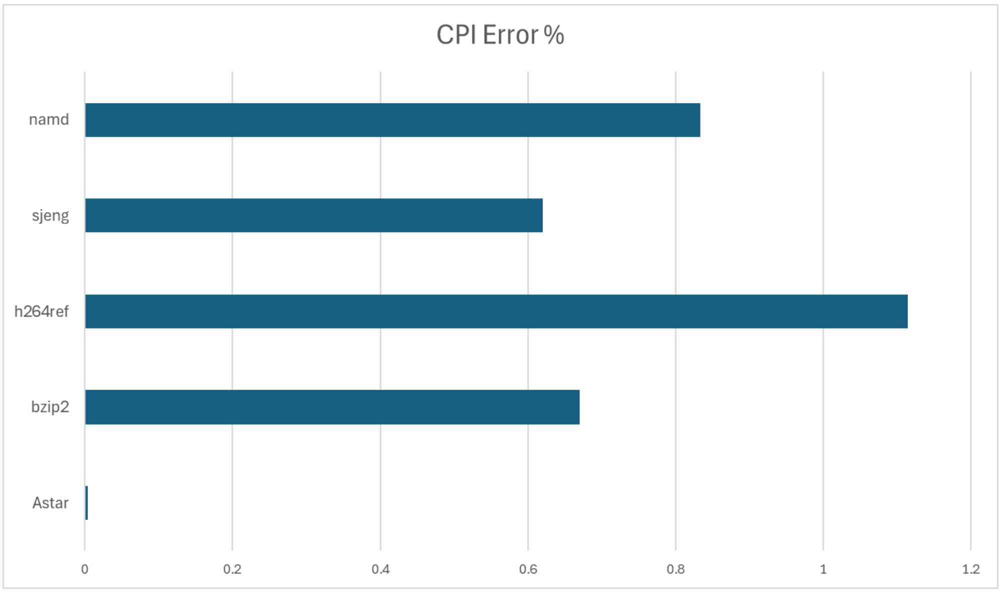

Figure 4: CPI error rates

Program	**SimPoint Time**	**Full exec Time**
Astar	122.75	16458.74
bzip2	146.23	7972.17
h264ref	128.28	73792.27
sjeng	59.11	13551.92
namd	163.32	86673.29

Table 2: Runtime of the programs during a full simulation and with checkpoints

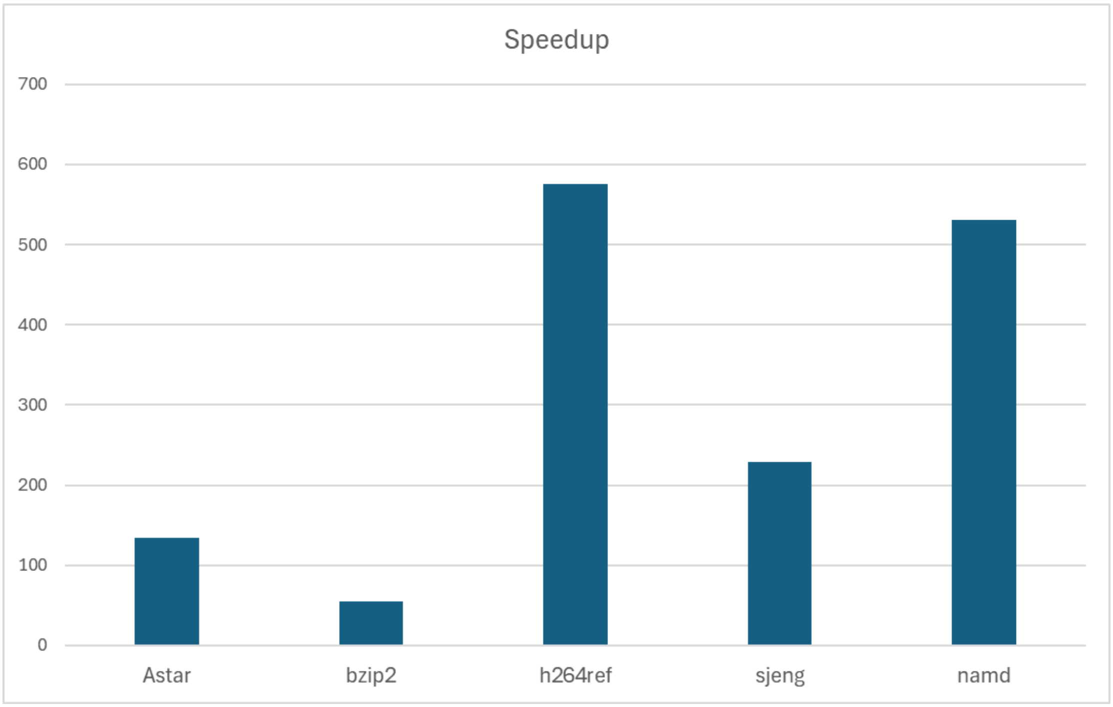

Figure 5: Speedup with checkpoints

Conclusion and Future Work

This report discussed the process of generating Gem5 checkpoints using the SimPoint tool. BBV similarity matrices help visualize the patterns in a program's behavior and give an idea of whether clustering is even possible. If the matrix has very few patterns, then the simulation points won't be representative of the entire execution, or possibly fail to find correct checkpoints. A small portion of the SPEC2006 benchmarks were checkpointed to perform a speed vs accuracy analysis. It was found that the checkpoints helped significantly speedup the execution, while only producing an average CPI error rate of 0.64%. This value was far smaller than the 3% mentioned in the SimPoint and Speed Vs Accuracy papers [4],[8]. However, those studies used a reference benchmark data set (SPEC2000), so the error rate here is expected to be small. One of the issues of sampling simulations is the time overhead required for creating basic block vectors and then re-running the simulation to generate the checkpoint states. For reference, the `namd` program took 2.5 days to complete the process of creating checkpoints. Another constraint was the system resources causing some of the simulations to fail after they have run for several days, with kernel panics and memory running out. Future work can be put towards minimizing these overheads to streamline the process even further. Simulators like Gem5 could implement sampling features so that users have an easier time with this process since tools like SimPoint have been outdated for a while, with minimal public support.

References

[1] Sabu, A., Patil, H., Heirman, W., and Carlson, T. E. Looppoint: Checkpoint-driven sampled simulation for multi-threaded applications. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2022), pp. 604–618.

[2] Sandberg, A., Nikoleris, N., Carlson, T. E., Hagersten, E., Kaxiras, S., and Black-Schaffer, D. Full speed ahead: Detailed architectural simulation at near-native speed. In *2015 IEEE International Symposium on Workload Characterization* (2015), pp. 183–192.

[3] Sherwood, T., Perelman, E., and Calder, B. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques* (2001), pp. 3–14.

[4] Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2002), ASPLOS X, Association for Computing Machinery, p. 45–57.

[5] Sherwood, T., Perelman, E., Hamerly, G., Sair, S., and Calder, B. Discovering and exploiting program phases. *IEEE Micro 23*, 6 (2003), 84–93.

[6] Van Biesbrouck, M., Calder, B., and Eeckhout, L. Efficient sampling startup for simpoint. *IEEE Micro 26*, 4 (2006), 32–42.

[7] Wunderlich, R. E., Wenisch, T. F., Falsafi, B., and Hoe, J. C. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2003), ISCA '03, Association for Computing Machinery, p. 84–97.

[8] Yi, J. J., Sendag, R., Lilja, D. J., and Hawkins, D. M. Speed versus accuracy trade-offs in microarchitectural simulations. *IEEE Transactions on Computers 56*, 11 (2007), 1549–1563.